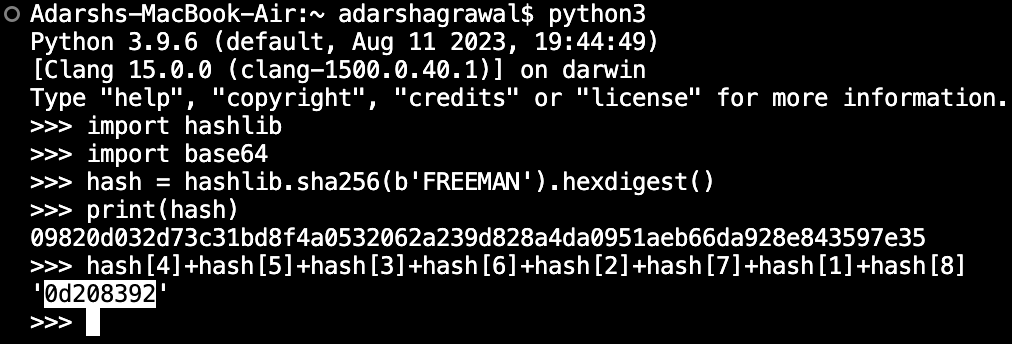
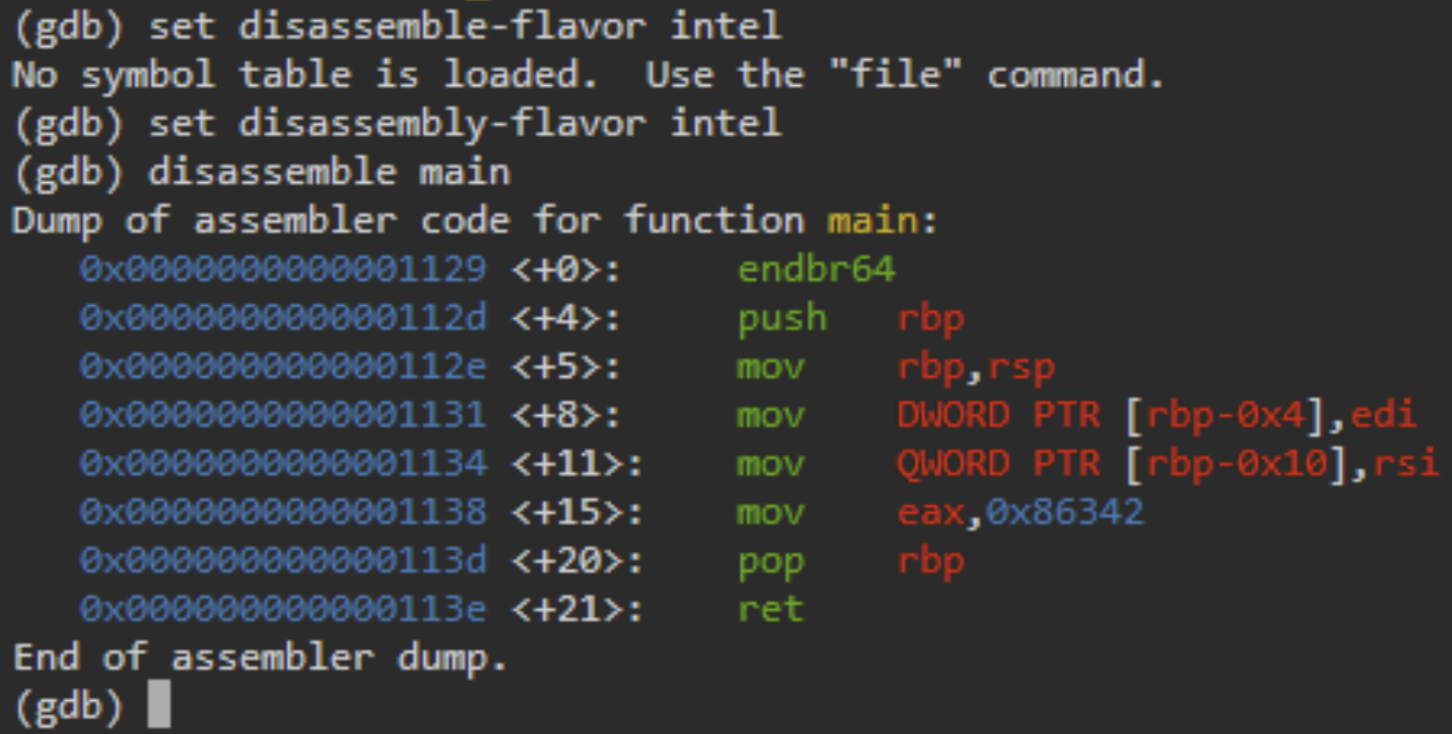
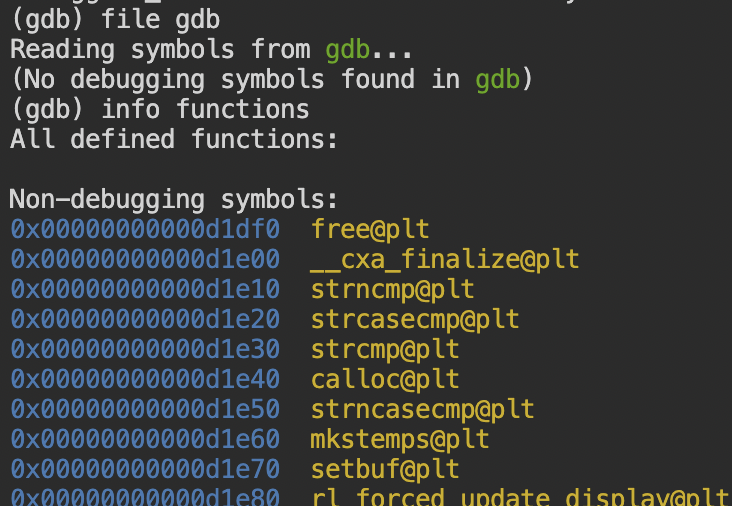
PICOCTF CHALLENGE WRITEUPS.

