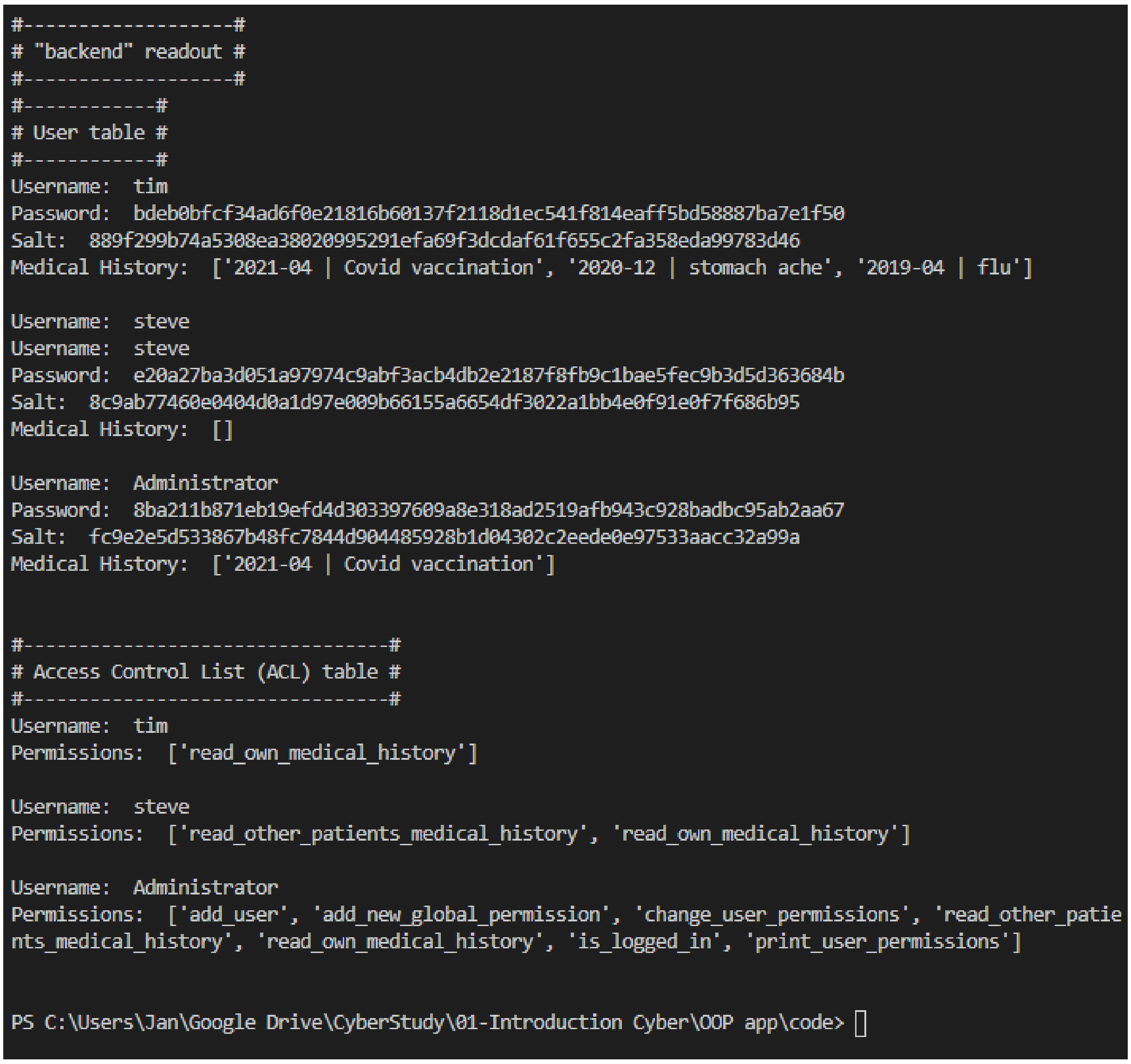
# System Test

# System Test

## 1 Auth Module – User authentication

### 1.1 Test Data

This version of the ASMIS stores data in pickle files. A backend like SQL can be simulated by this. To test the ASMIS the pickle backend (compare Figure 1) was filled with random test data such as e.g. user credentials, patient history & privileges. In Figure 1 it can be seen, that passwords are not stored in clear but as salted hashes.

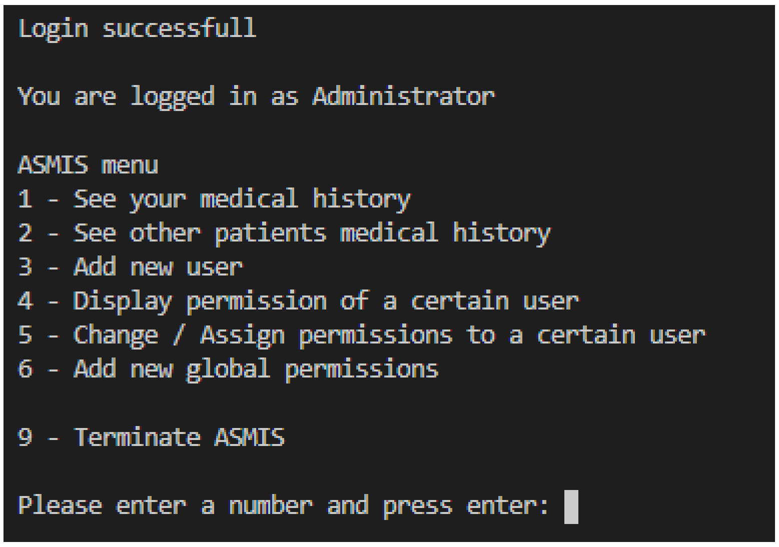


```
#-------------------#
# "backend" readout #
#-------------------#
#------------#
# User table #
#------------#
Username:  tim
Password:  bdeb0bfcf34ad6f0e21816b60137f2118d1ec541f814eaff5bd58887ba7e1f50
Salt:  889f299b74a5308ea38020995291efa69f3dcdaf61f655c2fa358eda99783d46
Medical History:  ['2021-04 | Covid vaccination', '2020-12 | stomach ache', '2019-04 | flu']

Username:  steve
Username:  steve
Password:  e20a27ba3d051a97974c9abf3acb4db2e2187f8fb9c1bae5fec9b3d5d363684b
Salt:  8c9ab77460e0404d0a1d97e009b66155a6654df3022a1bb4e0f91e0f7f686b95
Medical History:  []

Username:  Administrator
Password:  8ba211b871eb19efd4d303397609a8e318ad2519afb943c928badbc95ab2aa67
Salt:  fc9e2e5d533867b48fc7844d904485928b1d04302c2eede0e97533aacc32a99a
Medical History:  ['2021-04 | Covid vaccination']


#---------------------------------#
# Access Control List (ACL) table #
#---------------------------------#
Username:  tim
Permissions:  ['read_own_medical_history']

Username:  steve
Permissions:  ['read_other_patients_medical_history', 'read_own_medical_history']

Username:  Administrator
Permissions:  ['add_user', 'add_new_global_permission', 'change_user_permissions', 'read_other_patie
nts_medical_history', 'read_own_medical_history', 'is_logged_in', 'print_user_permissions']


PS C:\Users\Jan\Google Drive\CyberStudy\01-Introduction Cyber\OOP app\code> 
```

*Figure 1 – ASMIS backend test data*

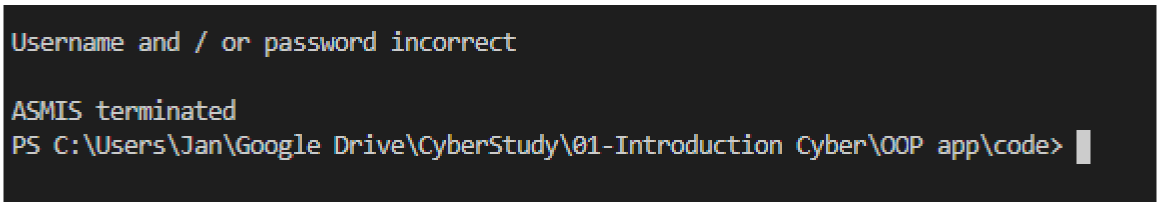
### 1.2 Results obtained

For Authentication a user login was implemented. The test results show, that with incorrect credentials access to the system is denied (compare Figure 3). The test results also show, that with correct credentials access is granted (compare Figure 2).



```
Login successfull

You are logged in as Administrator

ASMIS menu
1 - See your medical history
2 - See other patients medical history
3 - Add new user
4 - Display permission of a certain user
5 - Change / Assign permissions to a certain user
6 - Add new global permissions

9 - Terminate ASMIS

Please enter a number and press enter:
```

*Figure 2 – Login successful*



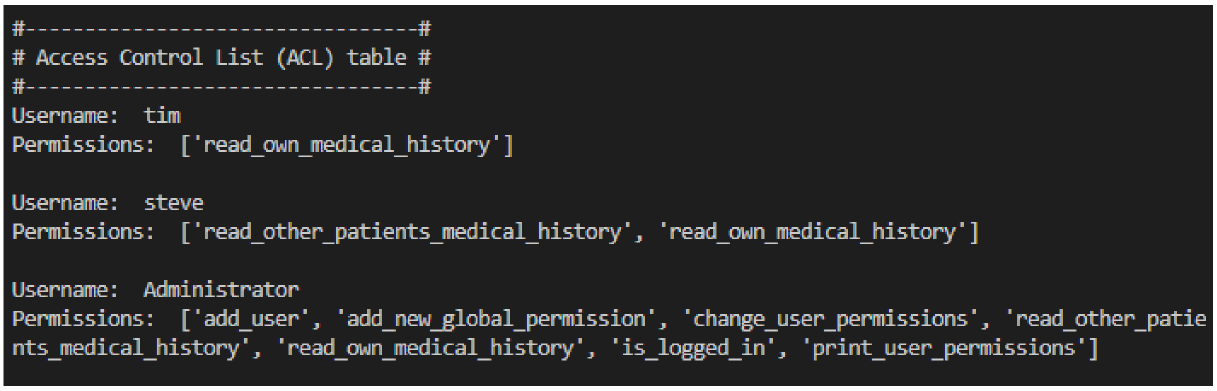
```
Username and / or password incorrect

ASMIS terminated
PS C:\Users\Jan\Google Drive\CyberStudy\01-Introduction Cyber\OOP app\code>
```

*Figure 3 – Login not successful*

## 2 Auth Module – User authorization

### 2.1 Test Data

The same test data as shown in section 1.1. was used. For this test case the Access Control Lists as shown in Figure 4 is important, since it defines which user is allowed to perform which task.



```
#--------------------------------#
# Access Control List (ACL) table #
#--------------------------------#
Username:  tim
Permissions:  ['read_own_medical_history']

Username:  steve
Permissions:  ['read_other_patients_medical_history', 'read_own_medical_history']

Username:  Administrator
Permissions:  ['add_user', 'add_new_global_permission', 'change_user_permissions', 'read_other_patie
nts_medical_history', 'read_own_medical_history', 'is_logged_in', 'print_user_permissions']
```

*Figure 4 - ASMIS backend test data Access Control List (ACL)*

### 2.2 Results obtained

This ASMIS has several functions, that should only be used by privileged users such as

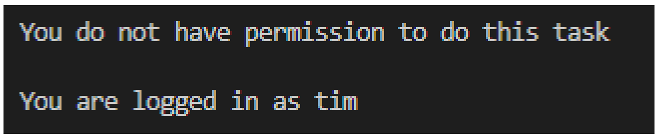
- 2 - See other patients medical history
- 3 - Add new user
- 4 - Display permission of a certain user
- 5 - Change / Assign permissions to a certain user

The ASMIS has some function, that can be used by all user such as

- 1 - See your medical history
- 9 - Terminate ASMIS

Which user can use which functionality can however be altered by a user with sufficient privileges. To do so the user needs to start function 5 – Change / Assign permissions to a certain user.

