**Web Exploitation**

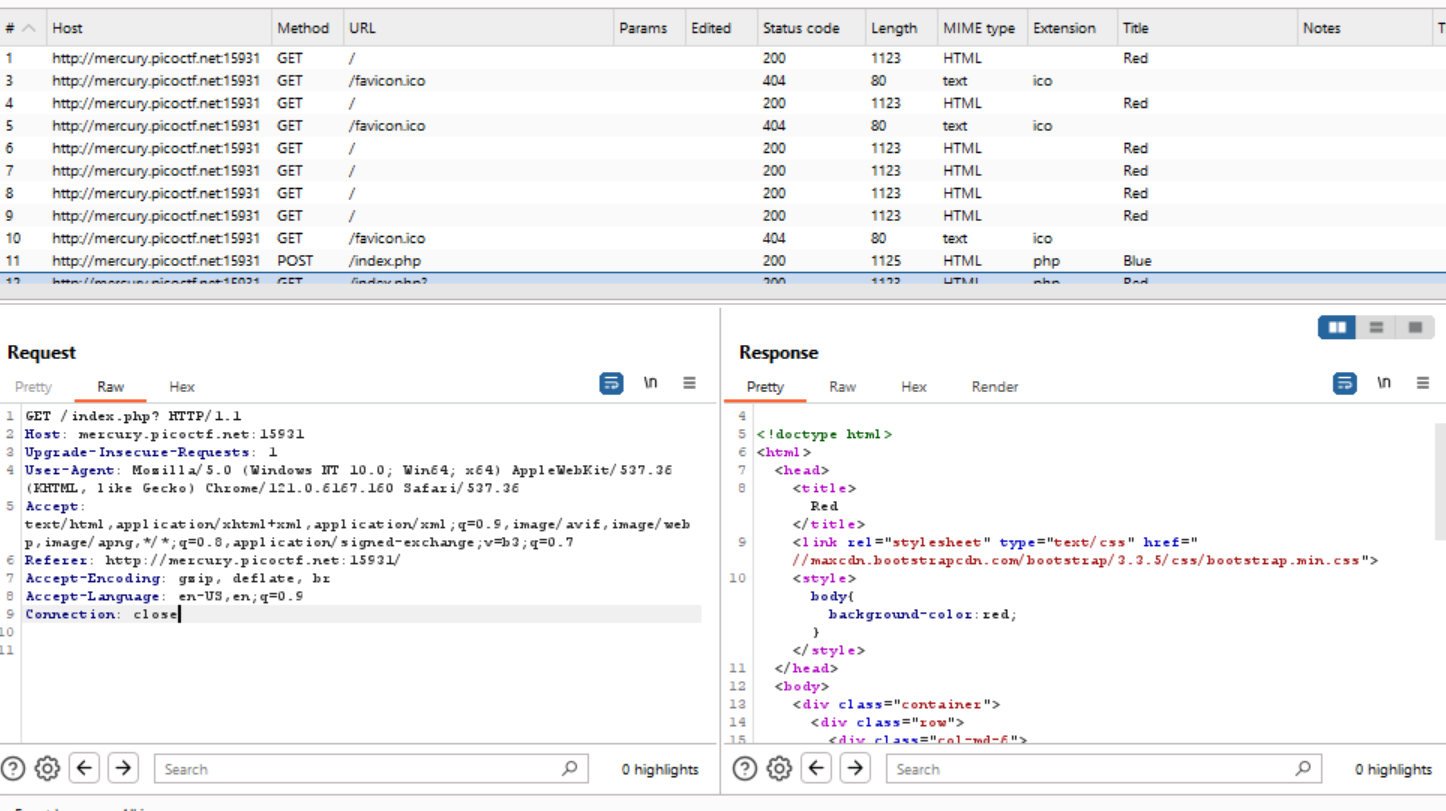
1. *Get Ahead*: Find the flag being held on this server to get ahead of the competition <http://mercury.picoctf.net:15931/>
2. *Cookies*: Who doesn't love cookies? Try to figure out the best one. <http://mercury.picoctf.net:21485/>
3. *Insp3ct0r*: Kishor Balan tipped us off that the following code may need inspection: https://jupiter.challenges.picoctf.org/problem/41511/ ([link](https://jupiter.challenges.picoctf.org/problem/41511/)) or http://jupiter.challenges.picoctf.org:41511

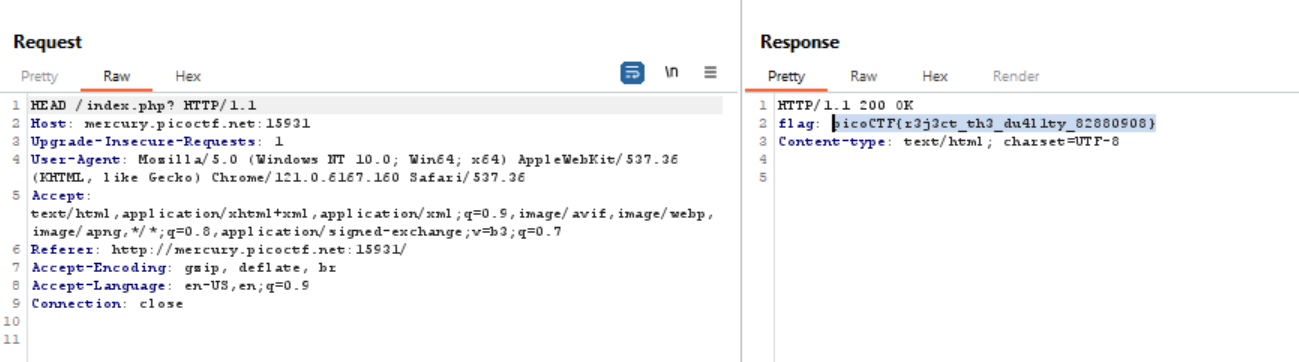
**Solution**

1. GET aHEAD

The challenge presents a link that directs us to a webpage where we can interact with two buttons. Despite exploring the website thoroughly, examining its source code, and inspecting cookies, I still hadn’t identified any potential injection points or exploits to exploit.

At this point I wasn’t sure what to do, so I looked at both the hints, downloaded Burpsite and began intercepting the website using proxy.





I turned off the intercept and went to the http history, recalling that ‘Head’ is one of the http request methods. From there, I sent the GET data to the repeater and changed the request from ‘GET’ to ‘HEAD’. Sending the data, we can see that I have obtained the flag.

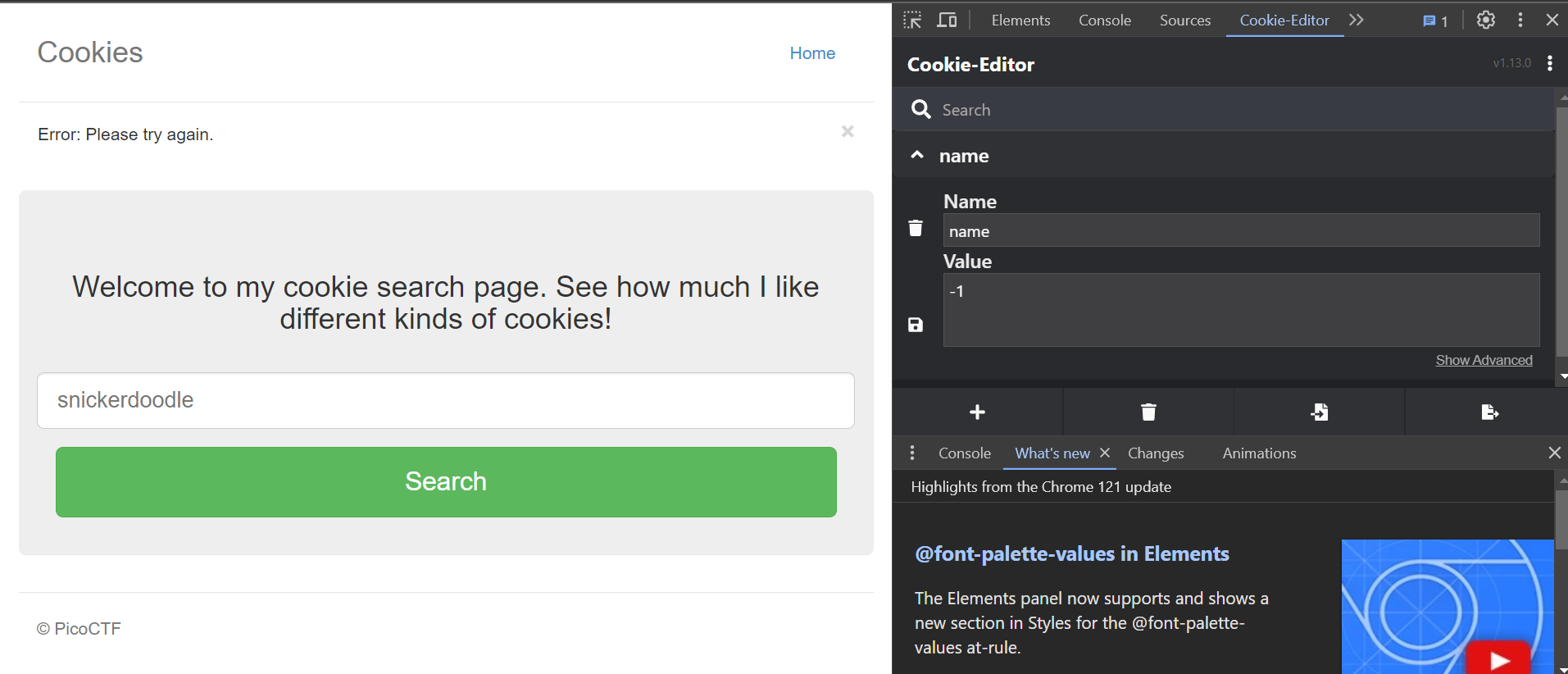
Flag: picoCTF{r3j3ct\_th3\_du4l1ty\_82880908}

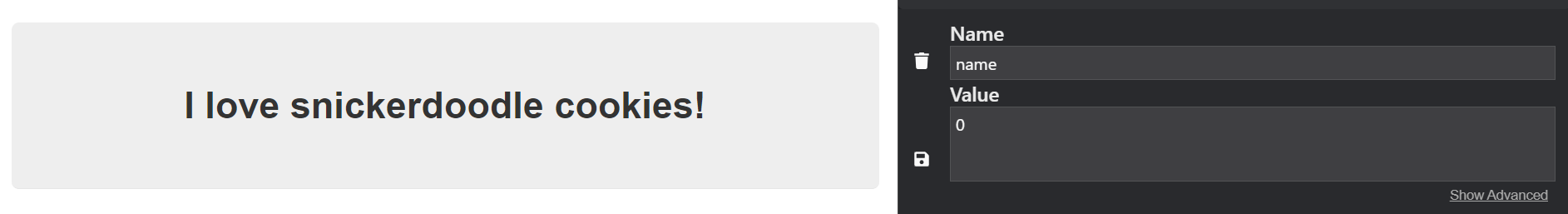
References: <https://www.youtube.com/watch?v=GrUuWYwA5l0>

1. Cookies

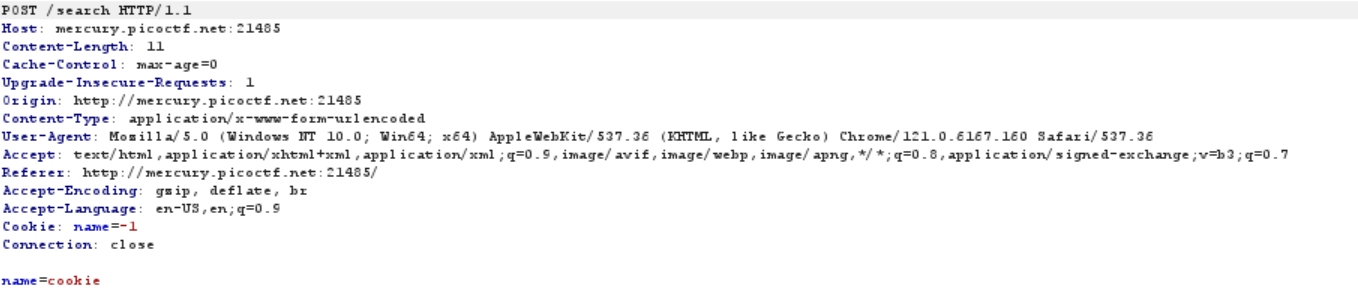
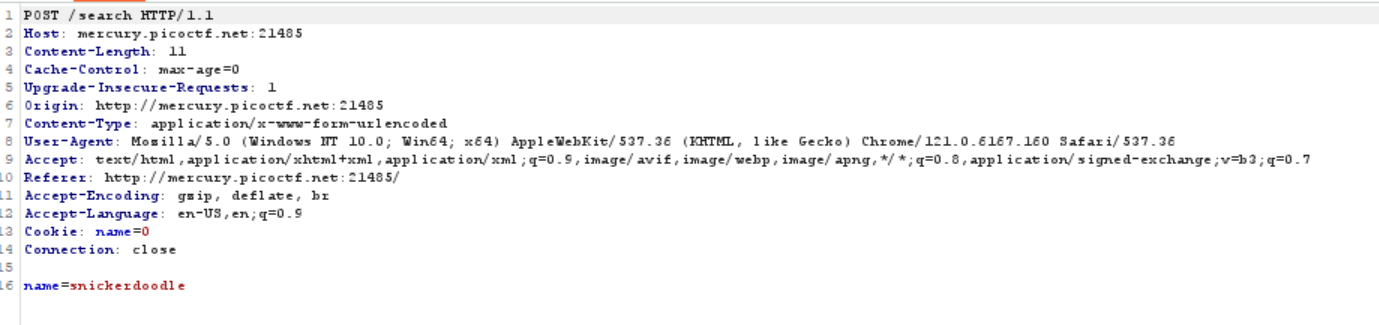
This question was rather challenging since I was still quite new to CTF concepts. I had to look up different ways to use cookie editors, even trying out a chrome extension.

Unfortunately, although I understood that each cookie had a different value with snickerdoodle being 0 and chocolate chip being 1 and so on, I was never actually able to go the other way around, i.e. figure out the name of the cookie from the value assigned to it.

