Challenge 5.

We found a suspicious binary running on one of our servers. <u>Here</u> is the binary file for you to see. The interface below allows you to send an input to the binary. Can you see if you can make it output the vaccine component?

1. I downloaded the binary file to inspect and see if there was any obvious answer to what could be submitted. Inside the binary file the following stood out:

```
...libc.so.6._IO_stdin_used.printf.read.system.__libc_start_main.__gmon_start__.GLIBC_2.0.....
```

This looked like something being with a c library.

2. Putting some input into the message field returned some interesting results.

```
User provided 5 bytes. Buffer content is: sdfd ���
```

The extra characters I was getting at the end made me think there was a buffer overflow happening.

- **3.** I tried supplying 129 bytes and then only 1 to see what extra memory I could access. Only garbage results were returned. I also noticed that there was a max length of input around 400 bytes.
- **4.** I changed it to be executable via *chmod +x buff* to run some tests on my virtual machine.
- **5.** At this point I used boomerang to attempt to gain the source code from the binary. The resulting source code wasn't intelligible.

```
To the electronsessing with digits 4.1, bit commission and for the electronses which 4.5, 1.20 are found to the electronses with a first product of the electronses with a first product of the electronses with the electronses of the electronses with the electron
```

```
Unknown library function __libc_start_main
got filename crtstuff.c for main
got filename crtstuff.c for overflow
after removing matching assigns (t[WILDSTR]).
after removing matching assigns (%pc).
### WARNING: iteration limit exceeded for dfaTypeAnalysis of procedure overflow ###
in proc overflow adding addrExp r28{0} - 516 to local table
in proc overflow adding addrExp r28{0} - 4 to local table
in proc overflow adding addrExp r28{0} - 532 to local table
in proc overflow adding addrExp r28{0} - 536 to local table
in proc overflow adding addrExp r28{0} - 536 to local table
in proc overflow adding addrExp r28{0} - 540 to local table
```

6. Time to bring out the big guns and see if we can reverse engineer the file with Ghidra. A little internal inspection reveals something quite interesting.

```
void helper(void)
{
    system("cat flag.txt");
    return;
}
```

7. From close inspection it looks like this function isn't used. This made me wonder if I could overflow the buffer and pass an address to the function that would result in the flag being printed out. I noticed in a function that is being called an array of length 500.

```
undefined& sverflow(void)

undefined local_204 [500];

size_t local_10;

local_10 = read(0,local_204,700);

printf("\n8ser provided %d bytes, Buffer content is; %s\n",local_10,local_10,local_10,local_10.
```

Providing a 500 character string. Followed by an address (I tried 0804847d) didn't seem to do much. But I could see the address part wasn't being returned as a string because it overflowed the buffer. I tried writing the address backwards as is often the case for loading it in hex.

8. At this point a bash script seemed like a worthwhile option. I setup a fake flag.txt file for the buff program to cat and let the automation run.

```
parameter (A...) (A...) (O...)

parameter (A...) (A...) (O...)

ittem til at 

licano "licalica" ( ...)

acho "panica"

frequency tack of us making progress

it dams "Si" ( yran og "hh")

acho "Si"

fi

betain i "Ficherper(B))"

for mar is "Ficherper(B))"
```

9. Considering that would take all of eternity I also wrote up a c program to run the same functionality. That was considerably more efficient but nevertheless ineffective.

```
snprintf(cmd, sizeof(cmd), "echo \"1234567890

/* Open the command for reading. */
fp = popen(cmd, "r");
if (fp = NULL) {
    printf("Pailed to run command\n");
    exit(1);
}

/* Read the output a line at a time = output
while (fgets(path, sizeof(path), fp) != NULL)
    if ( (strstr(path, "flag") != NULL) ) (
        printf("FLRS+s", path);
}
```

10. Downloading cyclic and running against the binary allowed me to capture the memory address on which the program tried to jump to 0x66616165. From here I used cyclic to indicate the length of the buffer overflow needed which is 516 bytes.

```
The provided TSP bytes. Buffer content is: amantaear amminous managements and amminous managements amminous managements and amminous managements a
```

Using objdump we can also confirm the address of our function we would instead like to route the computer towards 0804847d (noting we need to write this backwards).

Unfortunately the server now returns nothing. Which means we have managed to get a segmentation fault, but not quite managed to call the function.