PSP0201
WEEKLY WRITE UP
WEEK 5
GROUP 7

Group Name : Sang Haekeo (The Hackers)

Sang
- Taken from a Malay word , sang , meaning ‘the’

Haekeo
- Taken from a Korean word , 해커 , meaning ‘hacker’

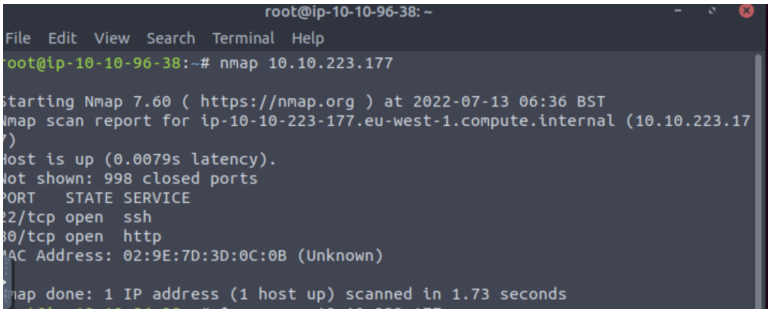
ID	NAME	ROLE
1211102162	AMILIA NADZEERA BINTI BAHARUDIN	Leader
1211100930	KU NAJWA SYAUQINA BINTI KU AZRIN	Member
1211101693	SAVITHA MURUGUMUNISEGARAN	Member

Day 16: Scripting ; Help! Where is Santa?

Question 1: What is the port number for the web server?

The port number can be found using nmap and syntax , host number is shown.

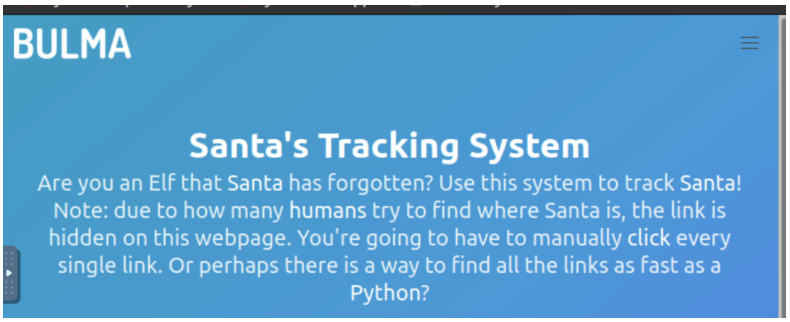
Answer 1: 80



Question 2: What templates are being used?

For this , we checked the top left of the website

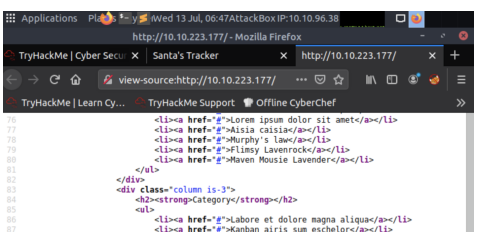
Answer 2: BULMA



Question 3: Without using enumeration tools such as Dirbuster, what is the directory for the API?

Open the developers tools in your browser , then open your elements tab. Scroll down and check the footer to find the HTML tag

Answer 3: /api/

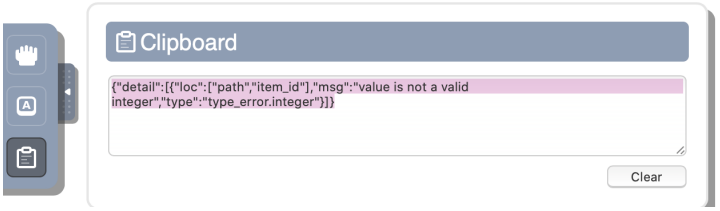


Question 4: Go to the API endpoint. What is the Raw Data returned if no parameters are entered?

the raw data returned was retrieved from the endpoint of the API.

Answer 4: {"detail":[{"loc":["path","item_id"],"msg":"value is not a valid integer","type":"type_error.integer"}]}

{"detail":[{"loc":["path","item_id"],"msg":"value is not a valid integer","type":"type_error.integer"}]}



Question 5: Where is Santa right now?

Answer 5: Winter Wonderland , Hyde Park , London

Question 6: Find out the correct API key. Remember, this is an odd number between 0-100.

