Upon looking at the challenge you can see that it is a perl script to create CTF flags. There is also a text file with a hash, how it was hashed, and a time frame in which it was hashed. This makes you think that this is possibly a hash for the challenge flag.

hash.txt

```
1   Generated at 21:55-22:00 CST on 12/17/24
2   SHA2 - 256 64 rounds
3
4   633b4e367ccc9f770d94de429c85c0acd49daf4c4c79a449f4d44201f4aad456
```

From this you can determine that there is a vulnerability in the randomness that creates the flags. This would allow you to possibly recreate a flag.

```
# Get the current time as a seed
my $seed = time;

# Seed the random number generator
srand($seed);
```

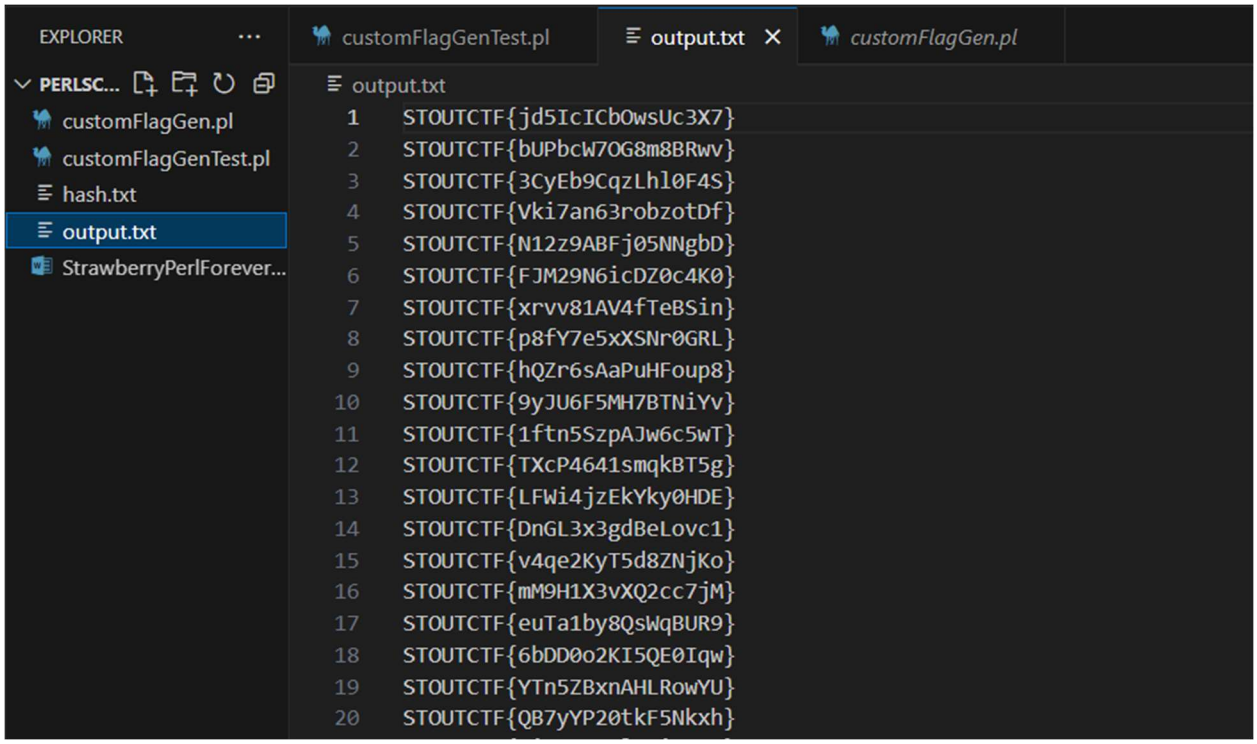
If you print out the seed at the end of the script you can see that it is generated in the standard Unix time format. This seed is then used in the srand function. Earlier you got the time range that the hash was created so theoretically you can generate all the Unix timestamps from that range and generate CTF flags from them. To do this you have to make a script to generate 5 minutes of timestamps on 12/17/24 at 9:55 CST. After looking at https://www.unixtimestamp.com/ you can determine that Unix time stamps are based on UTC. The conversion of CST to UTC is CST-6. This means that the time in UTC is 03:55-04:00 12/18/24.

