PSP0201 Week 5 Writeup

Group name: VVannaCry

Members

ID	Name	Role
1211102056	Ahmad Fathi bin Amir	Leader
1211101999	Wong Wei Han	Member
1211101975	Muhammad Syahmi bin Mohd Azmi	Member

Day 16: Scripting - Help! Where is Santa?

Tools used: Kali Linux, OpenVPN, Python3, Nmap

Walkthrough

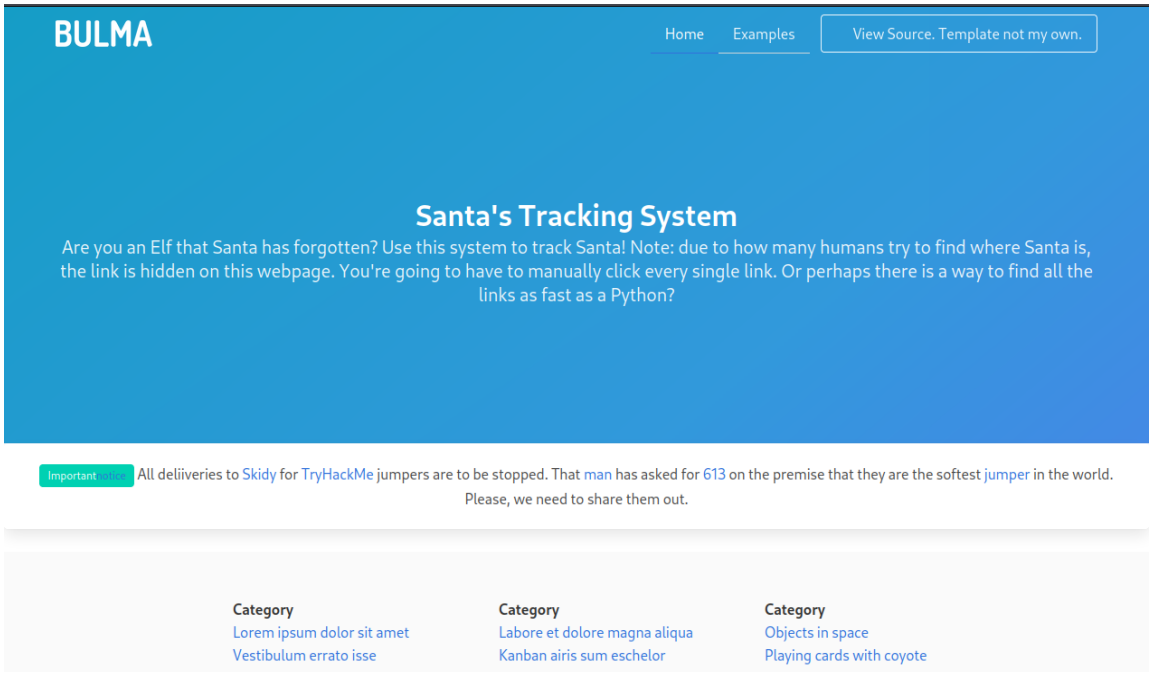
Question 1

As the question asks, we can use our previous knowledge about nmaps to get the port number of the target website.

```
┌──(1211101975㉿kali)-[~]
└─$ nmap 10.10.40.242
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-12 09:53 EDT
Nmap scan report for 10.10.40.242
Host is up (0.37s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 47.58 seconds
```

Question 2

The templates used can be seen on the top left corner of the target website



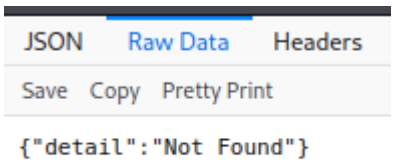
Question 3

We can look at the page source and look through the links given and there's gonna be one that sticks out like a sore thumb

```
<div class="column is-3">
    <h2><strong>Category</strong></h2>
    <ul>
        <li><a href="#">Labore et dolore magna aliqua</a></li>
        <li><a href="#">Kanban airis sum eschelor</a></li>
        <li><a href="http://machine_ip/api/api_key">Modular modern free</a></li>
        <li><a href="#">The king of clubs</a></li>
        <li><a href="#">The Discovery Dissipation</a></li>
        <li><a href="#">Course Correction</a></li>
        <li><a href="#">Better Angels</a></li>
    </ul>
</div>
```

Question 4

We can enter the previous link to the api to get the raw data in the browser



Question 5&6

We can create a python script to automatically insert the numbers to the link and execute it to get the location and also the correct api key

Thought Process:

After grasping the knowledge of python programming the day before, we can easily do the challenge. We get the port first by doing a normal nmap scan. Then we can get the correct directory of the api by clicking the links available in the target website or we can look at the page source that is available in browsers nowadays. After a very quick glance into the page source and getting the correct link to the api, we can now build our python script according to the link. By the given link we know that we mostly are going to use html based commands in our scripts. The creation of the script is quite simple and easy as we have learned it in the previous day. Then, after executing the script we now wait for it to automate odd numbers from 0-100 for the api keys so we can just sit back while waiting for the one that succeeded guessing the correct api key. After knowing the right one we can now see where santa’s current location and that's another challenge done

Day 17: Reverse Engineering - ReverseELFneering

Tools used: Kali Linux, radare2

Walkthrough:

Question 1

For Byte it will be **1**, Word will be **2**, Double Word will **4**, Quad will be **8**, Single Precision will be **4** and Double Precision will be **8**

3. Register me this, register me that...

The core of assembly language involves using registers to do the following:

- Transfer data between memory and register, and vice versa
- Perform arithmetic operations on registers and data
- Transfer control to other parts of the program Since the architecture is x86-64, the registers are 64 bit and Intel has a list of 16 registers:

Initial Data Type	Suffix	Size (bytes)
Byte	b	1
Word	w	2
Double Word	l	4
Quad	q	8
Single Precision	s	4
Double Precision	l	8

Question 2

It will be **aa** to analyse the program in radare2

Time to see what's happening under the hood! **Run** the command `r2 -d ./file1`

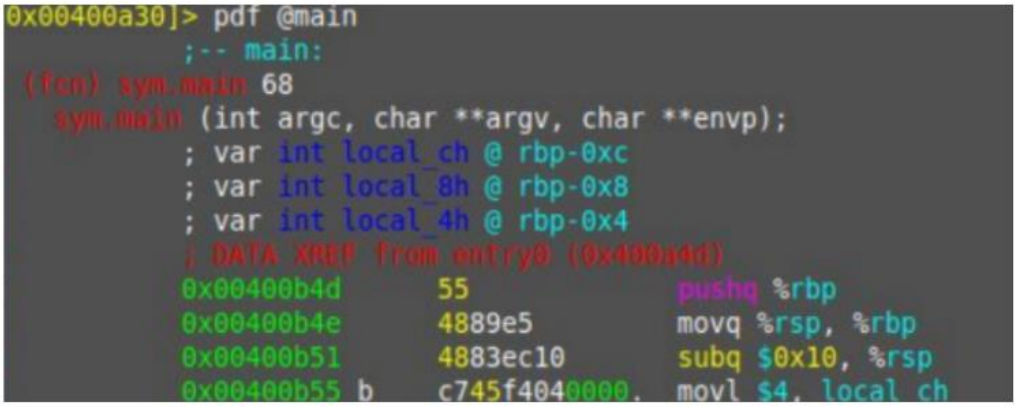
This will open the binary in debugging mode. Once the binary is open, one of the first things to do is ask r2 to analyze the program, and this can be done by typing in: `aa`

Note, when using the `aa` command in radare2, this may take between 5-10 minutes depending on your system.

