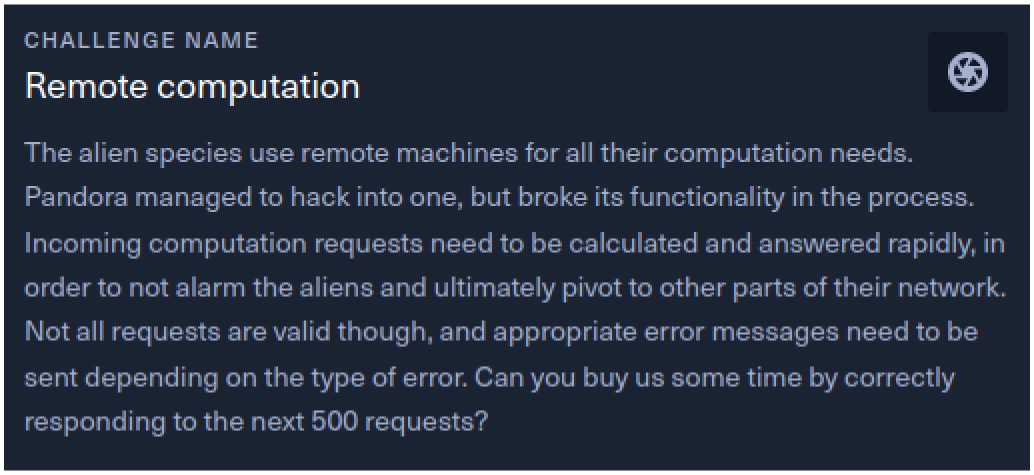
Challenge 5 – Remote Computation

Easy – Misc. Category

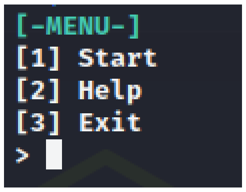


CHALLENGE NAME

Remote computation

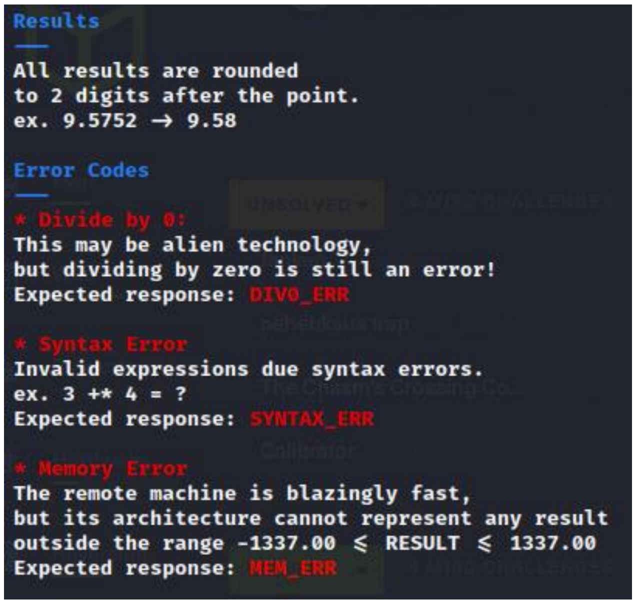
The alien species use remote machines for all their computation needs. Pandora managed to hack into one, but broke its functionality in the process. Incoming computation requests need to be calculated and answered rapidly, in order to not alarm the aliens and ultimately pivot to other parts of their network. Not all requests are valid though, and appropriate error messages need to be sent depending on the type of error. Can you buy us some time by correctly responding to the next 500 requests?

Similar to the last few challenges we are provided an IP address and a port number. Again, we must use Netcat to access the challenge.



```
[-MENU-]
[1] Start
[2] Help
[3] Exit
> 
```

Before diving in headfirst we check what the help option has to offer…

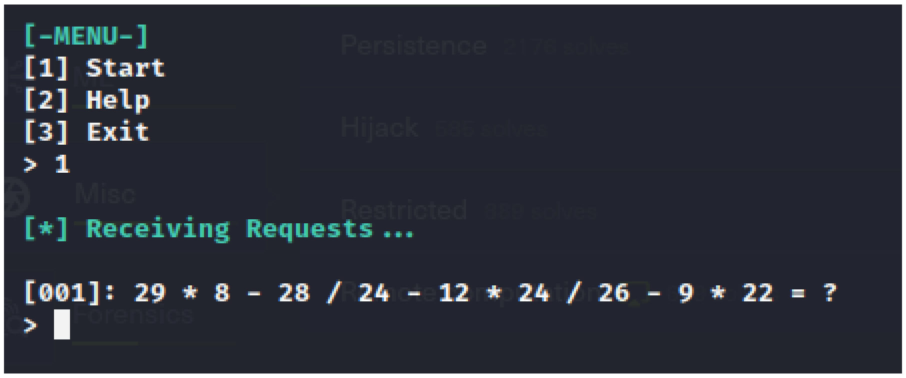


```
Results
──
All results are rounded
to 2 digits after the point.
ex. 9.5752 → 9.58

Error Codes
──
* Divide by 0:
This may be alien technology,
but dividing by zero is still an error!
Expected response: DIV0_ERR

* Syntax Error
Invalid expressions due syntax errors.
ex. 3 +* 4 = ?
Expected response: SYNTAX_ERR

* Memory Error
The remote machine is blazingly fast,
but its architecture cannot represent any result
outside the range -1337.00 ⩽ RESULT ⩽ 1337.00
Expected response: MEM_ERR
```