***DOMAIN : REVERSE ENGENEERING***

*CHALLENGE : KEYGENME-PY*

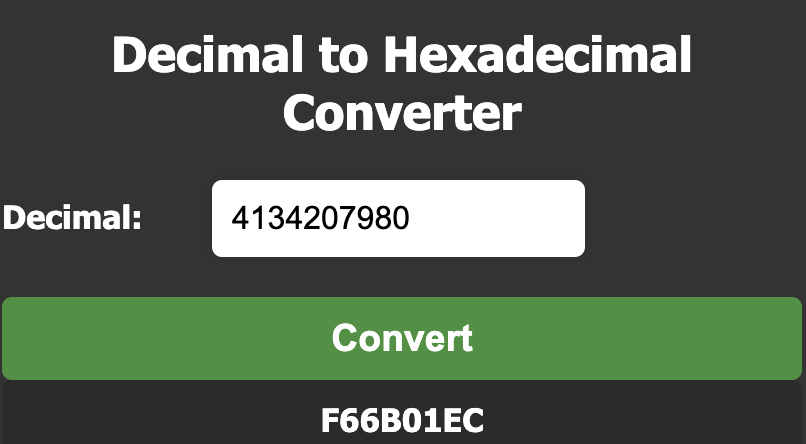
This challenge was fun and quite interesting. We analysed the source code as given and found that a part of the flag was already given as ‘picoCTF{1n\_7h3\_|<3y\_of\_’ and then ‘xxxxxxxx’ was supposed to be found and ‘}’ was to be added in the end and we would get our flag. First we will connect to the python shell in our terminal using “python3”. After we are connected to python shell, we need to connect to the hashlib library that was created in the source code. After that we also import the base64 in order to obtain data as it is in base64 format. After that we just create a variable “hash” and assign the value hashlib.sha256(b’FREEMAN’).hexdigest(). This is so because in the source code, we analysed that enter\_license function has the code to decrypt the hex using the check\_key function. Now the username is “FREEMAN” as defined in the source code in the beginning. So our syntax is justified. After that we just print the ‘hash’ value and get data in hex format. Then we use the specified positions give in the check\_key function and get our missing part of the flag.

So the flag for keygenme-py is picoCTF{1n\_7h3\_|<3y\_of\_0d208392}

*CHALLENGE : GDB BABY STEP 1*

In this challenge, first we needed to connect to the gdb directory in the webshell and then get information about all functions using the “info functions” command. After that we get a bunch of data and the syntax for gdb is set to AT & T by default. We have to use intel syntax for which we use “set disassembly-flavour intel” command. To disassemble the assembly code of the main function in this file, we will use the “disassemble main” command. We need to search for the contents of the "eax" register. The contents are in hexadecimal (“0x86342"). We can easily convert this hexadecimal to decimal using an online converter.