Answer 6: 57

For both questions , we had to try our luck and key in an odd

ta is right now. 57 is nswer for Question 6 and

Day 17 : Reverse Engineering ; ReverseELFngineering

Question 1: Match the data type with the size in bytes

For this question , we followed the tale provided in TryHackMe.

Answer 1:

- Byte - 1
- Word - 2
- Double Word - 4
- Quad - 8
- Single Precision - 4
- Double Precision - 8

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: What is the command to analyse the program in radare2?

The answer was provided in the text shown below.
Answer 2: pdf@main

As seen here, there actually is a function at main. Let's examine the assembly code at main by running the command `pdf @main` Where **pdf** means **p**rint **d**isassembly **f**unction. Doing so will give us the following view:

Question 3: What is the command to set a breakpoint in radare2?

The answer was provided in the text shown below.

Answer 3: db

The line starting with **sym.main** indicates we're looking at the **main functio**n. The next 3 lines are used to represent the variables stored in the function. The second column indicates that they are integers(*int*), the 3rd column specifies the name that `r2` uses to reference them and the 4th column shows the actual memory location.

The first 3 instructions are used to allocate space on that stack (ensures that there's enough room for variables to be allocated and more). We'll start looking at the program from the 4th instruction (`movl $4`). We want to analyse the program while it runs and the best way to do this is by using **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: What is the command to execute the program until we hit a breakpoint?

The answer was provided in the text shown below.

Answer 4: dc

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:

```
[0x00400b55]> px @ rbp-0xc
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
```

Question 5: What is the value of local_ch when its corresponding movl instruction is called (first if multiple)?

The value is 1 as the instruction stated mov dwod [local_ch],1
Answer 5: 1

```
elfmceager@tbfc-day-17:~$ r2 -d ./challenge1
Process with PID 1615 started...
= attach 1615 1615
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]> 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
[0x00400a30]>
```

Question 6: What is the value of eax when the imull instruction is called?
Answer 6: 6

```
elfmceager@tbfc-day-17:~$ r2 -d ./challenge1
Process with PID 1615 started...
= attach 1615 1615
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]> 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
[0x00400a30]>
```

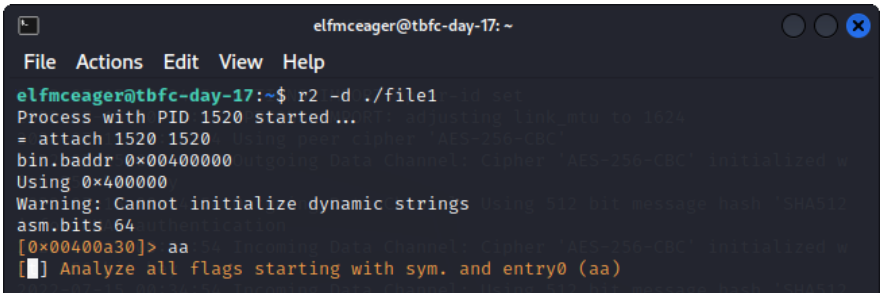
Question 7: What is the value of local_4h before eax is set to 0?

Answer 7: 6


```
elfmceager@tbfc-day-17:~$ r2 -d ./challenge1
Process with PID 1615 started...
= attach 1615 1615
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]> 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
[0x00400a30]>
```