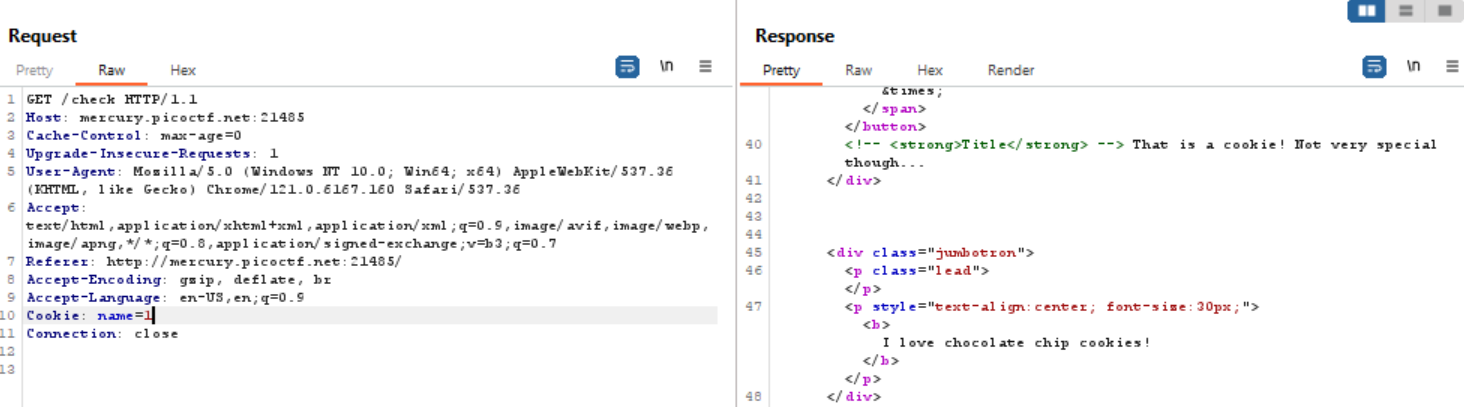


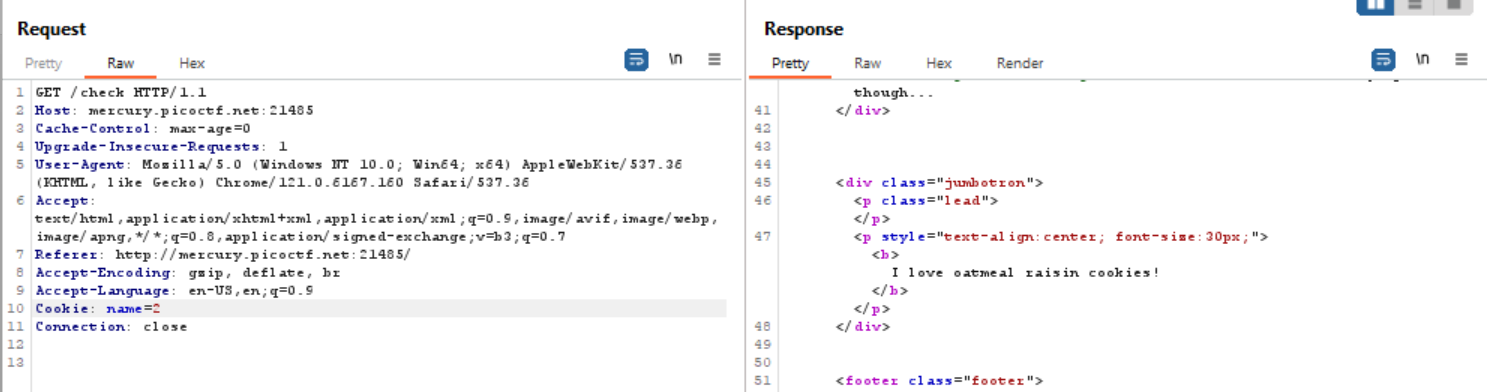


Eventually I settled on using burp suite again as I had used it in the first problem. I sent the POST data to the repeater and change name value to 1 then 2. The responses generated clearly have something to do with the cookie value.

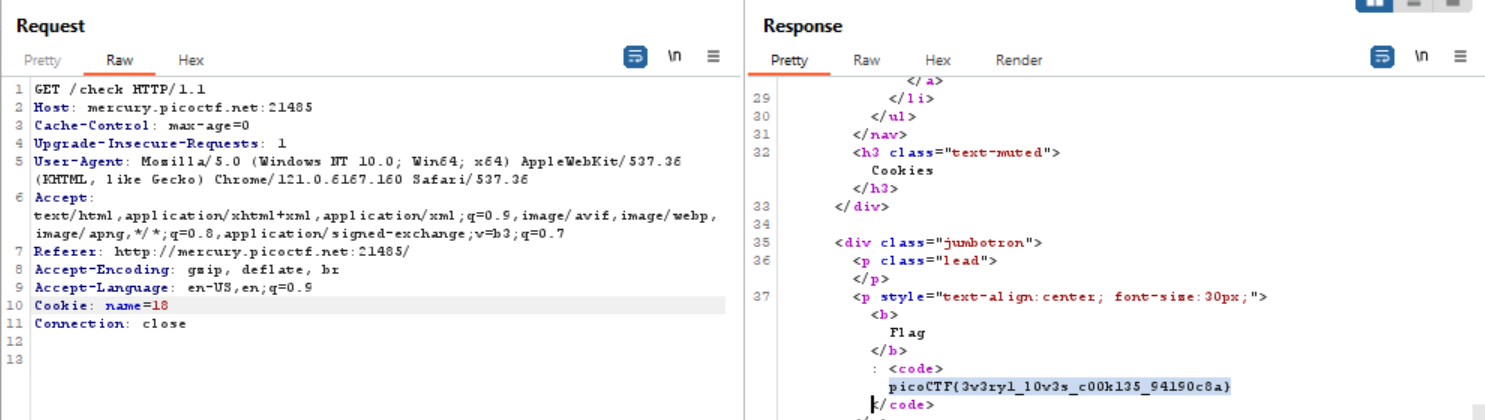


Sent this request to the Intruder and change the cookie name parameter, ascending up slowly and hoping to stumble upon the right flag.





I observe a different response length for the Cookie: name=18. In this response, I am also able to obtain the flag required.

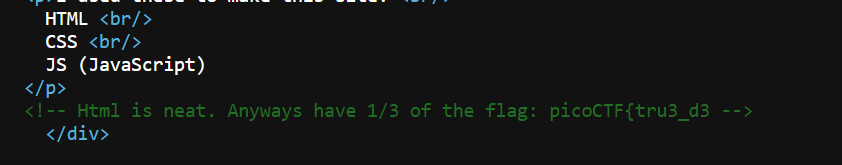


Flag: picoCTF{3v3ry1\_l0v3s\_c00k135\_94190c8a}

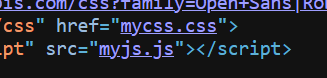
References: <https://portswigger.net/support/using-burp-to-hack-cookies-and-manipulate-sessions>

1. Insp3ct0r

Open the website and begin to inspect it. Then I move onto viewing the page source to see if that will lead me anywhere. It immediately does, as I can see a comment in green, giving me a third of the required flag.



I realize that, maybe the way that HTML, CSS and JS are written on the website have some significance, since a third of the flag came from association with HTML. With a bit of scrolling, I am able to locate a .css and .js file at the head of the page source code.



A bit more scrolling on each of those websites gives me the second and third parts of the flag.





Flag: picoCTF{tru3\_d3t3ct1ve\_0r\_ju5t\_lucky?832b0699}

References: none

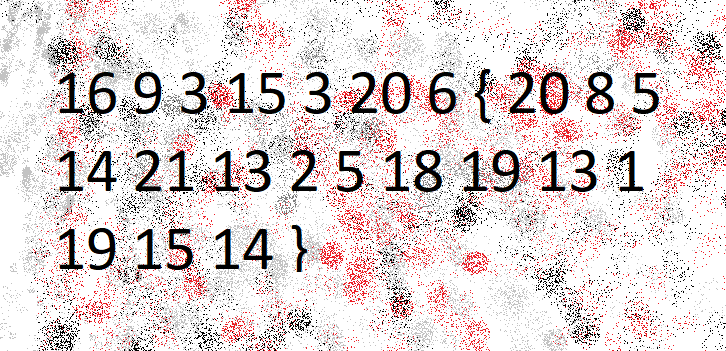
**Cryptography**

1. *The Numbers:* The [numbers](https://jupiter.challenges.picoctf.org/static/f209a32253affb6f547a585649ba4fda/the_numbers.png)... what do they mean?
2. *13*: Cryptography can be easy, do you know what ROT13 is? cvpbPGS{abg\_gbb\_onq\_bs\_n\_ceboyrz}
3. *caeser:* Decrypt this [message](https://jupiter.challenges.picoctf.org/static/6385b895dcb30c74dbd1f0ea271e3563/ciphertext).

**Solution**

1. The Numbers

This one was very simple, I just did the first thing that comes to mind when I saw the given png, which is to convert the numbers into their respective alphabetical counterpart.



Which means, 1 is A, 2 is B and so on. Doing so, we get the flag: PICOCTF{THENUMBERSMASON}

Flag: PICOCTF{THENUMBERSMASON}

Links used: None

1. 13

The question gives us the answer. All I did was run the given alphabets through a Rot13 translator, that pushes forward each letter by 13 letters in the alphabetical value.

Eventually, we get the flag really easily.