Question 3

The command **db** will set the breakpoints

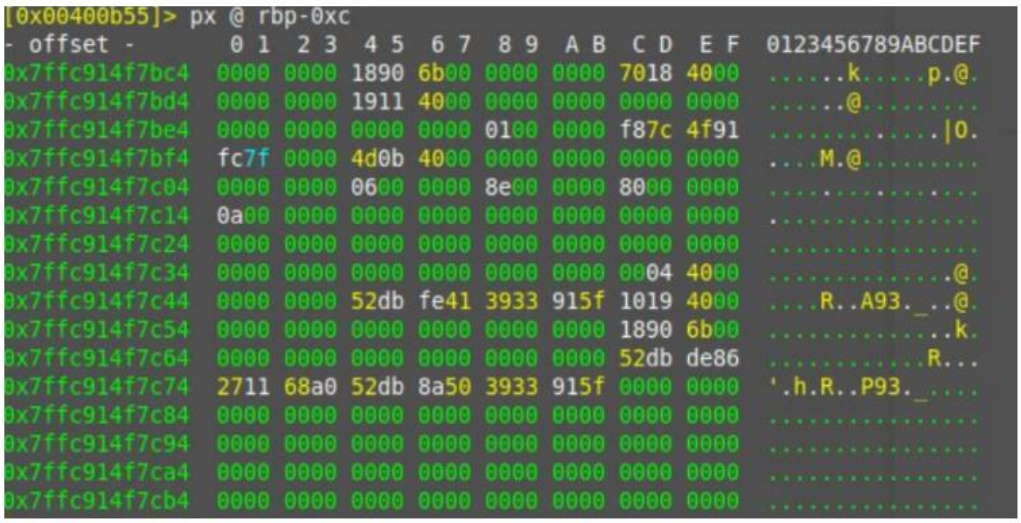
A **breakpoint** specifies where the program should stop executing. This is useful as it allows us to look at the state of the program at that particular point. So let's set a breakpoint using the command `db` in this case, it would be `db 0x00400b55` To ensure the breakpoint is set, we run the `pdf @main` command again and see a little **b** next to the instruction we want to stop at.



Question 4

The command **dc** will execute the program until it hits the breakpoint

Running `dc` will execute the program until we hit the breakpoint. Once we hit the breakpoint and print out the main function, the rip which is the current instruction shows where execution has stopped. From the notes above, we know that the **mov** instruction is used to transfer values. This statement is transferring the value 4 into the `local_ch` variable. To view the contents of the `local_ch` variable, we use the following instruction `px @memory-address` In this case, the corresponding memory address for `local_ch` will be `rbp-0xc` (from the first few lines of **@pdf main**) This instruction prints the values of memory in hex:



Question 5

Start analysing the **challenge1** file with the **aa** command

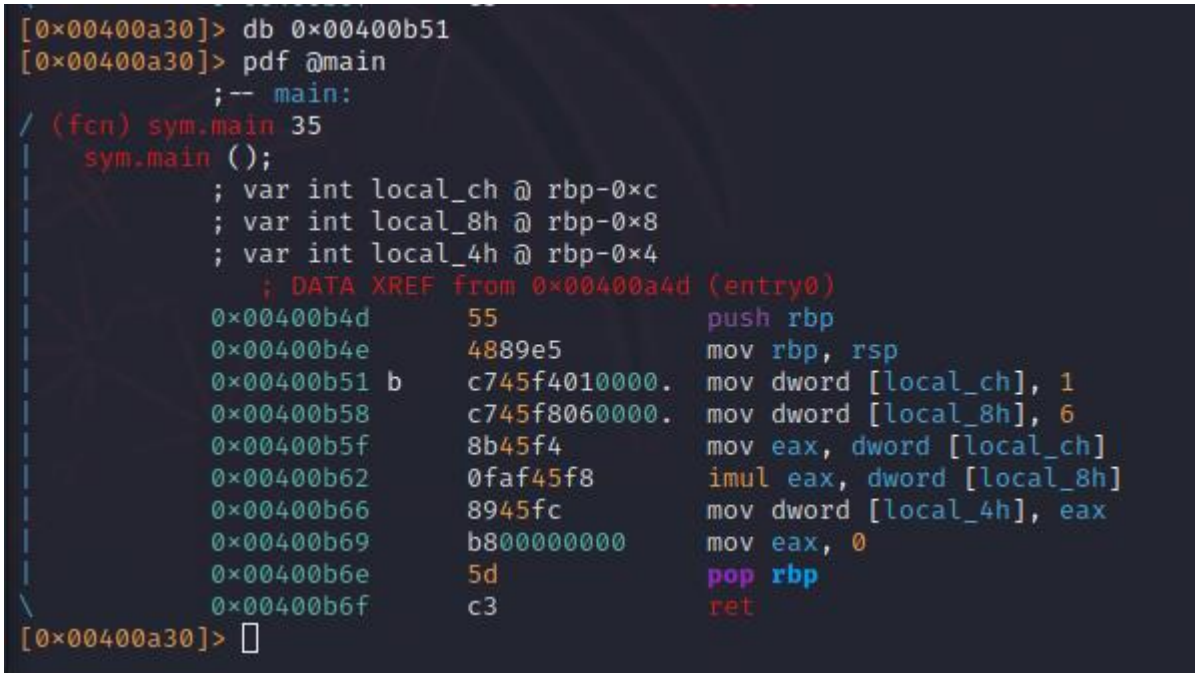
```
elfmceager@tbfc-day-17:~$ ls
challenge1  file1
elfmceager@tbfc-day-17:~$ r2 -d challenge1
Process with PID 1665 started...
= attach 1665 1665
bin.baddr 0x00400000
Using 0x400000
Warning: Cannot initialize dynamic strings
asm.bits 64
[0x00400a30]> aa
[ WARNING : block size exceeding max block size at 0x006ba220
[+] Try changing it with e anal.bb.maxsize
 WARNING : block size exceeding max block size at 0x006bc860
[+] Try changing it with e anal.bb.maxsize
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00400a30]>
```