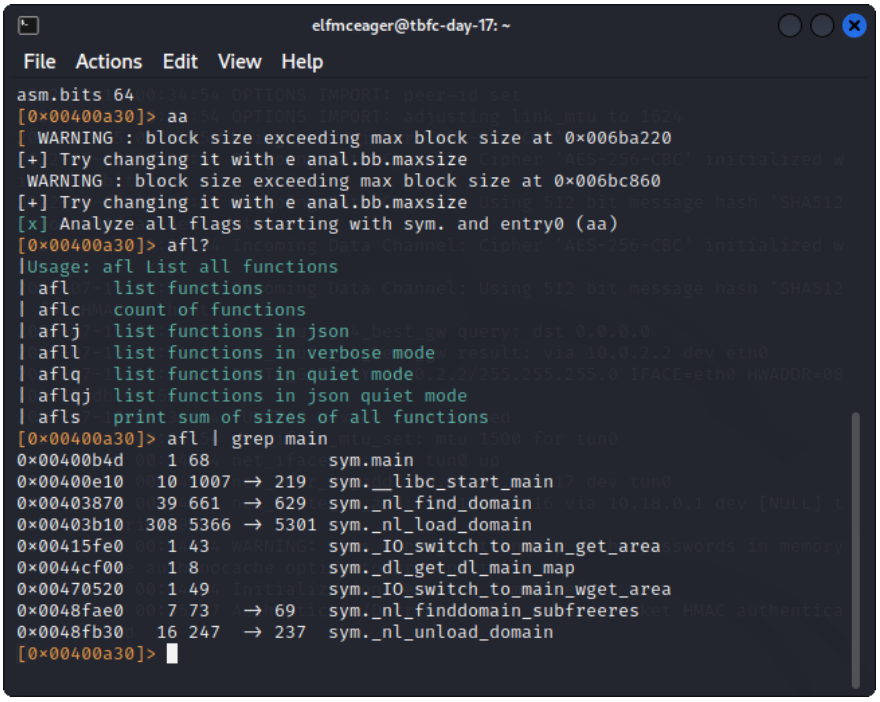
[Day 17] – Reverse Engineering – Reverse ELFneering

Log into the instance with the given credentials. Opens the binary files in debugging mode using command r2 -d ./file1



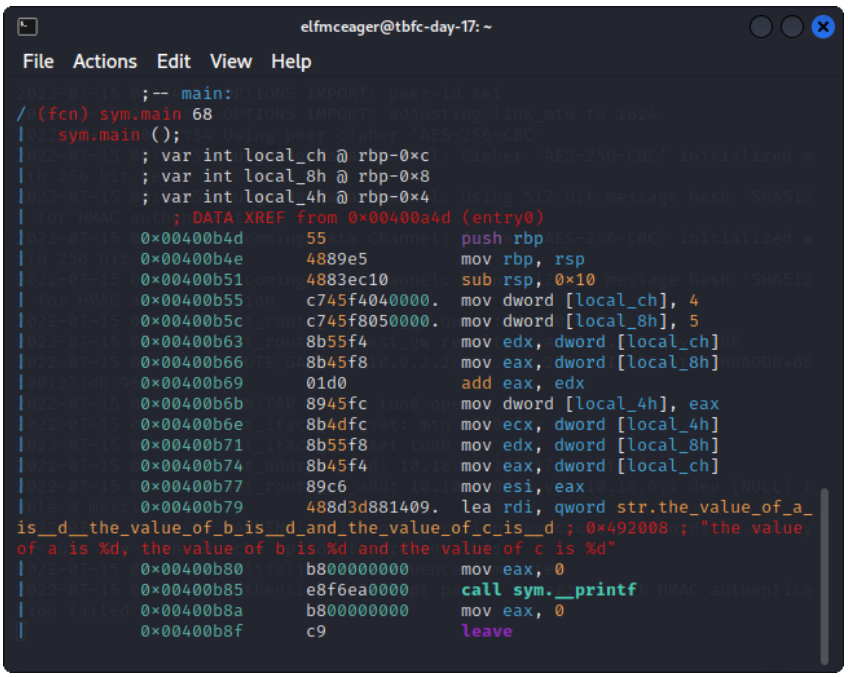
Once the analyzing completes we can seek out the entry point to the file.

afl | grep main



Now we can see where in memory the program starts from. Since we found a function here we can examine it with the pdf (Print Disassembly Function) command.

pdf @main



We can set a breakpoint with db.

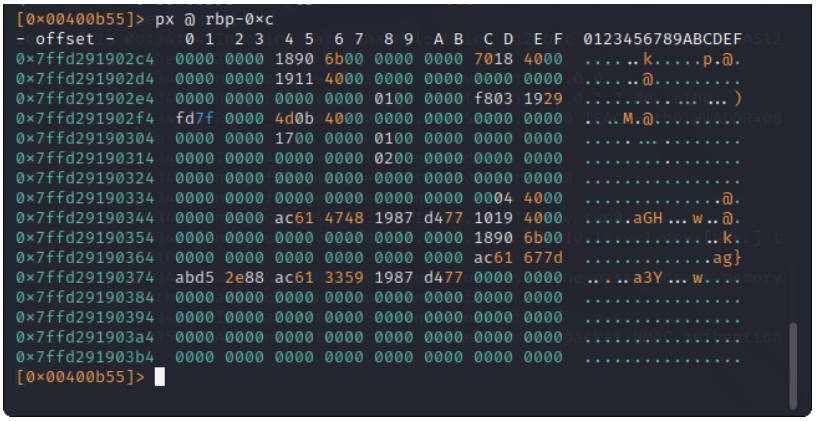
db 0x00400b55

After setting our break point we, we can again run pdf @main and now we will see a lowercase b right after our break point location

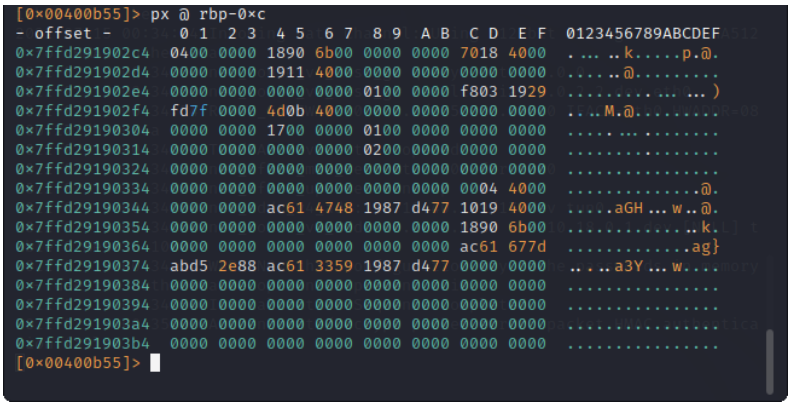
Next, we use the dc command to run the program until it hits the break point. If we run pdf, the memory address at our breakpoint will be highlighted.

Next, to view the contents of, in our case, the local_ch variable, we will run px @memory-address

px @rbp-0cx



But we do not see our variable 4 so we will step forward using ds. Then we will run px @rbp-0cx again.



And we now can see the variable 4 returned. Going ahead to the challenge file, we follow the same initial steps

```
elfmceager@tbfc-day-17:~$ r2 -d ./challenge1
Process with PID 1555 started...
= attach 1555 1555
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]> 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