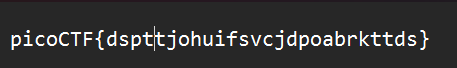


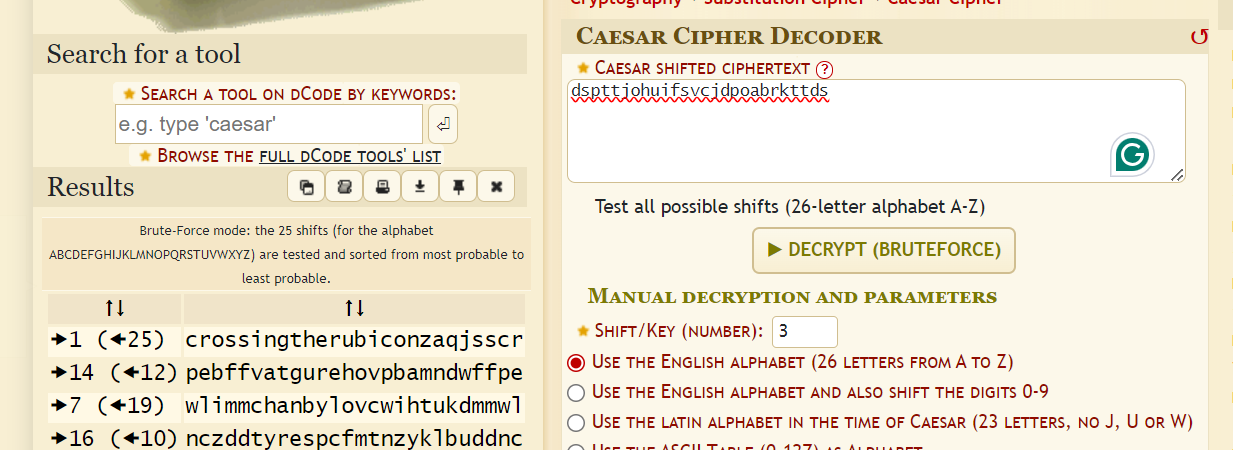
Flag: picoCTF{not\_too\_bad\_of\_a\_problem}

Links used: <https://www.dcode.fr/rot-13-cipher>

1. caeser

Again, the question gives us a clue as to what type of cipher it may be. So, I pulled up a caeser cipher decoder and input the flag that was given to me, that I opened in a notepad.



We get a few results but only the very first one forms some comprehensible flag. 

Flag: picoCTF{crossingtherubiconzaqjsscr}

Links used: <https://www.dcode.fr/caesar-cipher>

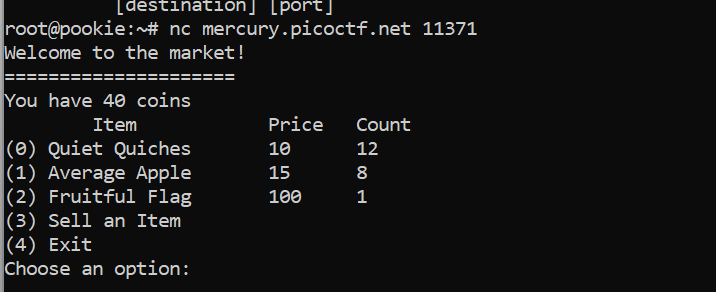
**Reverse Engineering**

1. *Shop*: Best Stuff - Cheap Stuff, Buy Buy Buy... Store Instance: [source](https://mercury.picoctf.net/static/bd84b0d8b57e043a028c36381910d0b7/source). The shop is open for business at nc mercury.picoctf.net 11371.
2. *vault-door-training:* Your mission is to enter Dr. Evil's laboratory and retrieve the blueprints for his Doomsday Project. The laboratory is protected by a series of locked vault doors. Each door is controlled by a computer and requires a password to open. Unfortunately, our undercover agents have not been able to obtain the secret passwords for the vault doors, but one of our junior agents obtained the source code for each vault's computer! You will need to read the source code for each level to figure out what the password is for that vault door. As a warmup, we have created a replica vault in our training facility. The source code for the training vault is here: [VaultDoorTraining.java](https://jupiter.challenges.picoctf.org/static/1afdf83322ee9c0040f8e3a3c047e18b/VaultDoorTraining.java)
3. *vault-door-1*: This vault uses some complicated arrays! I hope you can make sense of it, special agent. The source code for this vault is here: [VaultDoor1.java](https://jupiter.challenges.picoctf.org/static/29b91e638ccbd76aaa8c0462d1c64d8d/VaultDoor1.java)

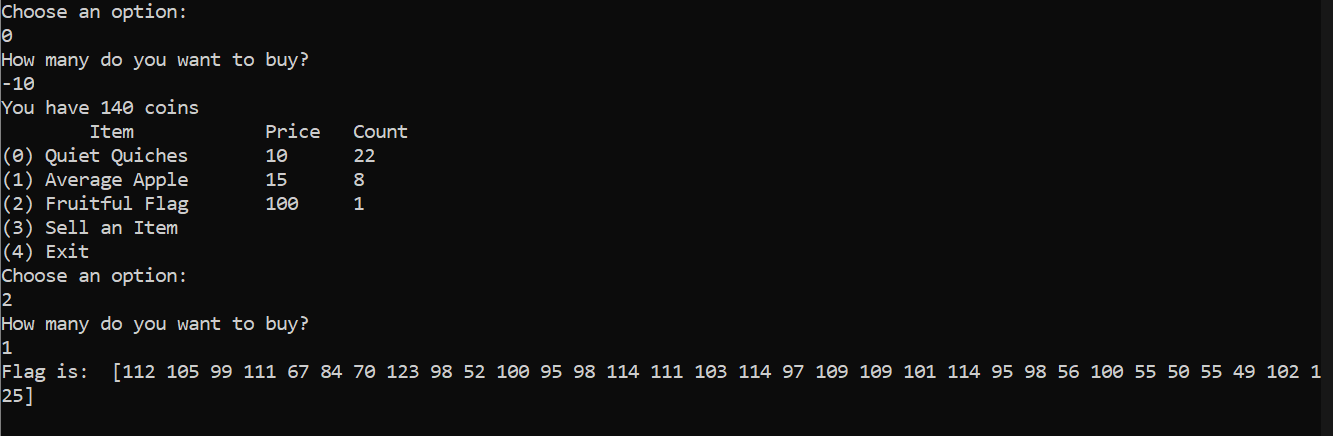
**Solutions**

1. Shop

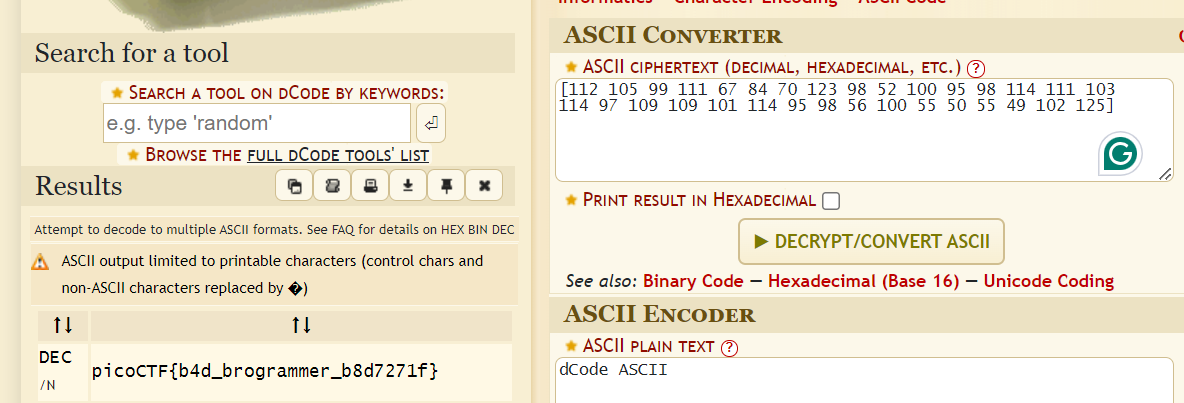
Saw the command and opened Ubuntu, and input the command first. Then clicked on the hint to see if I should go in a particular direction.



The hint says to check edge cases so that is what I did, inputting negative numbers, since that is the most out of ordinary thing possible. Now I have enough coins to buy the flag.



Got an ASCII value that I ran through an ascii decoder and got the required flag.