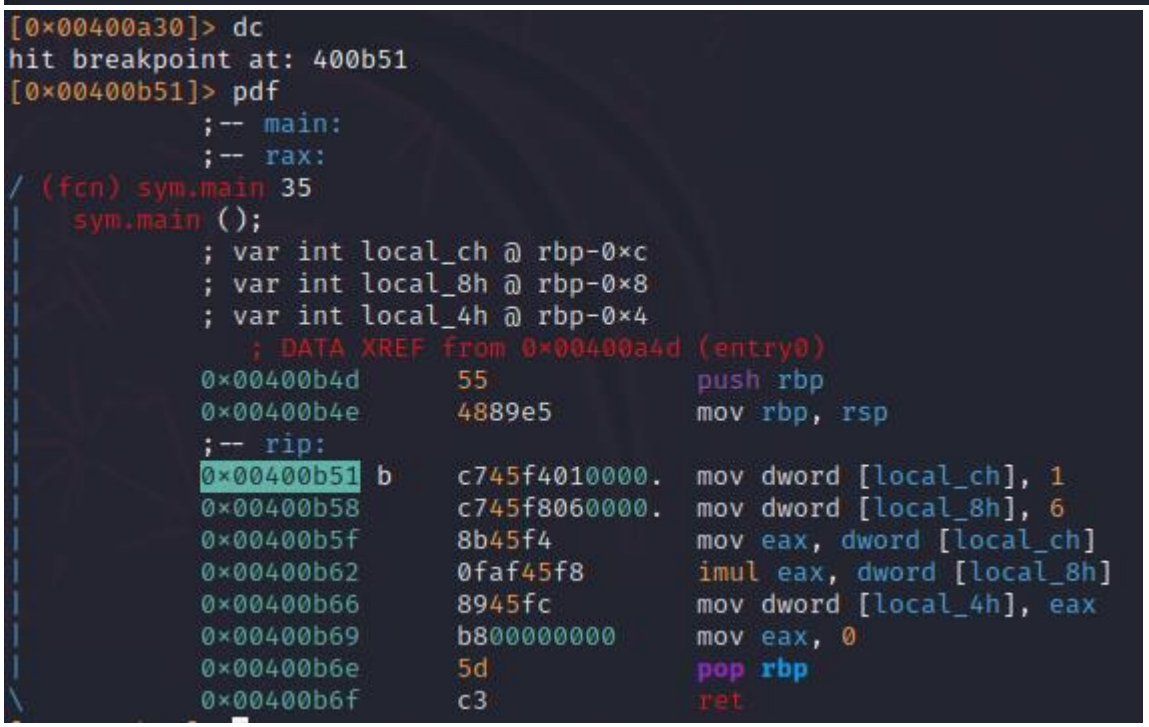
After analysing is done, we find main function and examine it with **pdf @main** command. We can see the **local_ch** value is **1**

```
[0x00400a30]> afl | grep main
0x00400b4d    1 35           sym.main
0x00400de0   10 1007 → 219   sym.__libc_start_main
0x00403840   39 661  → 629   sym._nl_find_domain
0x00403ae0  308 5366 → 5301  sym._nl_load_domain
0x00415ef0    1 43           sym._IO_switch_to_main_get_area
0x0044ce10    1 8            sym._dl_get_dl_main_map
0x00470430    1 49           sym._IO_switch_to_main_wget_area
0x0048f9f0    7 73   → 69    sym._nl_finddomain_subfreeres
0x0048fa40   16 247  → 237   sym._nl_unload_domain
[0x00400a30]> pdf @main
            ;-- main:
/ (fcn) sym.main 35
|   sym.main ();
|           ; var int local_ch @ rbp-0xc
|           ; var int local_8h @ rbp-0x8
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x00400a4d (entry0)
|           0x00400b4d      55              push rbp
|           0x00400b4e      4889e5          mov rbp, rsp
|           0x00400b51      c745f4010000.   mov dword [local_ch], 1
|           0x00400b58      c745f8060000.   mov dword [local_8h], 6
|           0x00400b5f      8b45f4          mov eax, dword [local_ch]
|           0x00400b62      0faf45f8        imul eax, dword [local_8h]
|           0x00400b66      8945fc          mov dword [local_4h], eax
|           0x00400b69      b800000000      mov eax, 0
|           0x00400b6e      5d              pop rbp
\           0x00400b6f      c3              ret
```