So the flag for gdb baby step 1 is picoCTF{549698}

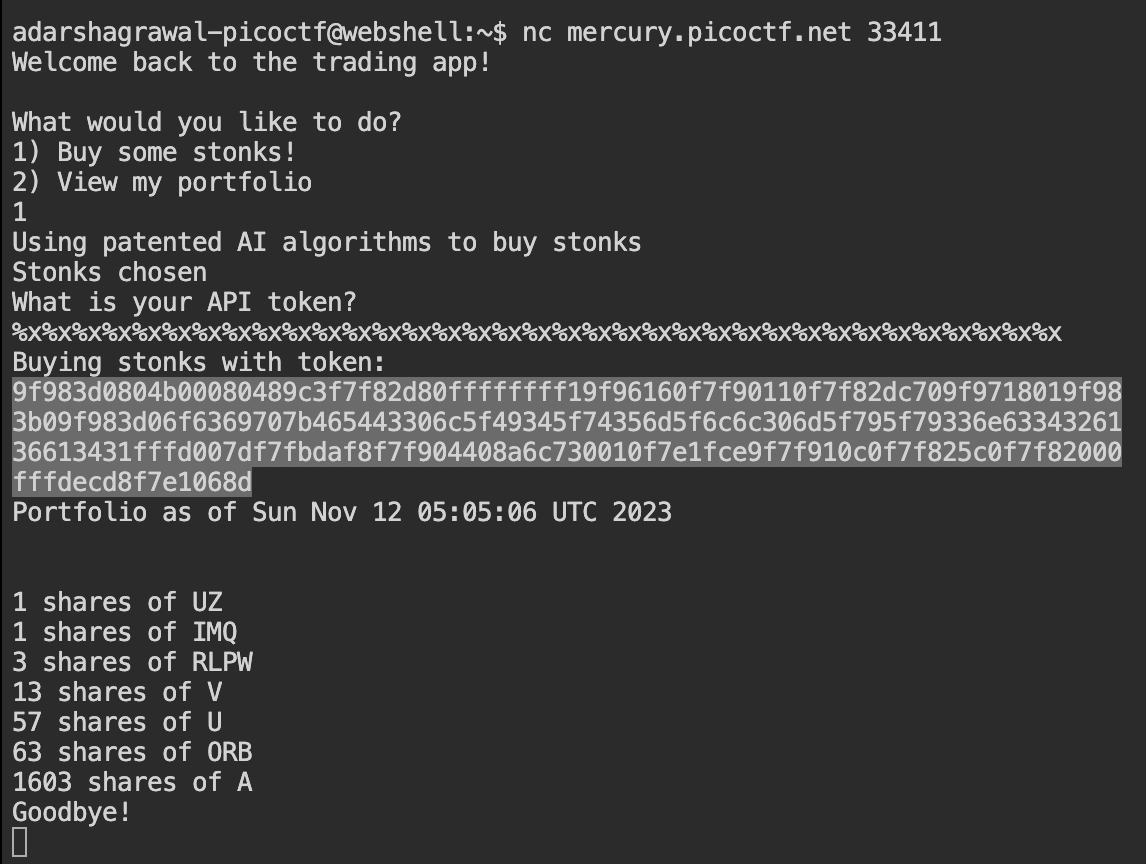
*CHALLENGE : ARMSSEMBLY 0*

Upon analysis, we found that we just have to convert the bigger number into the hexadecimal.

Also we need to type the content in lowercase as specified in the instructions.

So the flag for armssembly 0 is picoctf{f66b01ec}

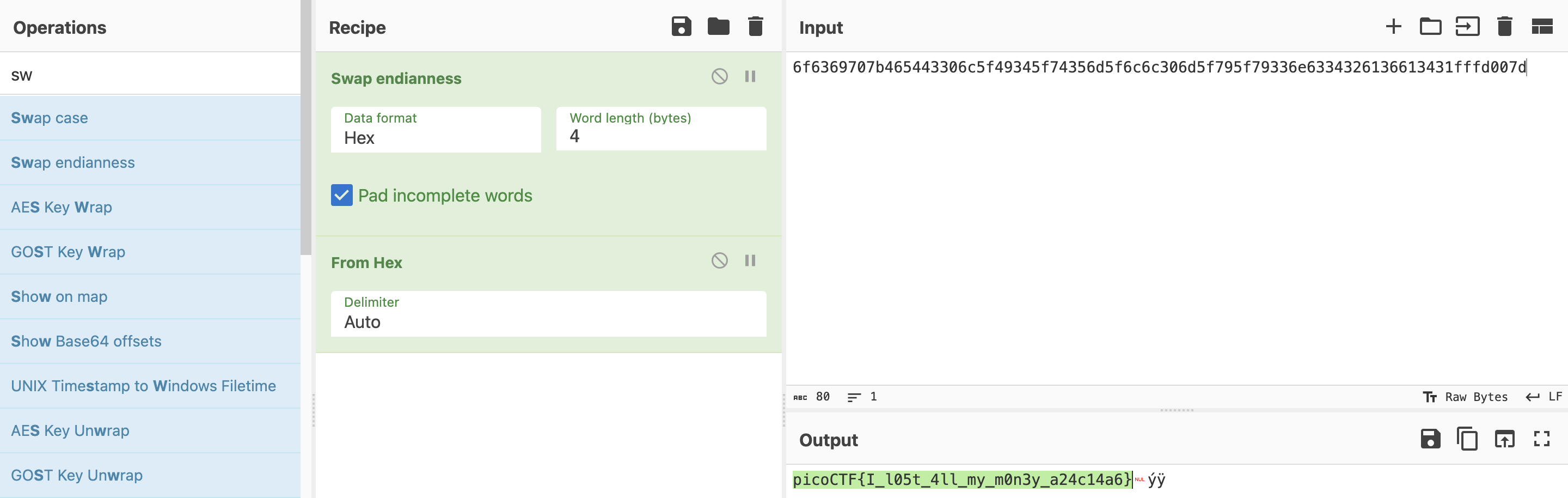
***DOMAIN : BINARY EXPLOITATION***

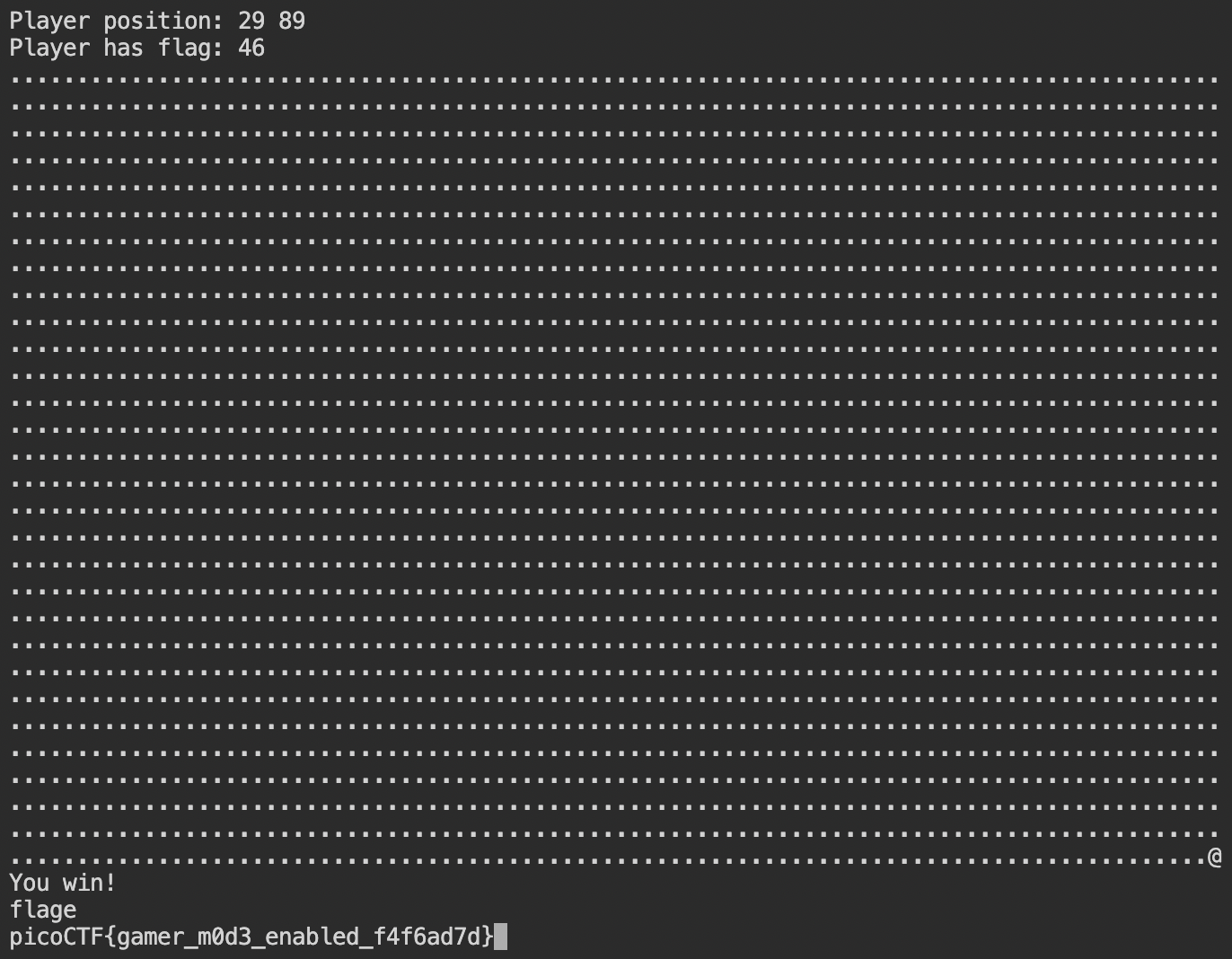
*CHALLENGE : STONKS*

In this challenge, first we had to connect to nc mercury.picoctf.net 33411 server which is given in the challenge description and then we had to type lots of ‘%x’ in order to get the data back from the stack in hex form which we will convert to ascii using a hex to binary converter for which I used cyber chef. After all this, we get back some data in hex form as highlighted in the ss. We will copy paste this on the site and convert it. We also see that it is jumbled, so we will add another operation that is swap endianness to bring it in the picoctf format as highlighted.

It has format string vulnerability. It means that we have to specify that data type using ‘%’, and when the program can’t find that data type, it starts printing data from stack.

As the only options were ‘1’ and ‘2’ and strings so it needed decimal or char data type inputs, but we specified it as ‘%x’ which is hex and hence it returned data in hex from the stack.

So the flag for stocks is picoCTF{I\_l05t\_4ll\_my\_m0n3y\_a24c14a6}

*CHALLENGE : BABY GAME01*

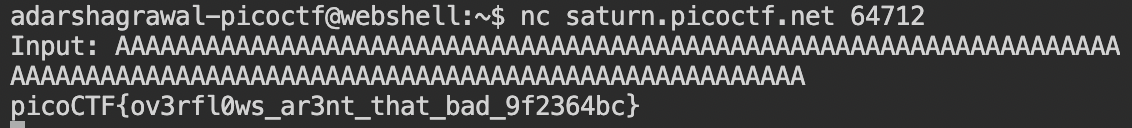
In this challenge, we have to move the ‘@‘ to the ‘x’ end position and after doing so it only displayed the “you win” message but that doesn’t solve our problem. We can move the ‘@‘ using ‘w’, ‘a’, ’s’, ‘d’ keys.

We looked upon the code and found that flag will be printed if the local variable ‘local\_a4’ is not a null value. To do so we will go to the {0,-4} position by typing 4 “a’s” and 5 “w’s” and then again 4 “a’s” to manurer it to {0’-4}. Now we will simply type ‘p’ to win the game and get the flag.

So the flag for baby game01 is picoCTF{gamer\_m0d3\_enabled\_f4f6ad7d}

*CHALLENGE : BUFFER OVERFLOW 0*

In this challenge, we connected to the server and as the name suggests we just had to overflow the input with binary data. So we just typed’A’, but we can type anything just to overflow it and then we will get our flag. It has stack overflow vulnerability.