```
[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
```

Q1 : Match the data type with the size in bytes:

Byte = 1

Word = 2

Double word = 4

Quad = 8

Single precision = 4

Double precision = 8

Q2: What is the command to analyse the program in radare2?

Ans : aa

Q3: What is the command to set a breakpoint in radare2 ?

Ans : db

Q4: What is the command to execute the program until we hit a breakpoint?

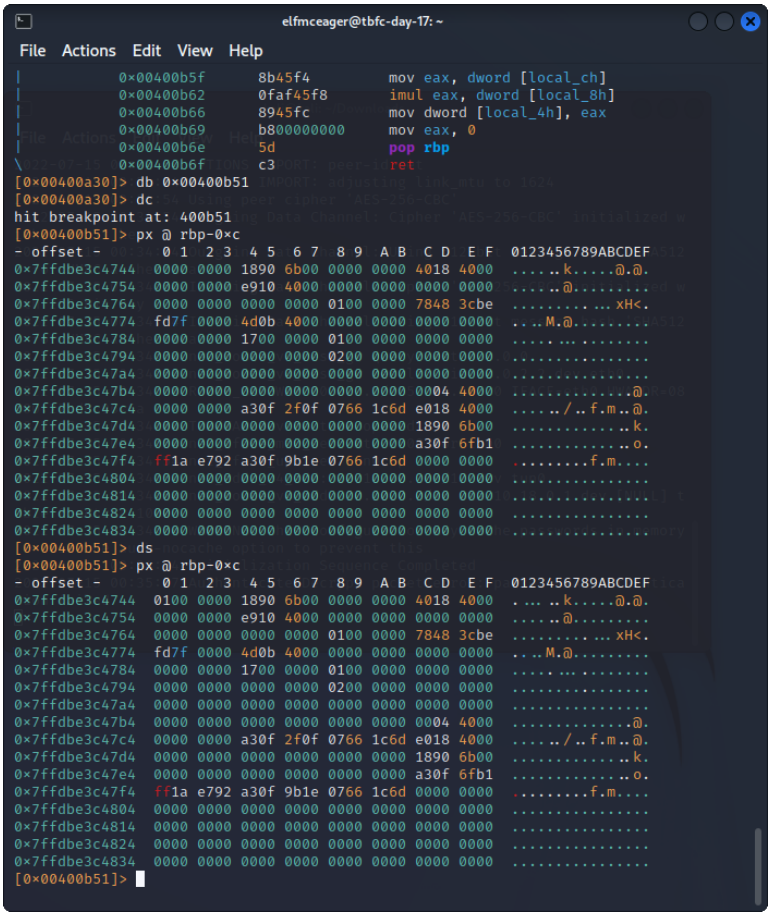
Ans : dc

Question 5

What is the value of local_ch when its corresponding movl instruction is called (first if multiple)?

Ans : 1

Find the address of local_ch which is 0x00400b51 and put a breakpoint at this location using command db. Now use dc to run the program. We can analyse the contents of local_ch with the command px @ rbp-0xc. Run command ds to step forward and command px @ rbp-0xc again. We see the value 1.



Question 6

What is the value of eax when the imull instruction is called?

Ans : 6

Next, we want to find the value of eax at the first imul instruction. We can simply use ds to move to the next line and then use dr to see the value of eax. Since we didn't see any number, Run command ds to step forward and dr to see the value of eax. We see that the value is 6.

Question 7

What is the value of local_4h before eax is set to 0?

Ans : 6

Lastly, we want to find the value of local_4h before the eax is set to 0. Use the ds command to get to that point and then check the value of local_4h with command px @ rbp-0x4. The value here is 6.

Thought Process/Methodology:

Log into the instance with the given credentials. Opens the binary files in debugging mode using command r2 -d ./file1 . Once the analyzing completes we can seek out the entry point to the file. Now we can see where in memory the program starts from. Since we found a function here we can examine it with the pdf (Print Disassembly Function) command. We can set a breakpoint with db. After setting our break point we can again run pdf @main and now we will see a lowercase b right after our break point location. Next, we use the dc command to run the program until it hits the break point. If we run pdf, the memory address at our breakpoint will be highlighted. Next, to view the contents of, in our case, the local_ch variable, we will run px @memory-address. But we do not see our variable 4 so we will step forward using ds. Then we will run px @rbp-0cx again.And we now can see the variable 4 returned. Going ahead to the challenge file, we follow the same initial steps. Find the address of local_ch which is 0x00400b51 and put a breakpoint at this location using command db. Now use dc to run the program. We can analyse the contents of local_ch with the command px @ rbp-0xc. Run command ds to step forward and command px @ rbp-0xc again. We see the value 1. Next, we want to find the value of eax at the first imul instruction. We can simply use ds to move to the next line and then use dr to see the value of eax. Since we didn't see any number, Run command ds to step forward and dr to see the value of eax. We see that the value is 6. Lastly, we want to find the value of local_4h before the eax is set to 0. Use the ds command to get to that point and then check the value of local_4h with command px @ rbp-0x4. The value here is 6.

Day 18: The Bits Of Christmas

Reverse Engineering the .Net app

Now we'll open the **TBFC APP** in **ILSpy**. This will allow us to conduct some reverse engineering and look at some of the app's source code.

When we open it, we are prompted to enter Santa's password. After some investigation, We discovered something intriguing in **CrackMe -> MainForm -> buttonActivate Click**.

```
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.Asterisk);
            return;
        }
    }
    MessageBox.Show("Uh Oh! That's the wrong key", "You're not Santa!", MessageBoxButtons.OK, MessageBoxIcon.Hand);
}
```

It appears that the programmer committed the error of hard-coding several important data! The password we're looking for is santapassword123.