Question 6

Set a breakpoint at **0x00400b51** and use the **dc** command

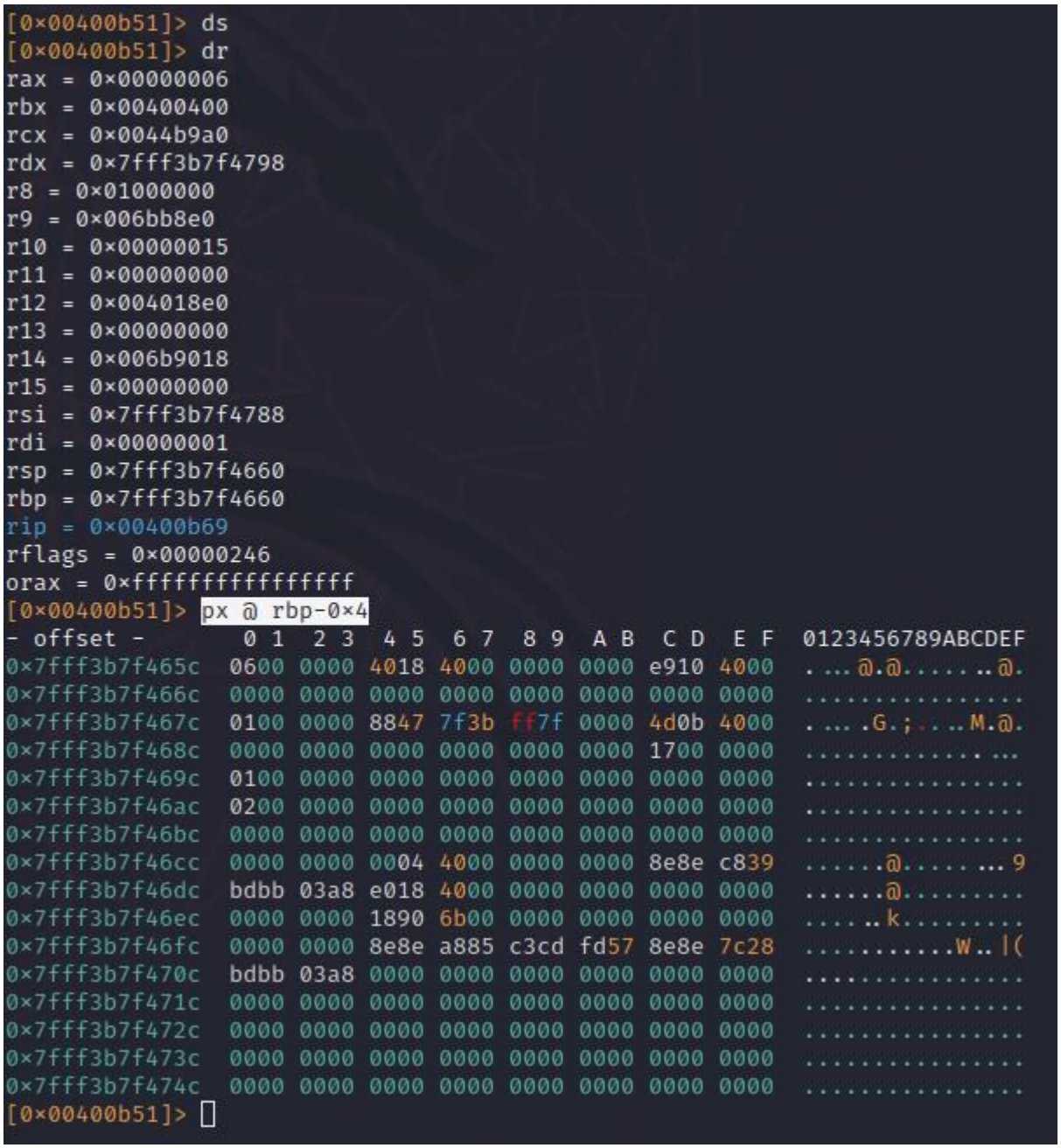




Moving to the imull instruction with **ds 4** command and running the **dr** command will show the value for the **eax** which is **6**

Question 7

By moving to the next instruction with **ds** command and **px @ rbp-0x4** to show the hex value, we can see the value for **local_4h** is **6**



Thought Process:

Data types contains sizes of how much bytes it takes, **Byte** is **1**, **Word** is **2**, **Double Word** is **4**, **Quad** is **8**, **Single Precision** is **4** and **Double Precision** is **8**. The command to analyse a program in radare2 is **aa**, **db** for setting a breakpoint and **dc** for executing the program until the breakpoint. We first send the **challenge1** file into radare2 debugger with **r2 -d ./challenge1** and analyse it with the **aa** command. Then do **pdf @main** and it reveals that one of the instruction shows the value for **local_ch** which is **1.** Then set the breakpoint at **0x00400b51** followed by **dc** to execute until the breakpoint. Move to the mull instruction with **ds** 4 and use **dr** to show the **eax** value which is **6**. Then do **ds** again to move to the **local_4h** instruction then do **px @ rbp-0x4** to show the hex value and we can see the value for **local_4h** is **6**.

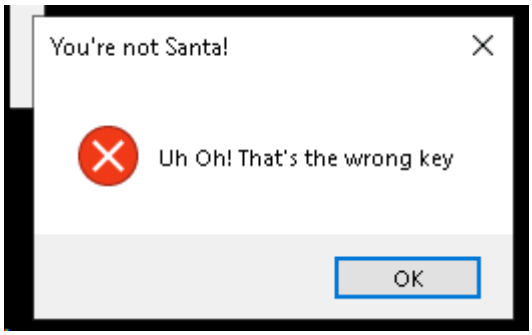
Day 18: Reverse Engineering – The Bits of Christmas

Tools used: Kali

Solution/walkthrough:

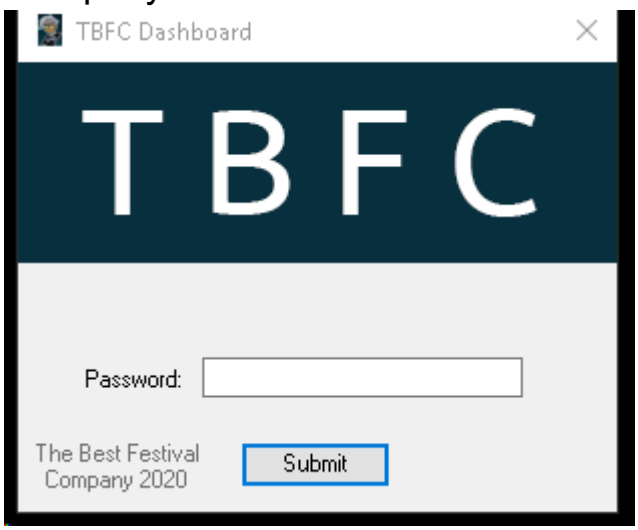
Question 1

This will pop out if you entered the wrong password



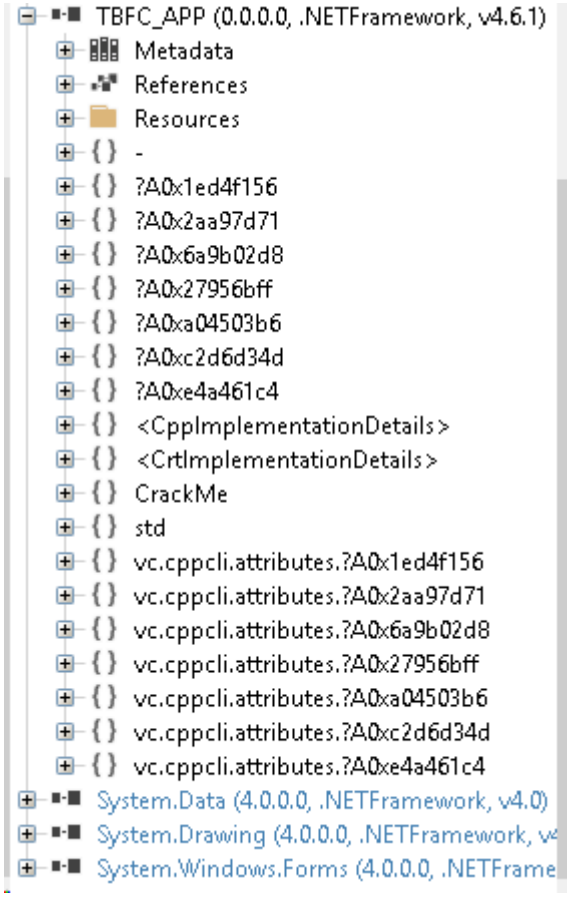
Question 2

At the bottom left, we are able to tell TBFC is “The Best Festival Company”



Question 3

The “CrackMe” module stands out the most compared to the other modules.