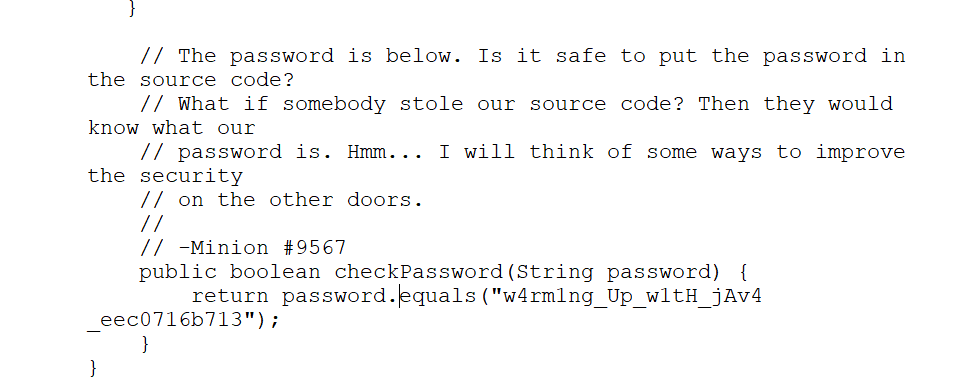


Flag: picoCTF{b4d\_brogrammer\_b8d7271f}

References: <https://testsigma.com/blog/edge-case-testing/>

1. vault-door-training

All I had to do was open the java file in wordpad since I wasn’t able to open it in BlueJ.

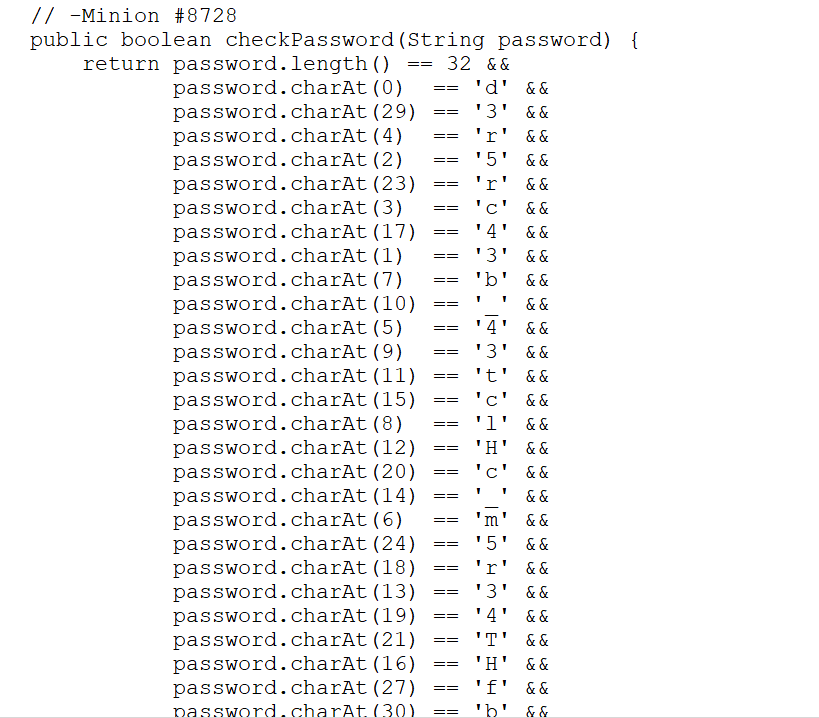


Flag: picoCTF{w4rm1ng\_Up\_w1tH\_jAv4\_eec0716b713}

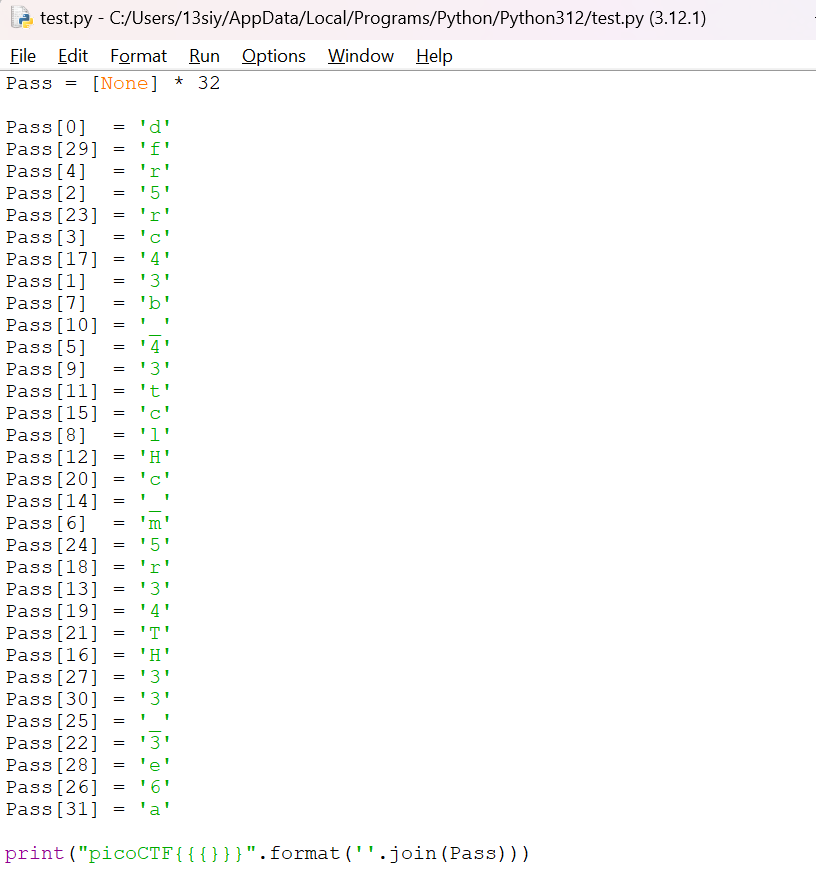
Links used: None

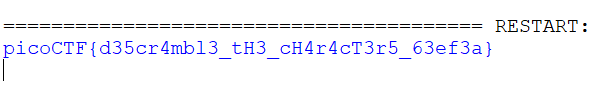
1. vault-door-1

Again, looked at the code in a wordpad and wasn’t able to figure it out for a minute and thought I would have to execute it. Then realized that the flag was stored in a vertical column and I could just type out the individual letters.



Did not work. Then looked at the first hint that said to use charAt(), which I looked up and decided to write the following code snippet to decrypt the password.





Flag: picoCTF{d35cr4mbl3\_tH3\_cH4r4cT3r5\_63ef3a}

Links used: <https://www.w3schools.com/java/ref_string_charat.asp>

**Reverse Engineering**

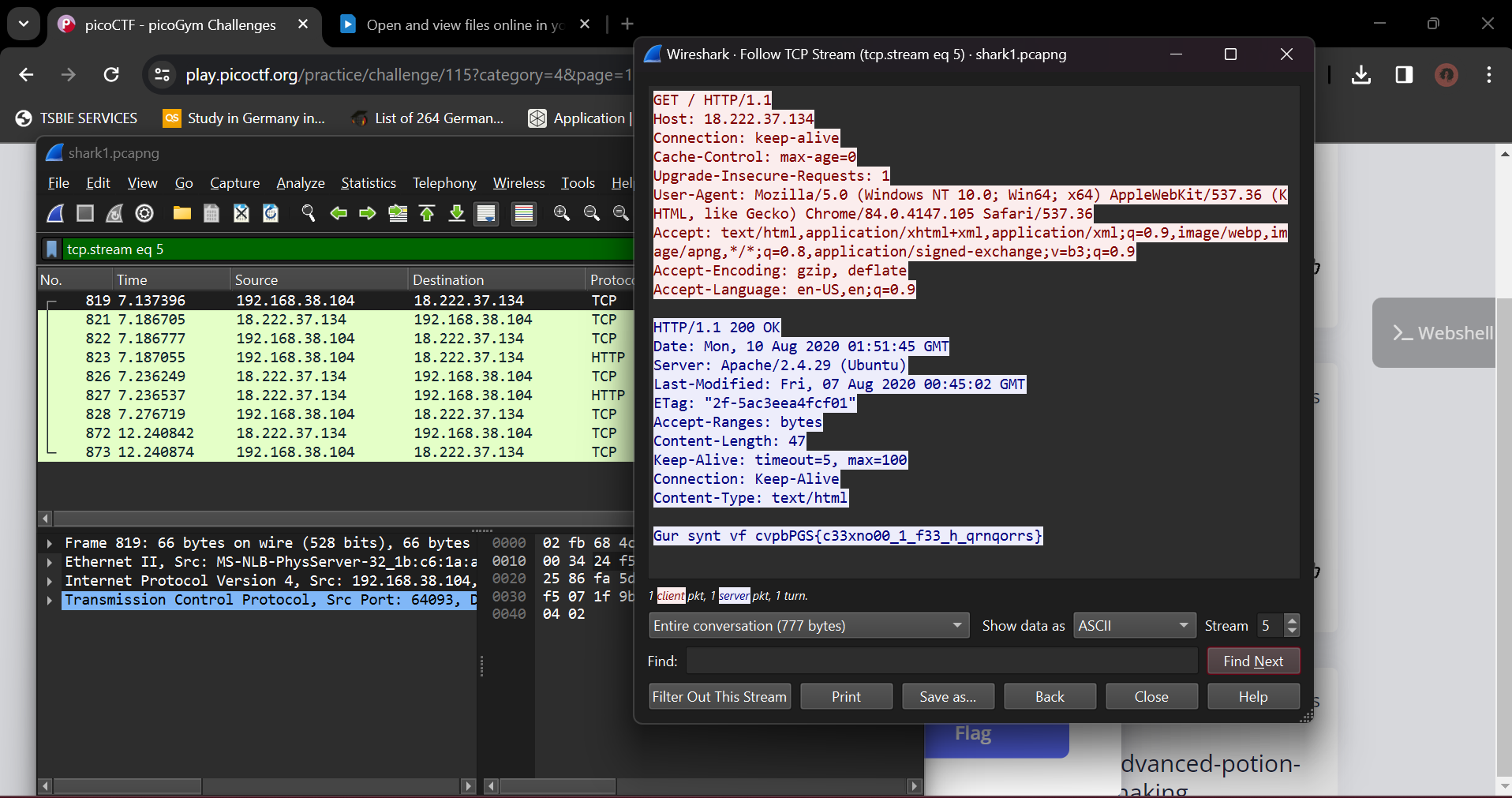
1. *wireshark doo do do*: Can you find the flag? [shark1.pcapng](https://mercury.picoctf.net/static/0505a462ac9beb7412596855df280f6b/shark1.pcapng).
2. *hideme:* Every file gets a flag.The SOC analyst saw one image been sent back and forth between two people. They decided to investigate and found out that there was more than what meets the eye [here](https://artifacts.picoctf.net/c/261/flag.png).
3. *Redaction gone wrong*: Now you DON’T see me.This [report](https://artifacts.picoctf.net/c/84/Financial_Report_for_ABC_Labs.pdf) has some critical data in it, some of which have been redacted correctly, while some were not. Can you find an important key that was not redacted properly?

**Solution**

1. wireshark doo do do

I already had a bit of knowledge about wireshark from a workshop I had taken and hence, the name of this challenge caught my eye.

I did not, however, know how to filter out the required data, even if I knew we needed TCP stream. So I looked it up and the guides said to intercept the 5th stream, as that is where the majority of the data was flowing through.



Intercepting the signal of the file and following the TCP stream, led me to a string that seemed like an encoded flag. Unfortunately, I did not know how it was encrypted so I used a cipher identifier to recognize the cipher.



From there, I decoded the rot13 in the decrypter to find the required flag.

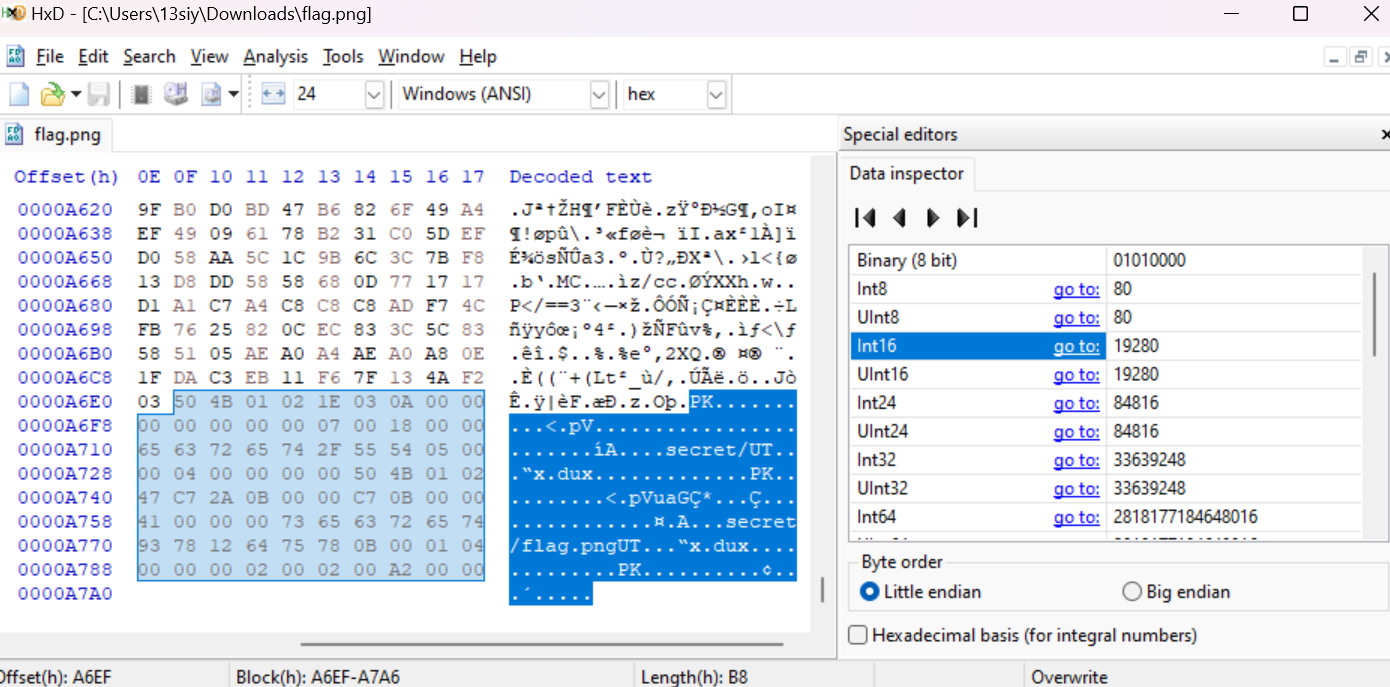
Flag: picoCTF{p33kab00\_1\_s33\_u\_deadbeef}

Links used: <https://www.dcode.fr/cipher-identifier>

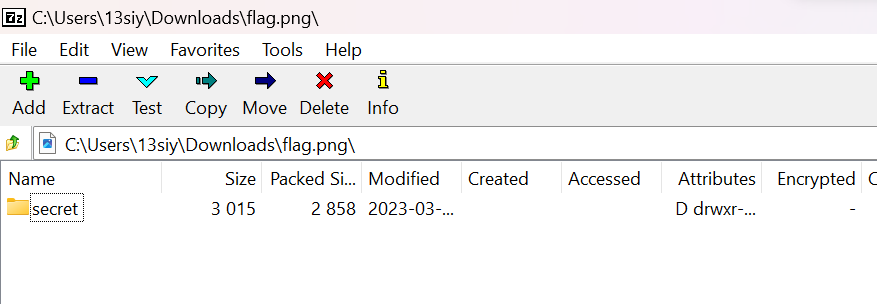
1. hideme

This task was by far the most challenging among the ones I had tried, especially since downloading the HxD program was so risky and Windows would not let me run the program, which was frustrating to get past.

Eventually I was able to open up the program and input the text that was clearly in hex, although I didn’t have to guess because the program automatically configured itself. At the very end of the decoded text, there was some comprehensible language that referred to secret/flag.png.



So, I figured there was a folder I wasn’t able to see, and ran the file through 7zip file manager. On opening the file again, an image file opens with the flag clearly written on it which I transcribed into the writeup.