```
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.Asterisk);
            return;
        }
    }
    MessageBox.Show("Uh Oh! That's the wrong key", "You're not Santa!", MessageBoxButtons.OK, MessageBoxIcon.Hand);
}
```

Question 1: santapassword123

We also see the flag that will be visible when we properly log in!

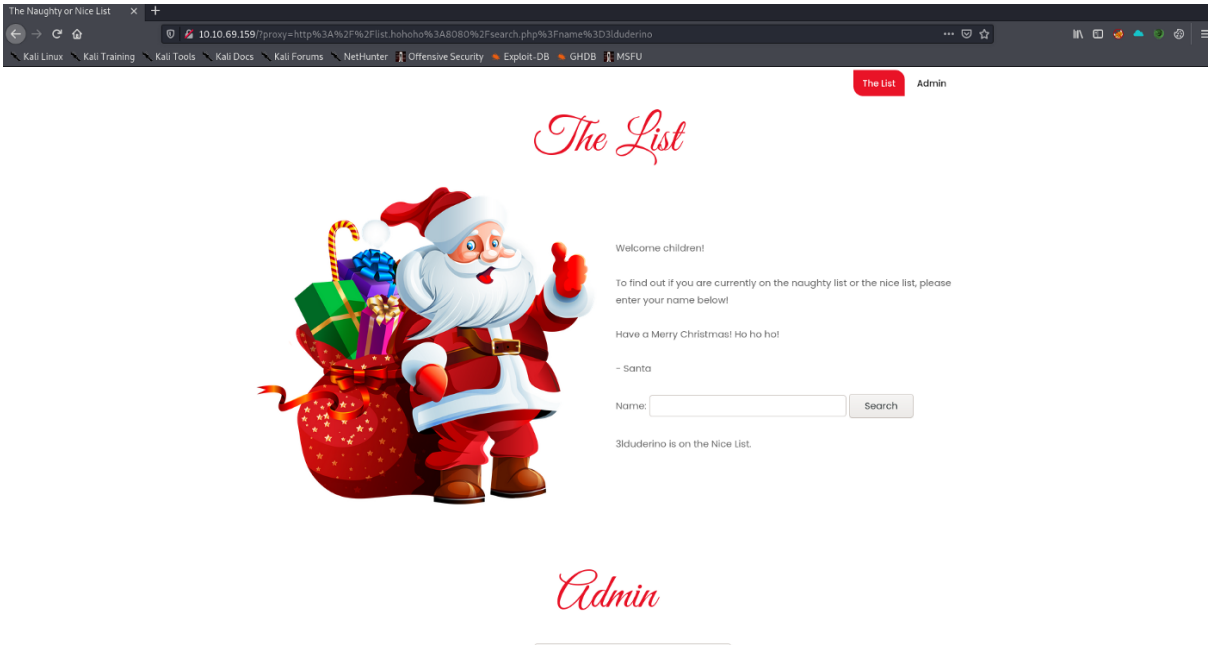
```
private unsafe void buttonActivate_Click(object sender, EventArgs e)
{
    IntPtr value = Marshal.StringToHGlobalAnsi(textBoxKey.Text);
    sbyte* ptr = (sbyte*)System.Runtime.CompilerServices.Unsafe.AsPointer(ref <Module>.??_C@_0BB@IKKDFEPG@santapassword321@);
    void* ptr2 = (void*)value;
    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.Asterisk);
            return;
        }
    }
    MessageBox.Show("Uh Oh! That's the wrong key", "You're not Santa!", MessageBoxButtons.OK, MessageBoxIcon.Hand);
}
```

Question 2: `thm{046af}`

Day 19: The Naught Or Nice List!

Question 1: What is Santa's password?

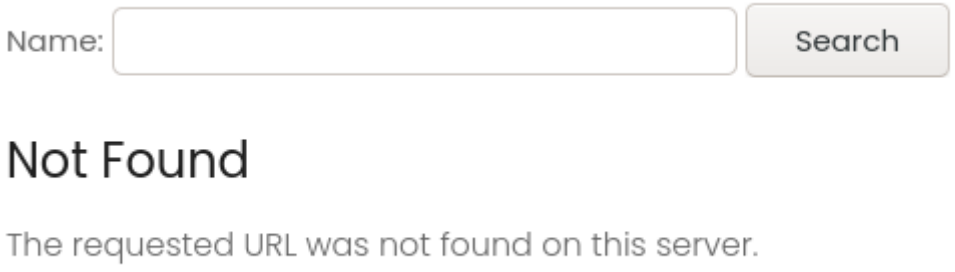
When we first visit the website, we see this form, along with an admin login underneath it.



The URL is intriguing: **http://10.10.69.159/**?proxy=**http%3A%2F%2Flist.hohoho%3A8080%2Fsearch.php%3Fname%3d3l Duderino** …. Also, we're not on the nice list, which is unfortunate. We conducted a short examination of the source code and found nothing of interest.

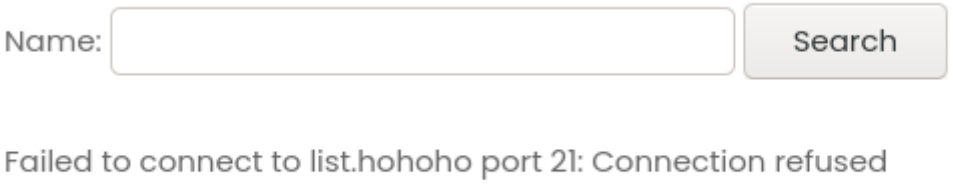
Burp did a short URL decoding and returned the following: **http://10.10.69.159/**?proxy=**http://list.hohoho:8080/search.php**?name=**3lduderino**

We search strip everything past the port number to determine if there is a root site and create a custom script to check for extra folders. In a failure site, we know what we're searching for (in this case, the root site).

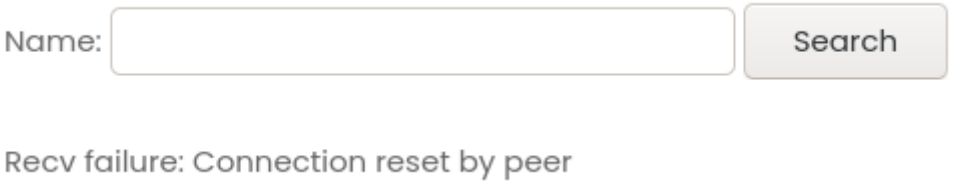


But for the time being, let's go on and see if there's another option.

So, it appears that we have a local server with port 8080 open; we may experiment with the port number to see what we can obtain. We can use this to perform basic port scanning.



However, it appears that we immediately receive a hit on port 22, which is open on the distant server.



We attempted to obtain the passwd file, but it appears that they are analysing the URL.

file://etc/passwd is on the Nice List.

When we try to determine whether there is a local only server operating, we are blocked (by their security team).

http://10.10.69.159/?proxy=http%3A%2F%2Flocalhost

Your search has been blocked by our security team.

Let's return to the port issue and see if we can find any local only services running on the server.

http://10.10.69.159/?proxy=http%3A%2F%2Flist.hohoho.localtest.me:22

With the connection reset by peer, we received the same problem as before. So far, so good. We may check to see if there are any ports that are exclusively accessible to the localhost. We now have a hit with port 80 containing some goodies, which addresses Question 1.

http://10.10.69.159/?proxy=http%3A%2F%2Flist.hohoho.localtest.me:80

Name: Search

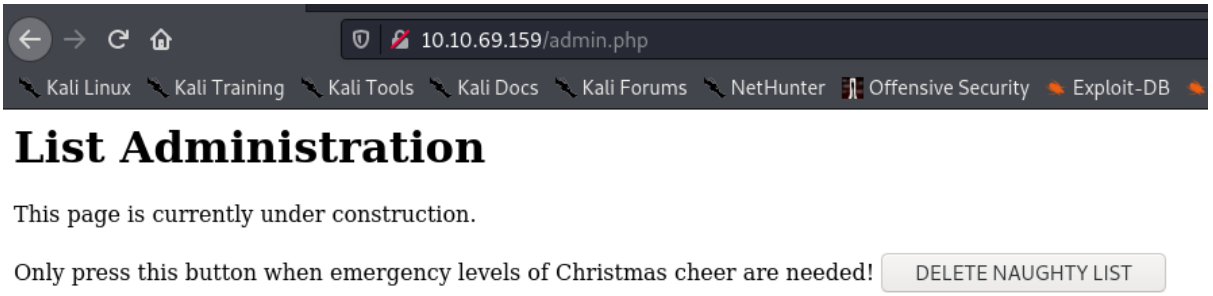
Santa,

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

Please keep in mind that the username is case sensitive.



We will obtain the flag for today's challenge after deleting the naughty list (or reading the source code).