Question 4

Under the module, there is 2 forms but we wanted to know more about “MainForm”, not “AboutForm".

Question 5

Based on Question 4, we are looking for a function after the “submit” button is pressed.

```
private unsafe void buttonActivate_Click(object sender, EventArgs e)
{
    IntPtr value = Marshal.StringToHGlobalAnsi(textBoxKey.Text);
    sbyte* ptr = (sbyte*)System.Runtime.CompilerServices.Unsafe.AsPointer(ref <Module>.??_C@_0BB@IKKDFEPG@santapassword321@);
    void* ptr2 = (void*)value;
    byte b = *(byte*)ptr2;
    byte b2 = 115;
    if ((uint)b >= 115u)
    {
        while ((uint)b <= (uint)b2)
        {
            if (b != 0)
            {
                ptr2 = (byte*)ptr2 + 1;
                ptr++;
                b = *(byte*)ptr2;
                b2 = (byte)(*ptr);
                if ((uint)b < (uint)b2)
                {
                    break;
                }
                continue;
            }
            MessageBox.Show("Welcome, Santa, here's your flag thm{046af}", "That's the right key!", MessageBoxButtons.OK, MessageBoxIcon.Asteri
            return;
        }
    }
    MessageBox.Show("Uh Oh! That's the wrong key", "You're not Santa!", MessageBoxButtons.OK, MessageBoxIcon.Hand);
}
```

Question 6

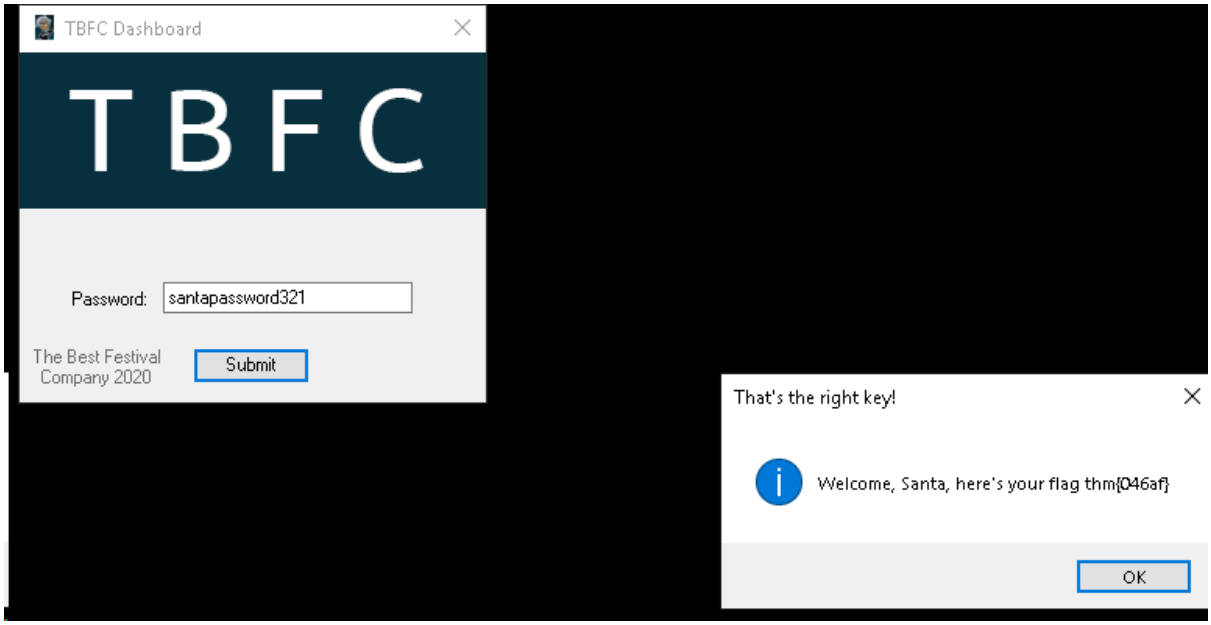
By looking through the code, we can find Santa’s password

```
private unsafe void buttonActivate_Click(object sender, EventArgs e)
{
    IntPtr value = Marshal.StringToHGlobalAnsi(textBoxKey.Text);
    sbyte* ptr = (sbyte*)System.Runtime.CompilerServices.Unsafe.AsPointer(ref <Module>.??_C@_0BB@IKKDFEPG@santapassword321@);
    void*
    byte
    byte
```

```
internal static <CppImplementationDetails>.$ArrayType$$$BY0BB@$$CBD global::<Module>.??_C@_0BB@IKKDFEPG@santapassword321@
```

Question 7

By entering the password, we captured the flag.



Thought Process/Methodology:

Firstly, we will have to launch Remmina and enter the Ip Address given. Then, we enter the username and password given. Once we're in, we open TBFC_APP and ILSpy app. In ILSpy, we go to file>open>TBFC_APP to decompile the codes. By looking through the modules, we see "CrackMe" stands out the most amongst the others. Under it, we go into the "MainForm" and start scrolling. We then come across a function called "buttonActivate_Click" which means what will happen if we enter the password and hit "submit" in the TBFC_APP. Looking through, we manage to find Santa's password. By entering Santa's password into the TBFC_APP, we are able to capture the flag.

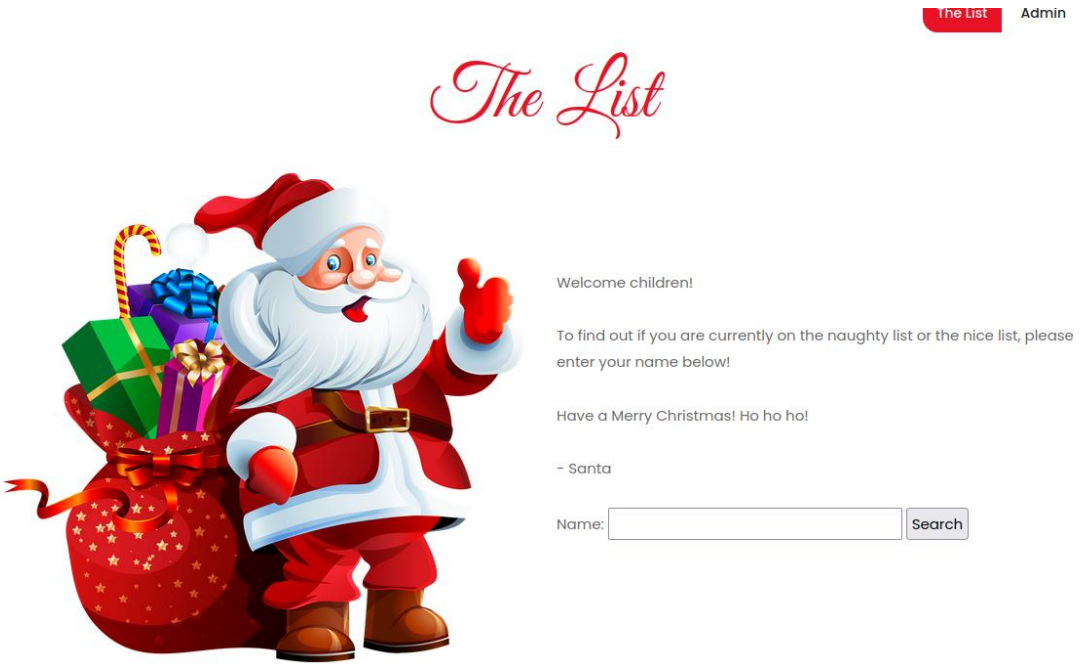
Day 19: Web Exploitation - The Naughty or Nice List

Tools Used: Kali Linux, OpenVPN

Walkthrough

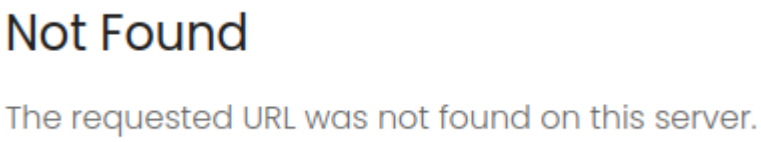
Question 1

After entering the target website we can enter the names into the search bar to check if its either on the naugthy or nice list



Question 2

The displayed message after trying to fetch the root of the site is like so



Question 3

Trying to change the port number to the default http port is going to put an error message like so

```
Failed to connect to list.hohoho port 80: Connection refused
```

Question 4

Trying to be creative and use an ssh port is also going to fail as a message is going to pop up like this

```
Recv failure: Connection reset by peer
```

Question 5

Changing the hostname to localhost/127.0.0.1 will output another failure

```
Your search has been blocked by our security team.
```

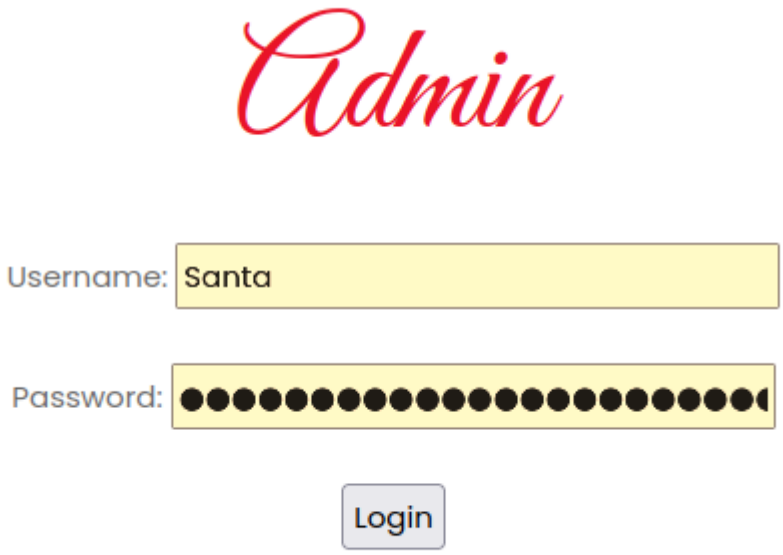
Question 6

After changing the hostname to any other readily available domain names (were going to use localtest.me here) we're going to be presented with a successful bypass and get Santa's password

```
Santa,

If you need to make any changes to the Naughty or Nice list, you need
to login.

I know you have trouble remembering your password so here it is: Be
good for goodness sake!

- Elf McSkidy
```

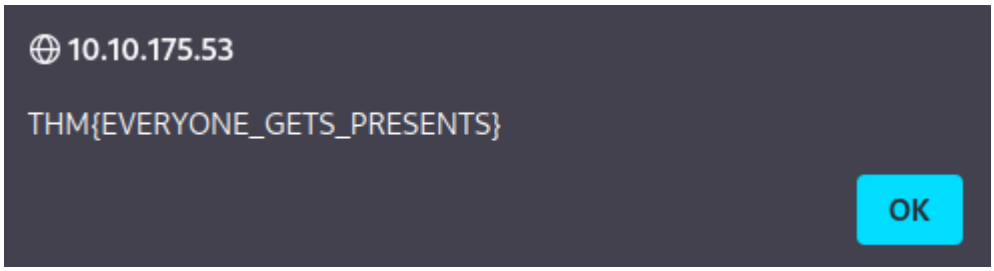
Question 7

Then we try to guess the correct username and use the password previously to login as Santa like so



Then we are greeted by a new page as we've login as Santa

The only thing left we need to do now is to delete the naughty list and were going to get the flag



Thought Process:

The challenge for this day is specialised in ssrf vulnerability which we only need to alter the url in order to bypass the target ip. After getting the target ip we can explore the webpage, for our context we can enter any names onto the search bar so that we can look at the url for any changes. We know that it's not using a very high level domain. By this way we can try to get into the root of the site or maybe even try to change the port numbers so that it allows us inside. If that doesn't do anything we could try and change the hostname to any available domain names we know i.e localhost. Seems like this is a tough nut to crack as the dev put in some effort for the back end. Well looks like we can resort to linking to a different subdomain and were going to use localtest.me and there we go a hint on the login credentials of santa in the website. The only thing left for us to do is go to the admin login page and guess the username and then we can delete the naughty list so that everyone gets a present and then finish off the day with the flag

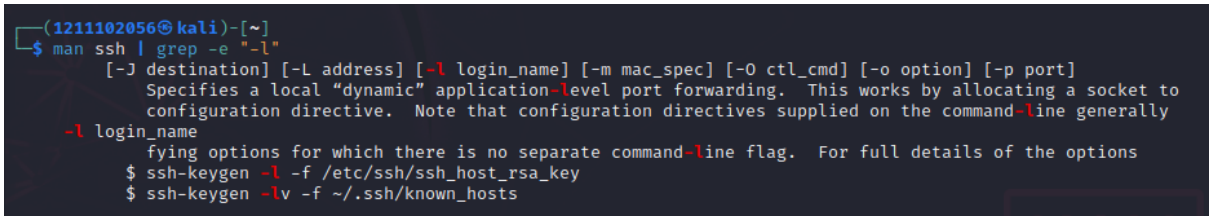
Day 20: Blue Teaming - PowershELlF to the rescue