From this we can gather a few things…

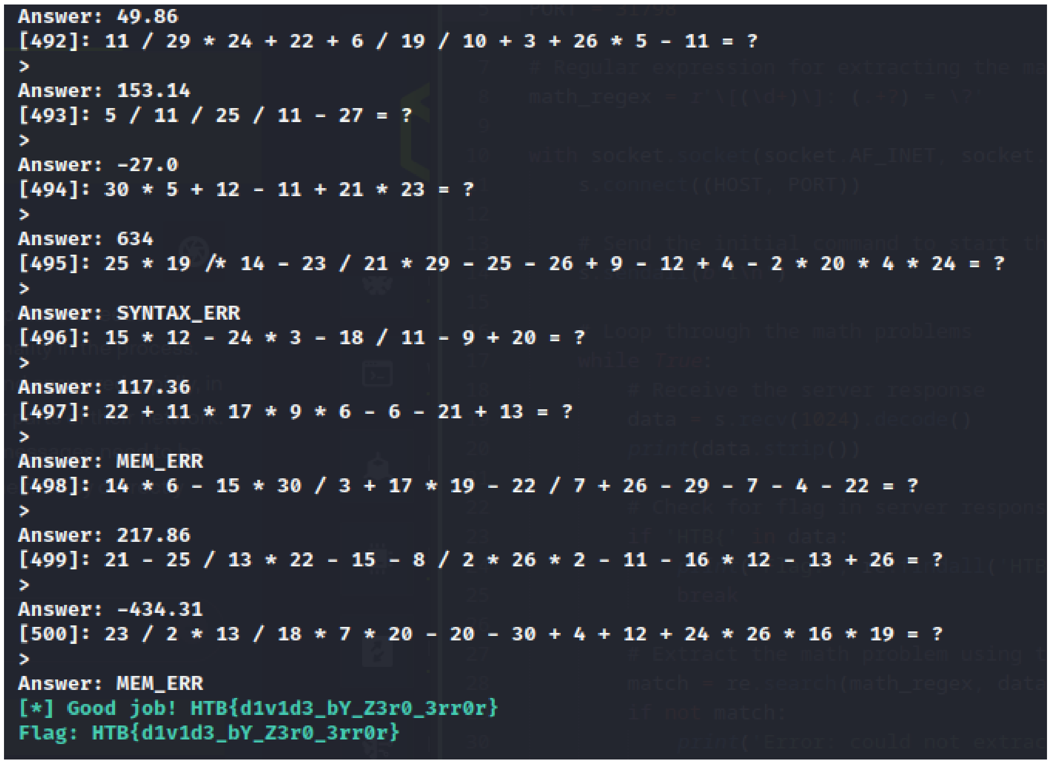
- We must answer any math question and round it to 2 decimal points.
- If we are asked to divide by zero we must respond with DIV0_ERR
- If we have an invalid expression we must respond with SYNTAX_ERR
- Any result larger than 1337.00 or smaller than -1337.00 (niiice) must give the response of MEM_ERR
- And that it is "blazingly fast"

After reading this it becomes obvious that this is another scripting challenge, but let's see if I can do it without a script…



```
[-MENU-]
[1] Start
[2] Help
[3] Exit
> 1

[*] Receiving Requests ...

[001]: 29 * 8 - 28 / 24 - 12 * 24 / 26 - 9 * 22 = ?
> 
```

I cannot.



```
Answer: 49.86
[492]: 11 / 29 * 24 + 22 + 6 / 19 / 10 + 3 + 26 * 5 - 11 = ?
>
Answer: 153.14
[493]: 5 / 11 / 25 / 11 - 27 = ?
>
Answer: -27.0
[494]: 30 * 5 + 12 - 11 + 21 * 23 = ?
>
Answer: 634
[495]: 25 * 19 /* 14 - 23 / 21 * 29 - 25 - 26 + 9 - 12 + 4 - 2 * 20 * 4 * 24 = ?
>
Answer: SYNTAX_ERR
[496]: 15 * 12 - 24 * 3 - 18 / 11 - 9 + 20 = ?
>
Answer: 117.36
[497]: 22 + 11 * 17 * 9 * 6 - 6 - 21 + 13 = ?
>
Answer: MEM_ERR
[498]: 14 * 6 - 15 * 30 / 3 + 17 * 19 - 22 / 7 + 26 - 29 - 7 - 4 - 22 = ?
>
Answer: 217.86
[499]: 21 - 25 / 13 * 22 - 15 - 8 / 2 * 26 * 2 - 11 - 16 * 12 - 13 + 26 = ?
>
Answer: -434.31
[500]: 23 / 2 * 13 / 18 * 7 * 20 - 20 - 30 + 4 + 12 + 24 * 26 * 16 * 19 = ?
>
Answer: MEM_ERR
[*] Good job! HTB{d1v1d3_bY_Z3r0_3rr0r}
Flag: HTB{d1v1d3_bY_Z3r0_3rr0r}
```

But Mathattack.py can!

```python
import socket
import re

HOST = '$IP'
PORT = $PortNumber

math_regex = r'\[(\d+)\]: (.+?) = \?'

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'1\n')

    while True:
        data = s.recv(1024).decode()
        print(data.strip())

        if 'HTB{' in data:
            print('Flag:', re.findall('HTB{.+}', data)[0])
            break

        match = re.search(math_regex, data)
        if not match:
            print('Error: could not extract question')
            continue
        problem_id = match.group(1)
        math_problem = match.group(2)

        try:
            math_answer = str(round(eval(math_problem), 2))
            if float(math_answer) < -1337 or float(math_answer) > 1337:
                math_answer = 'MEM_ERR'
        except ZeroDivisionError:
            math_answer = 'DIV0_ERR'
        except (SyntaxError, NameError):
            math_answer = 'SYNTAX_ERR'
        except MemoryError:
            math_answer = 'MEM_ERR'
        print('Answer:', math_answer)
        s.sendall((math_answer + '\n').encode())

    while True:
        data = s.recv(1024).decode()
        if not data:
            break
        print(data.strip())

    s.close()
```

This script is a bit more complicated than Requester.py and makes use of the socket and re (regex) modules.

```
import socket
import re
```

The socket module provides a set of classes that will let you create different kinds of sockets, connect to remote hosts using those sockets and manage those connections.

NOTE: For a more detailed description of the socket module, I really recommend the book "Black Hat Python, 2nd edition" by Justin Seitz and Tim Arnold.

The "re" module provides some very useful functions and methods to work with regular expressions in python. This is very important in this script because we want to take the output of the program and be able to make it simple for our script to read it and be correct when answering the math questions.

Then we need to assign our IP address and port of the target server…

```
HOST = '$IP'
PORT = $PortNumber
```

This next line essentially takes the math question output from the server and makes it easier to "digest".

```
math_regex = r'\[(\d+)\]: (.+?) = \?'
```

We know the output of the math question is always going to be something similar to…

[001]: 29 * 8 – 28 / 24 – 12 * 24 / 26 – 9 * 22 = ?

So, to break down each part the math_regex value…

- The **"r"** is a way to tell the script to treat each character in the following string as regular characters and allows us to use backslash as a way to escape special characters.
- The **(')** is how we tell it that this is the start of the regular string.
- The **"\[**" part of the value tells it that this is the start of the string that we are trying to solve
- The **"(\d+)"** is used to capture the digits in the output
- The **"\]"** is used to capture the characters that come after the number we captured (+, -, /, or *)
- Then we have **"(.+?)"** which will match the characters "= ?" in the output
- Finally, the **"= \?"** is the character we need to solve for.

The next 2 lines are to make the connection to the remote server using socket

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
```

This creates the TCP/IP socket "object" and assigns it to the "s" value.

Then connects to the assigned HOST and PORT

Because there is a menu when you first connect to the server we need to select the “1” option to start the math questions, we account for this with this line…

```
s.sendall(b'1\n')
```

Now we can start our loop to solve the math questions…

```
while True:
```

The next 2 lines are to receive the response from the server

```
data = s.recv(1024).decode()
        print(data.strip())
```

and check for the flag in the response…

```
if 'HTB{' in data:
            print('Flag:', re.findall('HTB{.+}', data)[0])
            break
```

NOTE: In hindsight, we didn’t really need this part because we are printing all of the output anyway, but it was better to be safe then sorry I guess.

The next few lines is how we extract the math problem using regular expression with the math_regex value we assigned earlier and the data value we assigned that is the response from the server…

```
match = re.search(math_regex, data)
        if not match:
            print('Error: could not extract question')
            continue
        problem_id = match.group(1)
        math_problem = match.group(2)
```

The following block of code is how we handle the “errors” that were provided to us in the game description by taking the “answer” of the question and running it through the requirements of an “error”…

```
            math_answer = str(round(eval(math_problem), 2))
            if float(math_answer) < -1337 or float(math_answer) > 1337:
                math_answer = 'MEM_ERR'
        except ZeroDivisionError:
            math_answer = 'DIV0_ERR'
        except (SyntaxError, NameError):
            math_answer = 'SYNTAX_ERR'
        except MemoryError:
            math_answer = 'MEM_ERR'
        print('Answer:', math_answer)
        s.sendall((math_answer + '\n').encode())
```

Finally, we wanted the script to be as verbose as possible and print out all the responses from the server…

```
    while True:
        data = s.recv(1024).decode()
        if not data:
            break
        print(data.strip())

    s.close()
```