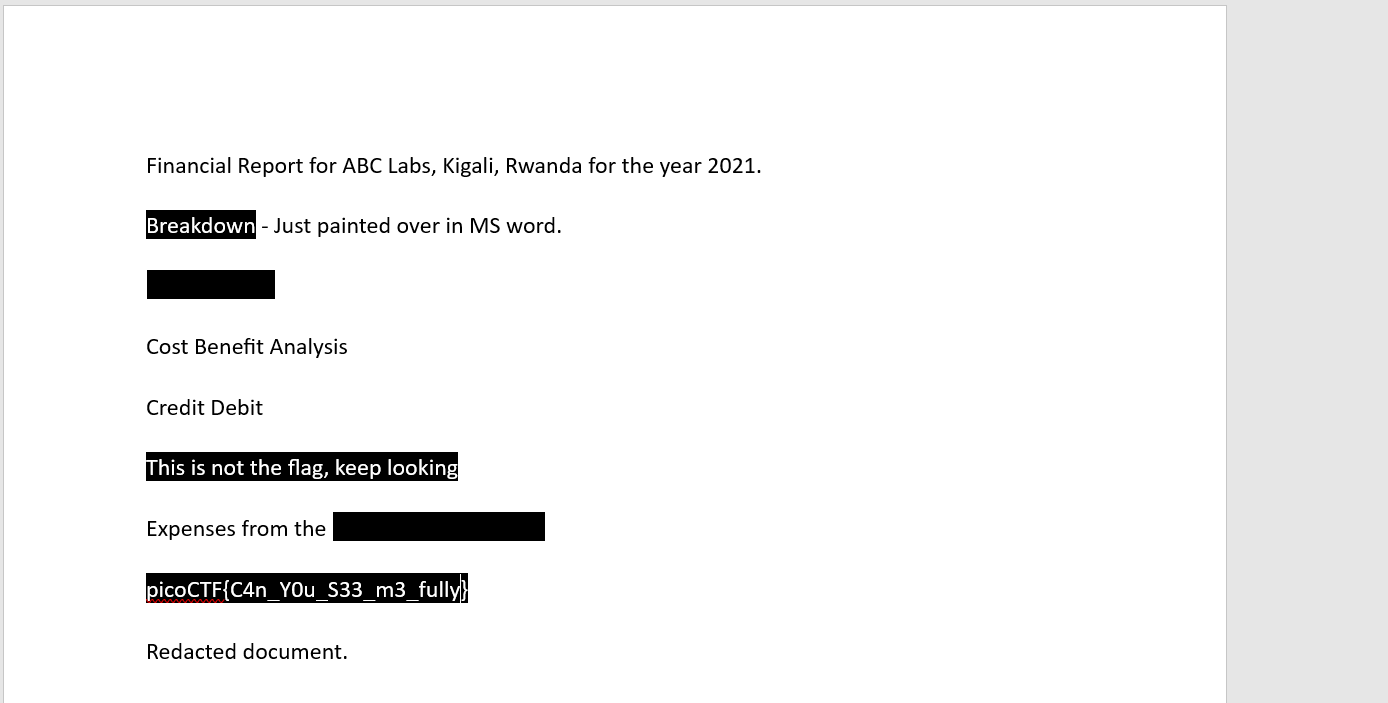


Flag: picoCTF{Hiddinng\_An\_imag3\_within\_@n\_ima9e\_96539bea}

Links used: <https://www.youtube.com/watch?v=BcWGMVOGdOI>

1. Redaction gone wrong

This question was much simpler than its predecessor, and it was immediately clear to me that I could select certain text under some of the redacted boxes. So, I reopened the file as a word document and changed the colour of the text to white so I could see it easier.



Alternatively, I could have copy pasted each individual phrase from the pdf directly. (I tried and it worked but it’s not possible to get a screenshot of that process so I did it this way instead.)

Flag: picoCTF{C4n\_Y0u\_S33\_m3\_fully}

Links used: None

**General Skills**

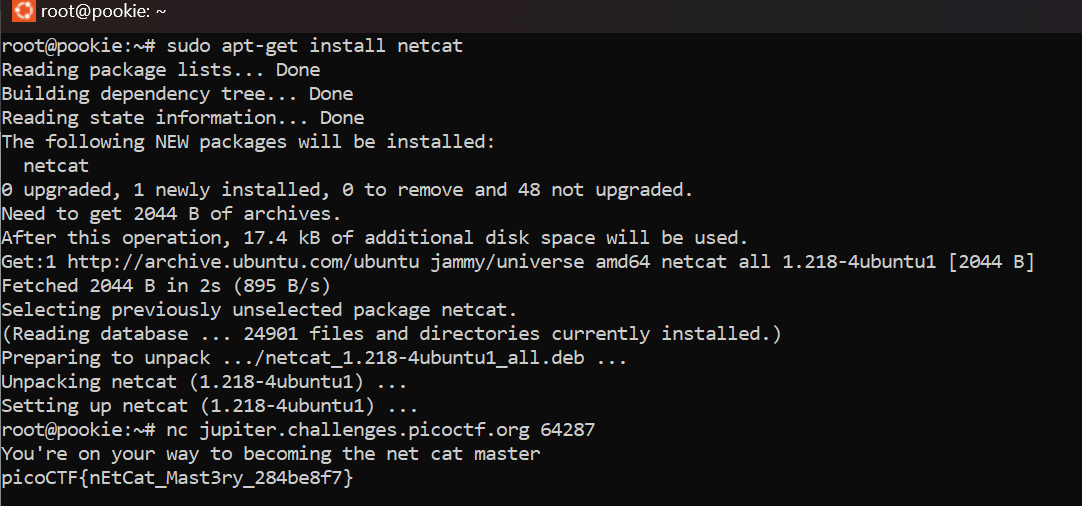
1. *What’s a net cat:* Using netcat (nc) is going to be pretty important. Can you connect to jupiter.challenges.picoctf.org at port 64287 to get the flag?
2. *Bases:* What does this bDNhcm5fdGgzX3IwcDM1 mean? I think it has something to do with bases.
3. *Glitch Cat*: Our flag printing service has started glitching! $ nc saturn.picoctf.net 49700

**Solution**

1. what’s a net cat

I tried to run net cat without having install anything, on both ubuntu and windows command prompt, but it said it didn’t recognize the nc command which was confusing, since I used it all the time while doing the bandit levels.

Eventually just gave up and installed net cat in ubuntu (and not Windows, because the Windows process is longer). Inputting just ‘nc’ will tell us the syntax for using nc, which I already knew, so I just input the command directly as instructed in the question to find the flag.



Flag: picoCTF{nEtCat\_Mast3ry\_284be8f7}

Links used: <https://www.unixmen.com/play-with-netcat-in-ubuntu/>

1. Bases

I put the given text into a base64 decoder and got the output, which was the main part of the flag. Added picoCTF{} to it and finished up the level.



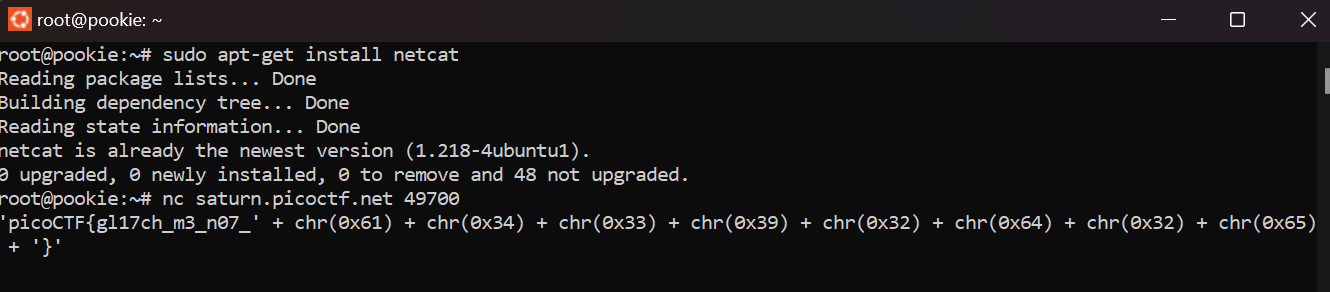
Flag: picoCTF{l3arn\_th3\_r0p35}

Links used: <https://www.dcode.fr/base-64-encoding>

1. Glitch Cat

Entered the given network and got a string of characters with a half decrypted flag.

Using an ASCII table from the link below, we can convert all the given hexadecimal code into ASCII values and get the missing part of the flag.