So the flag for buffer overflow 0 is picoCTF{ov3rfl0ws\_ar3nt\_that\_bad\_9f2364bc}

***DOMAIN : WEB EXPLOITATION***

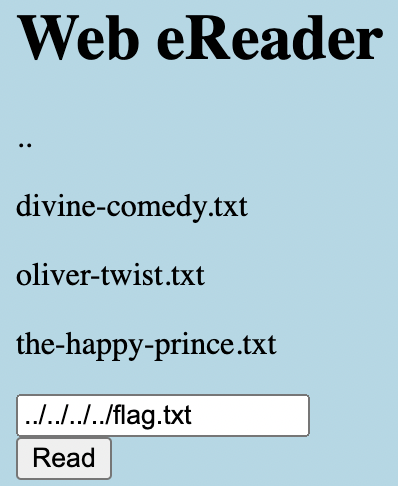
*CHALLENGE : CAAS*

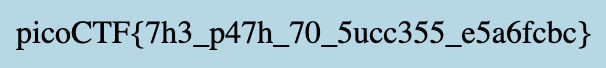
In this challenge we will open the URL given and use the link displayed there. We will use the “curl” command to access that link portal and then use “ls” to get the list of all directories. The vulnerability in this code lies in the fact that it directly uses user input (req.params.message) to construct a command that is executed with ‘exec'.

We will exploit this using the

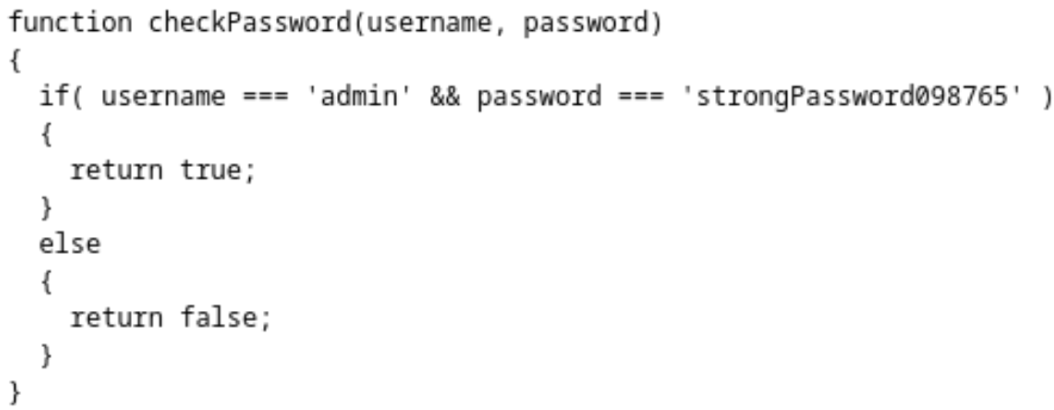
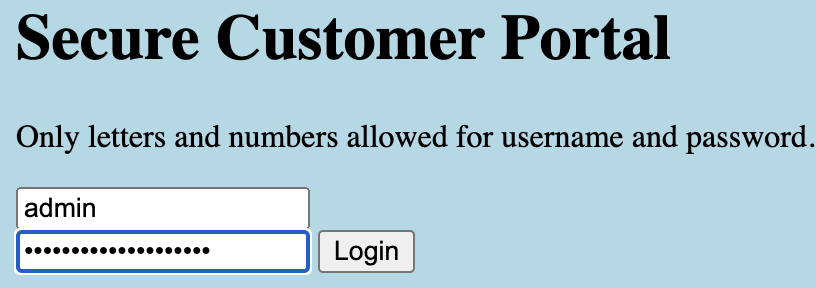
curl "<https://caas.mars.picoctf.net/cowsay>{message};cat%20falg.txt"; command.

So the flag for cars is picoCTF{moooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo0o}

*CHALLENGE : FORBIDDEN PATHS*

We know that the files are present in the "/usr/share/nginx/html/" directory, and the flag.txt file is present in the root directory. So, to reach the root directory, and display the flag.txt file, we can use the following input: “../../../../flag.txt”.

So the flag for forbidden paths is picoCTF{7h3\_p47h\_70\_5ucc355\_e5a6fcbc}

*CHALLENGE : LOCAL AUTHORITY*

*Screenshot 2023-11-16 at 3.48.04 PM.png*

In this one, after visiting the site, we will go to the page’s source code and found a locally included file, "secure.js", which tells us our required username and password.

So the flag for local authority is picoCTF{j5\_15\_7r4n5p4r3n7\_05df90c8}

***DOMAIN : FORENSICS***

*CHALLENGE : TUNN3L V1S10N*