This in Unix time stamp is between 1734515700 (03:55 UTC) and 1734516000 (04:00 UTC). As you can see there is a difference of 300 (60 seconds times 5 minutes). If you loop through these and add the flags to a text file you can get a full list of potential flags.

```
#!/usr/bin/perl
use strict;
use warnings;

# Welcome message
print "Generating a super random CTF flag";

# Create global variable to store the current time
my $startTime = 1734515700;
my $endTime = 1734516000;
```

```
# Open the file for writing
my $filename = "output.txt";
open(my $fh, '>', $filename) or die "Could not open file '$filename' $!";

while ($startTime < $endTime) {
    # Get the current time as a seed
    my $seed = $startTime;

    # Seed the random number generator
    srand($seed);

    # Define a function to generate a super dupper advanced and secure random
string
    sub random_string {
        my ($length) = @_;
        my @chars = ('a'..'z', 'A'..'Z', '0'..'9');
        my $random_string = '';

        for (1..$length) {
            $random_string .= $chars[int(rand(@chars))];
        }

        return $random_string;
    }

    # Generate the flag
    my $flag = "STOUTCTF{" . random_string(16) . "}";

    # Print the generated flag
    # Write to the file
    print $fh "$flag\n";

    # print $seed; # Debugging
    # Increase the start time by 1 second till you hit the end time
    $startTime++;
}

# Close the file
close($fh);
```

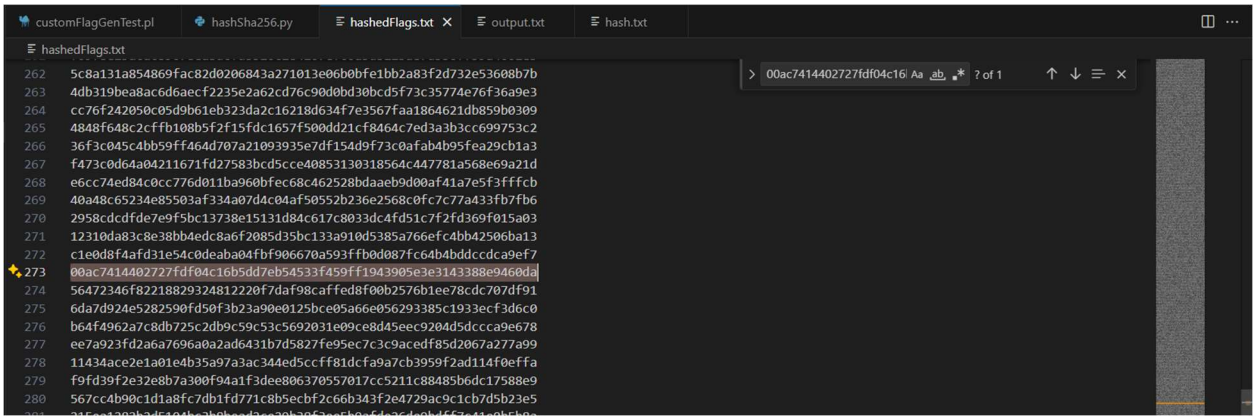
Now if you run the script it should print out the flags based on the unix time format to the specified text file. This can be seen below.



Now that we have the 300 possible flags we should make a copy of this file and hash the values. To do this I will make a python script to hash and line and write it into a new file.

```python
import hashlib

with open("output.txt") as infile, open("hashedFlags.txt", "a") as outfile:
    for line in infile:
        flag = line.strip()
        if not flag:  # Check for empty lines
            print("Empty line found, skipping...")
            continue
        print(f"Processing flag: {flag}")  # Debug print
        hash_object = hashlib.sha256(flag.encode())
        hashed_flag = hash_object.hexdigest()
        print(f"Hashed flag: {hashed_flag}")  # Debug print
        outfile.write(hashed_flag + "\n")
```

This script will then print the hashes. You can then find the hash value from hash.txt.



As you can see Line 273 has the hash we are looking for. That flag is: