

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) . "}";

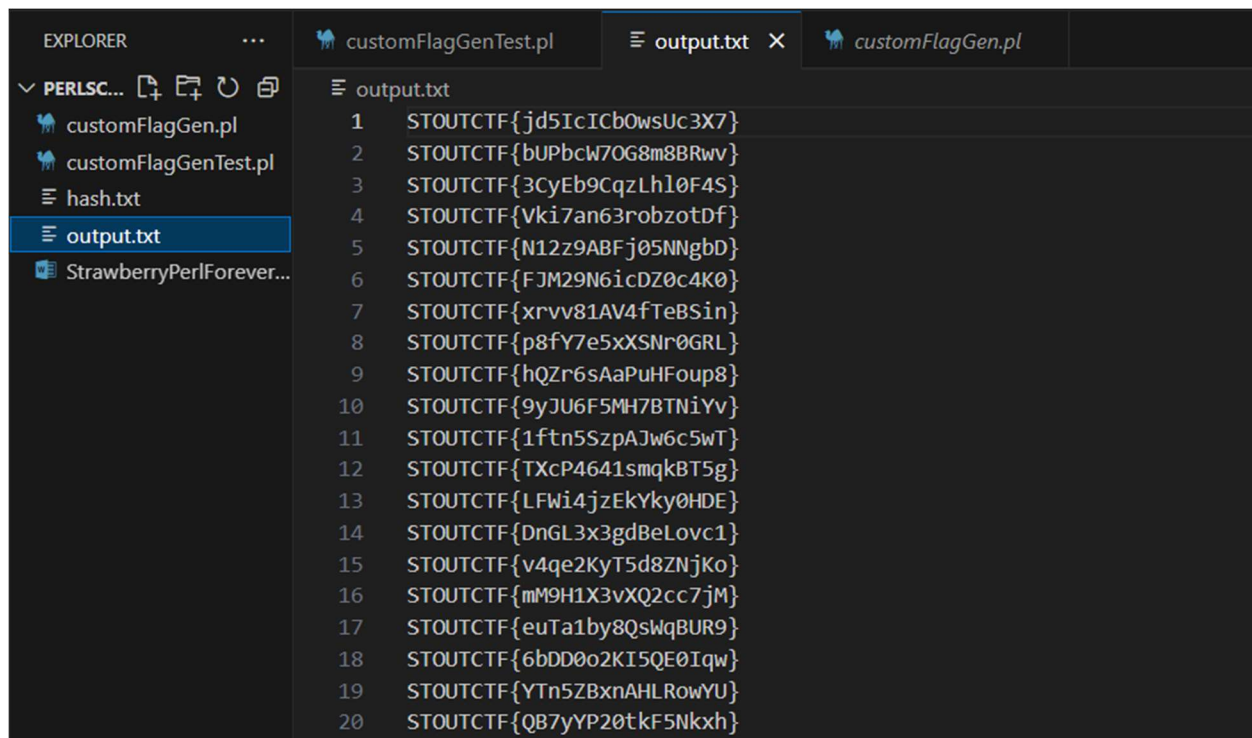
    # Print the generated flag
    # Write to the file
    print $fh "$flag\n";

    # 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.



The screenshot shows a code editor with three tabs: 'customFlagGenTest.pl', 'output.txt', and 'customFlagGen.pl'. The 'output.txt' tab is active, displaying a list of 20 lines, each containing a flag in the format 'STOUTCTF{...}'. The flags are numbered 1 through 20. The Explorer panel on the left shows the file structure, with 'output.txt' selected.

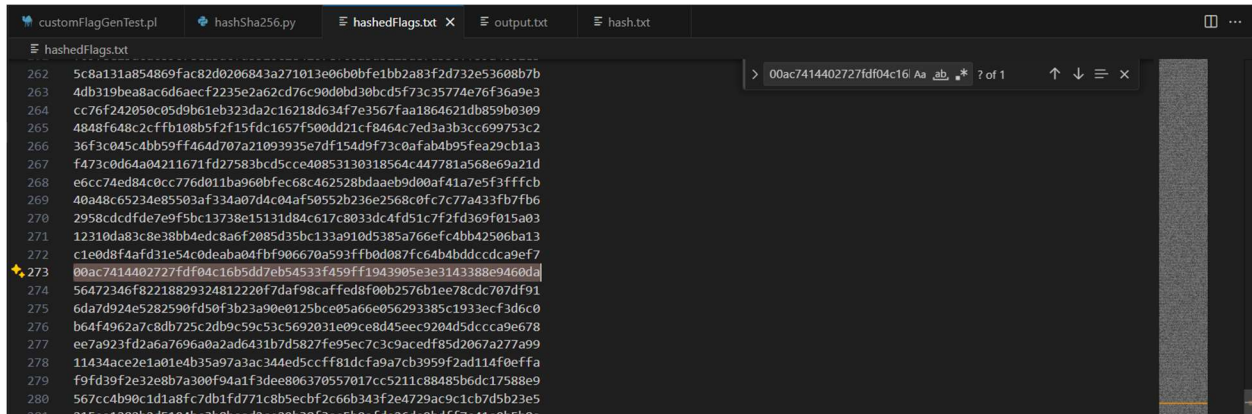
```
1 STOUTCTF{j d5IcICb0wsUc3X7}
2 STOUTCTF{bUPbcw7OG8m8BRwv}
3 STOUTCTF{3CyEb9CqzLh10F4S}
4 STOUTCTF{Vki7an63robzotDf}
5 STOUTCTF{N12z9ABFj05NNGbD}
6 STOUTCTF{FJM29N6icDZ0c4K0}
7 STOUTCTF{xrvv81AV4fTeBSin}
8 STOUTCTF{p8fY7e5xXSNr0GRL}
9 STOUTCTF{hQZr6sAaPuHFoup8}
10 STOUTCTF{9yJU6F5MH7BTNiYv}
11 STOUTCTF{1ftn5SzpAJw6c5wT}
12 STOUTCTF{TXcP4641smqkBT5g}
13 STOUTCTF{LFWi4jzEkYky0HDE}
14 STOUTCTF{DnGL3x3gdBeLovc1}
15 STOUTCTF{v4qe2KyT5d8ZNjKo}
16 STOUTCTF{mM9H1X3vXQ2cc7jM}
17 STOUTCTF{euTa1by8QswqBUR9}
18 STOUTCTF{6bDD0o2KI5QE0Iqw}
19 STOUTCTF{YTn5ZBxnAHLRowYU}
20 STOUTCTF{QB7yYP20tkF5Nkxh}
```

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.

```
import hashlib

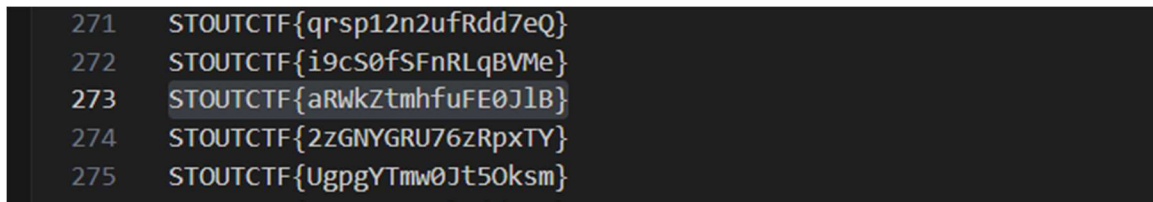
with open("output.txt") as infile, open("hashedFlags.txt", "a") as outfile:
    for line in infile:
        flag = line.strip()
        if not flag: # Check for empty lines
            print("Empty line found, skipping...")
            continue
        print(f"Processing flag: {flag}") # Debug print
        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.



```
customFlagGenTest.pl hashSha256.py hashedFlags.txt output.txt hash.txt
hashedFlags.txt
262 5c8a131a854869fac82d0206843a271013e06b0bfe1bb2a83f2d732e53608b7b
263 4db319bea8ac6daecf2235e2a62cd76c90d0bd30bcd5f73c35774e76f36a9e3
264 cc76f242050c85d9b61eb323da2c16218d634f7e3567faa1864621db859b0309
265 4848f648c2cfff108b5f2f15fdc1657f500dd21cf8464c7ed3a3b3cc699753c2
266 36f3c045c4bb59ff464d707a21093935e7df154d9f73c0afab4b95fea29cb1a3
267 f473c0d64a04211671fd27583bcd5cce40853130318564c447781a568e69a21d
268 e6cc74ed84c0cc776d011ba960bfec68c462528bdaeb9d00af41a7e5f3fffc
269 40a48c65234e8503af334a07d4c04af50552b236e2568c0fc7c77a433fb7fb6
270 2958cdcdfde7e9f5bc13738e15131d84c617c8033dc4fd51c7f2fd369f015a03
271 12310da83c8e38bb4edc8a6f2085d35bc133a910d5385a766efc4bb42506ba13
272 c1e0d8f4afd31e54c0deaba04fbf906670a593ffb0d087fc64b4bddccdc9ef7
273 00ac7414402727dfdf04c16b5dd7eb54533f459ff1943905e3e3143388e9460da
274 56472346f82218829324812220f7daf98caffed8f00b2576b1ee78cdc707df91
275 6da7d924e528250fd50f3b23a90e0125bce05a66e056293385c1933ecf3d6c0
276 b64f4962a7c8db725c2db9c59c53c5692031e09ce8d45ee9204d5dcca9e678
277 ee7a923fd2a6a7696a0a2ad6431b7d5827fe95ec7c3c9acedf85d2067a277a99
278 11434ace2e1a01e4b35a97a3ac344ed5ccff81dcfa9a7cb3959f2ad114f0effa
279 f9fd39f2e32e8b7a300f94a1f3dee806370557017cc5211c88485b6dc17588e9
280 567cc4b90c1d1a8fc7db1fd771c8b5ecbf2c66b343f2e4729ac9c1cb7d5b23e5
```

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



```
271 STOUTCTF{qrsp12n2ufRdd7eQ}
272 STOUTCTF{i9cS0fSFnRLqBVMe}
273 STOUTCTF{aRWkZtmhfufE0JlB}
274 STOUTCTF{2zGNYGRU76zRpxTY}
275 STOUTCTF{UgpgYTmw0Jt50ksm}
```