Tools used: Kali, ssh, PowerShell

Walkthrough:

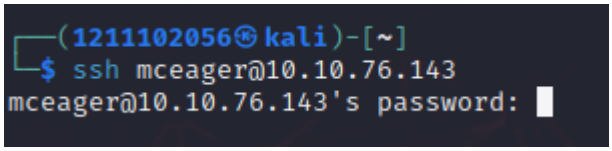
Question 1

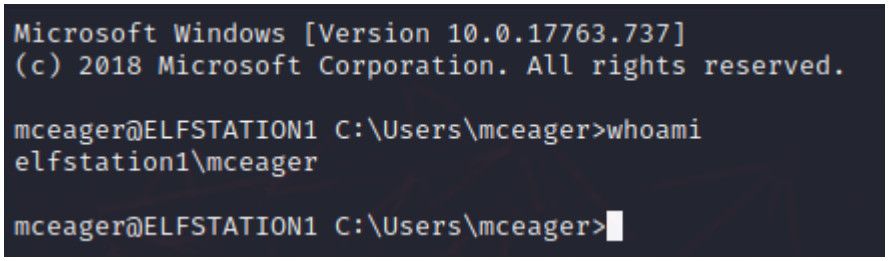
We can see that **-l** parameter do login name for ssh



Question 2

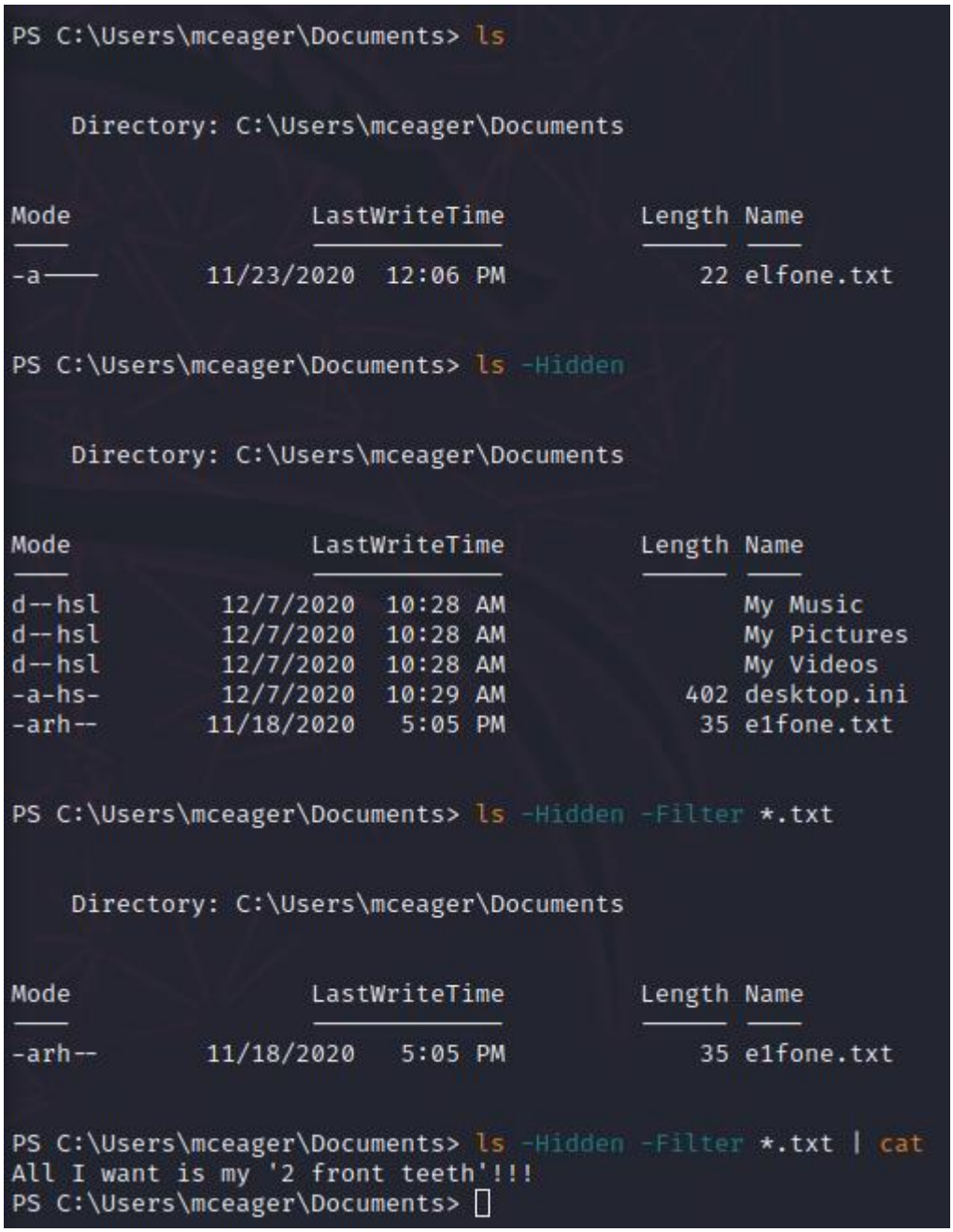
We first connect to the ssh with the provided password which is **r0ckStar!**





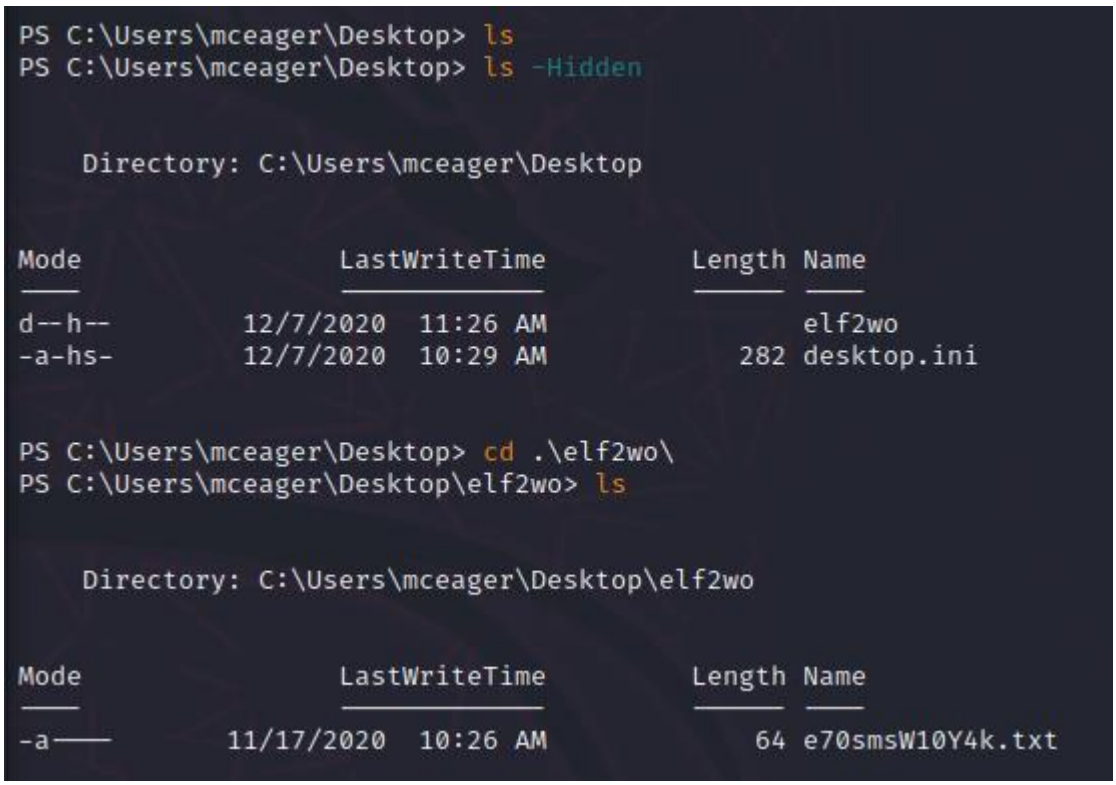
We then launch **Powershell** and set the location to **Documents**

We first list the file content of the directory and the hidden files with **ls** command, which is the alias command for **Get-ChildItem**. There's the **visible** elfone.txt and the **hidden** elfone.txt, concatenate the **hidden** elfone.txt with **cat**, that is an alias command for **Get-Content**. It shall reveal that elf 1 wants **2 front teeth**

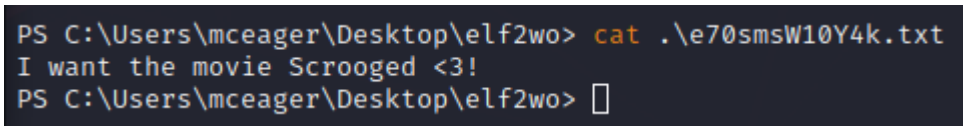


Question 3

We change the working directory to **Desktop** with **cd**, Which is the alias command for **Set-Location.** Then list out the content, we can see there's a **hidden** folder for Elf 2.



Concatenating the txt file in the elf 2 reveals that Elf 2 wants the movie **Scrooged**



Question 4

Change the Directory to **C:\Windows** , to find the **third** elf folder and change the directory to that, we do **ls -Recurse -Hidden -Directory -Filter '*3*' -ErrorAction SilentlyContinue | cd**. We can see the folder is called **3lfthr3e**

```
PS C:\Windows> ls -Hidden -Directory -filter '*3*' -Recurse -ErrorAction SilentlyContinue


    Directory: C:\Windows\System32


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d--h--        11/23/2020   3:26 PM                3lfthr3e


PS C:\Windows>
PS C:\Windows> ls -Hidden -Directory -filter '*3*' -Recurse -ErrorAction SilentlyContinue | cd
PS Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32\3lfthr3e>
```

Question 5

We list the hidden files which contains 2 txt files. We then Count how many words for the first file with **cat .\1.txt | measure -Words** Where **measure** is the alias command for **Measure-Object**.
It reveals that it is **9999** words.

```
PS Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32\3lfthr3e> ls -Hidden


    Directory: C:\Windows\System32\3lfthr3e


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-arh--        11/17/2020  10:58 AM          85887 1.txt
-arh--        11/23/2020   3:26 PM       12061168 2.txt


PS Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32\3lfthr3e> cat .\1.txt | measure -Word

Lines Words Characters Property
-----  -----  ----------  --------
       9999


PS Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32\3lfthr3e>
```

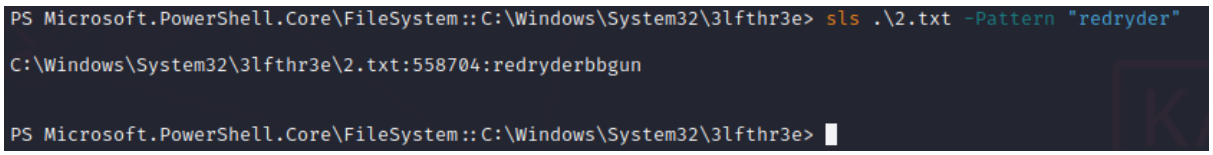
Question 6

To get the words from the **index** of **551** and **6991** we do **(cat .\1.txt)[551]; (cat .\1.txt)[6991]**. Which should give us **Red Ryder**.

```
PS Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32\3lfthr3e> (cat .\1.txt)[551]; (cat .\1.txt)[6991]
Red
Ryder
```

Question 7

To find the other half of the answer for elf 3 in the 2.txt, We use **sls**, Which is the alias command for **Select-String**, We do **sls .\2.txt -Pattern “redryder”**. Which gives us **redryderbbgun**



Thought Process:

We were given the machine ip, Username which is **mceager** and the password **r0ckStar!** to connect to **ssh**. We then launch **PowerShell** after connecting to it and set the location to the **Documents** directory. When showing the list for the directory, we found a elfone.txt but doesn't not give much information when we concatenate it. When showing the list that only shows hidden file, we found that there's a hidden file called elfone.txt as well, concatenating that hidden file reveals that elf 1 wants **2 front teeth**. We then change directory to the **Desktop**, showing the list of hidden files reveals that there is a **hidden folder** for elf 2, changing the directory into that folder reveals a txt file that concatenate into elf 2 wanting a movie called **Scrooged**. To find elf 3 folder, we need to do a listing command that executes recursive search in the **Windows** folder, that also only shows **Hidden** Files/Directories, only show **Directory, filtering** it where it only takes Directories that contains '**3**' in the name and to **hide** the **errors** that popped up. With that, it reveals the elf 3 folder named **3lfthr3e** that contains **2 .txt** file. When counting how many words in the first text file, it shows that it contains **9999** words. Then we execute a command where it only prints the line starting the index of **551** and **6991**, revealing the word **Red** and **Ryder** respectively. Since it is only half complete, We then find the **other half** in the **2nd txt file**. By executing a command that takes the pattern of '**redryder**', it reveals the full phrase of '**redryderbbgun**' or **red ryder bb gun**' when we add the spaces.