0x61 = a

0x34 = 4

0x33 = 3

0x39 = 9

0x32 = 2

0x64 = d

0x32 = 2

0x65 = e

Flag: picoCTF{gl17ch\_m3\_n07\_a4392d2e}

Links used: <https://www.asciitable.com/>

**Binary Exploitation**

1. *two-sum:* Can you solve this? What two positive numbers can make this possible: n1 > n1 + n2 OR n2 > n1 + n2Enter them here nc saturn.picoctf.net 52988. [Source](https://artifacts.picoctf.net/c/453/flag.c)
2. *basic-file-exploit:* The program provided allows you to write to a file and read what you wrote from it. Try playing around with it and see if you can break it!Connect to the program with netcat:$ nc saturn.picoctf.net 51109The program's source code with the flag redacted can be downloaded [here](https://artifacts.picoctf.net/c/141/program-redacted.c).
3. *Buffer overflow 0:*Smash the stack. Let's start off simple, can you overflow the correct buffer? The program is available [here](https://artifacts.picoctf.net/c/174/vuln). You can view source [here](https://artifacts.picoctf.net/c/174/vuln.c). And connect with it using:nc saturn.picoctf.net 65443

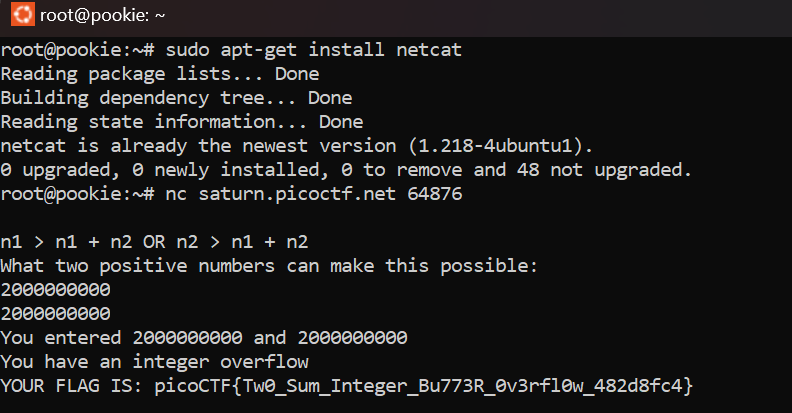
**Solution**

1. two-sum

(It should be noted that the reason I kept having to do sudo apt-get install for netcat is because any nc command refused to load unless I checked the installation like this.)

This challenge was time gated, but fortunately I did not need to use all the allotted time as the hints made it very clear what had to be done. I knew that this problem would not be mathematical, as it seemed to be more of an ‘exploit the system’ task.

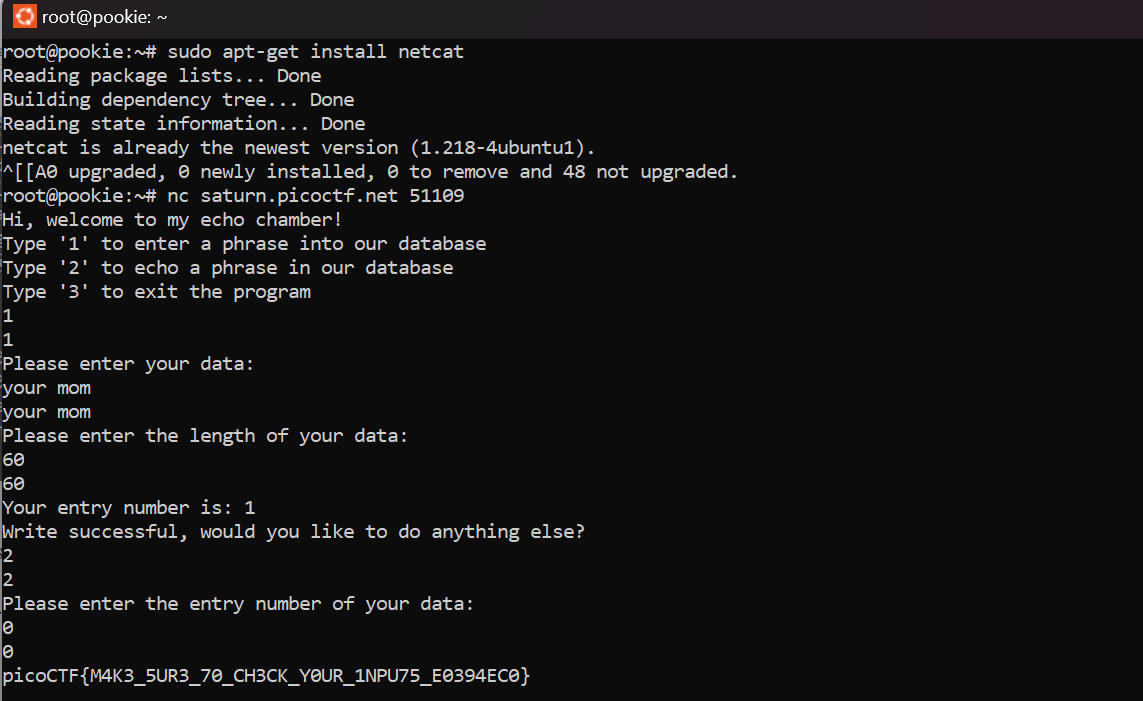
So, I tried to enter as big a number as possible, trying to exceed the binary limit of a 32-bit int data type, which I recalled was around 2,147,000,000. It worked and I got an unexpected output by exploiting the vulnerabilities of the code.



Flag: picoCTF{Tw0\_Sum\_Integer\_Bu773R\_0v3rfl0w\_482d8fc4}

Links used: <https://www.acunetix.com/blog/web-security-zone/what-is-integer-overflow/>

1. basic-file-exploit



Again, the instructions said to test out the edge and unexpected cases. Here, the conditional statement in the code has a puts() function which will display the value of flag variable if the result of the condition is True.

The Condition becomes true if the operand results in 0. So, if we put the value 0 at the point of Entry Number we’ll get the flag.

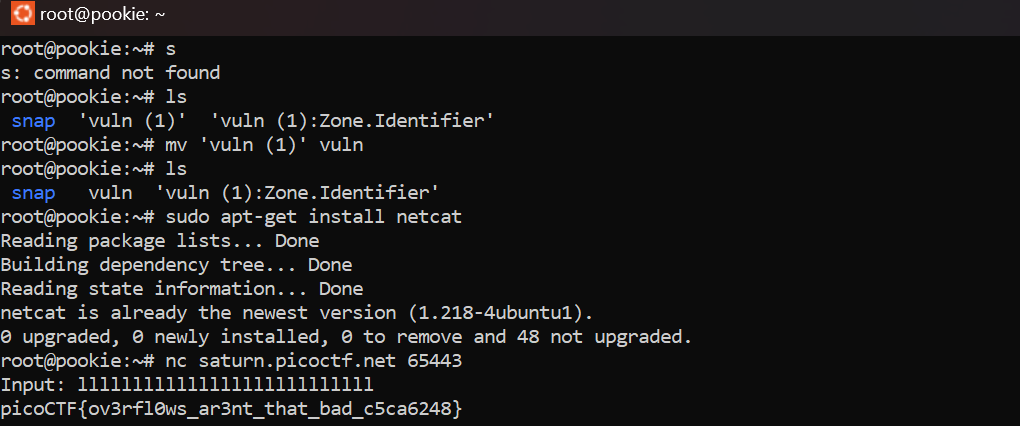
Flag: picoCTF{M4K3\_5UR3\_70\_CH3CK\_Y0UR\_1NPU75\_E0394EC0}

Links used: <https://www.we45.com/post/how-to-exploit-file-upload-vulnerabilities-and-how-to-fix-them>

1. buffer flow 0

The name of this task seemed very similar to the idea I applied in the first one under this category. I looked at the data overflow link I used in the previous task and also went through the hints.

Hint number 2 particularly was quite helpful as it basically confirmed exactly what I assumed would be the way out. Entered a random long string so it went past the flow limit and got the flag.



Flag: picoCTF{ov3rfl0ws\_ar3nt\_that\_bad\_c5ca6248}

Hints used: 2