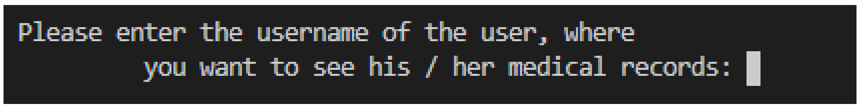
To verify, that authorization is properly functioning several test cases were conducted (according to the ACL as shown in Figure 4). It was for example tested if a user with insufficient privileges is able to access a function he / she should not be able to access. A screenshot of one of the test cases is given as follows:



```
You do not have permission to do this task

You are logged in as tim
```

*Figure 5 – authorization denied*

On the other hand it was also verified within several test cases, that user with sufficient privileges can use the functions as defined in the ACL. A screenshot of one of the test cases is given as follows. Here the user got access to a function, that shows medical records of any user:



```
Please enter the username of the user, where
         you want to see his / her medical records:
```

*Figure 6 – authorization successful*

## 3 Salted hash

### 3.1 Test Data

The following data was used. The salted hashes are computed by the implementation of the ASMIS.

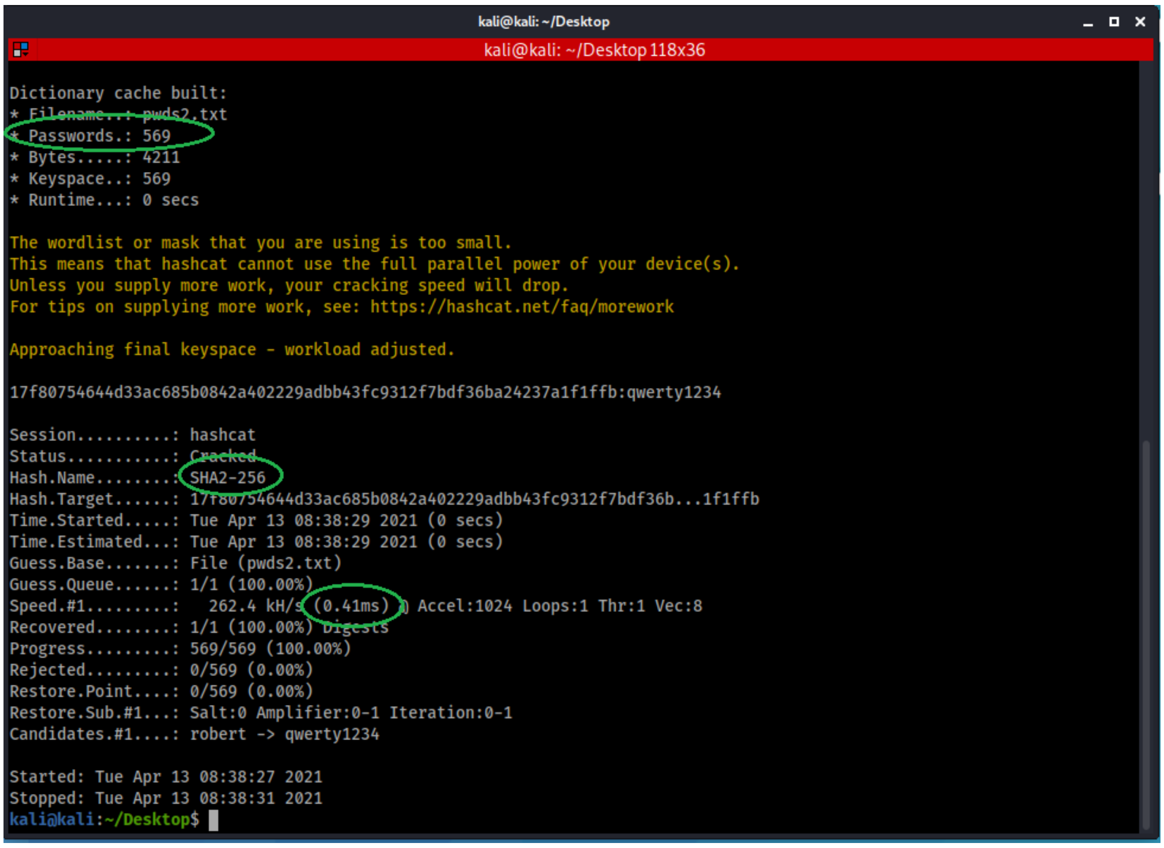
| password | Hash (SHA256) | Salted Hash |
|---|---|---|
| qwerty1234 | 7f80754644d33ac685b0842a402229ad bb43fc9312f7bdf36ba24237a1f1ffb | bdeb0bfcf34ad6f0e21816b60137f2118 d1ec541f814eaff5bd58887ba7e1f50 |
| 5pdjZkMRay | cb336eddac00d29b183cb1f493d45ec98 6d63d5297aa7ca0dd9fb34decdc37f8 | f782dc7691c73852ccad3361b00f4281c b17c1dd64e9738506d7738aa1efcaa3 |

## 3.2 Results obtained

Dictionary attack using hashcat (compare Figure 7) was used to obtain results. Hashcat was running on a standard PC having standard computational power.

| Type | Password | Time to crack password |
|---|---|---|
| Password in cleartext | qwerty1234 | 0 seconds (No need to crack, since password is in clear text) |
| | 5pdjZkMRay | |
| Password hashed (SHA256) | qwerty1234 | A few seconds |
| | 5pdjZkMRay | Could not be cracked within reasonable time in the test case.<br>Note: Time to crack password can be a few minutes, if rainbow tables are used. |
| Password Salted Hash (Salt also known to the attacker) | qwerty1234 | Could not be cracked within reasonable time in the test case.<br>Note: Time to crack password can be a few minutes, if rainbow tables are used. |
| | 5pdjZkMRay | Could not be cracked within reasonable time in the test case.<br>Note: Rainbow tables cannot easily be used. Some considerable computing power still necessary |

As the table shows salt hashing a password is not the one solution for all problems. A strong and unique password nonetheless should always be used. Salt hashing a password however always adds an additional burden to the hacker, since the can’t use pre computed hashes.



```
Dictionary cache built:
* Filename..: pwds2.txt
* Passwords.: 569
* Bytes.....: 4211
* Keyspace..: 569
* Runtime...: 0 secs

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

17f80754644d33ac685b0842a402229adbb43fc9312f7bdf36ba24237a1f1ffb:qwerty1234

Session..........: hashcat
Status...........: Cracked
Hash.Name........: SHA2-256
Hash.Target......: 17f80754644d33ac685b0842a402229adbb43fc9312f7bdf36b...1f1ffb
Time.Started.....: Tue Apr 13 08:38:29 2021 (0 secs)
Time.Estimated...: Tue Apr 13 08:38:29 2021 (0 secs)
Guess.Base.......: File (pwds2.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   262.4 kH/s (0.41ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 569/569 (100.00%)
Rejected.........: 0/569 (0.00%)
Restore.Point....: 0/569 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: robert -> qwerty1234

Started: Tue Apr 13 08:38:27 2021
Stopped: Tue Apr 13 08:38:31 2021
kali@kali:~/Desktop$
```

*Figure 7 – cracking of SHA265*

# System Test

## 4 Password policy

### 4.1 Test Data

A simple python script was coded to demonstrate, that the time necessary to crack a password does increase with the length of the password. For the ease of simulation passwords consisting of only numbers, where used.

To simulate the fact, that the ASMIS will be web based, and that a realistic brute force attack, likely done via proxy servers or TOR network has some delay, a short pause  was integrated in the script to simulate this.

```
n=0 #initialize the counter
while True:
    time.sleep(0.0000000001)  #simulate communication time of the internet
    n=n+1  #increase increment
    if n==99999: #this will simulate a password with 5 digits
        end = time.time() #measure end time
        print(end - start) #display delta time aka time it took
        break
```

### 4.2 Results obtained

The following results are not to be interpreted absolute, meaning that a password of 5 characters needs x amount of time. The following results can only be interpreted in relation to one another, meaning that brute forcing a password of 6 characters needs more time than brute forcing a password of 6 characters, since the bar for 6 characters is many times bigger then the bar for 5 characters (compare Figure 8).

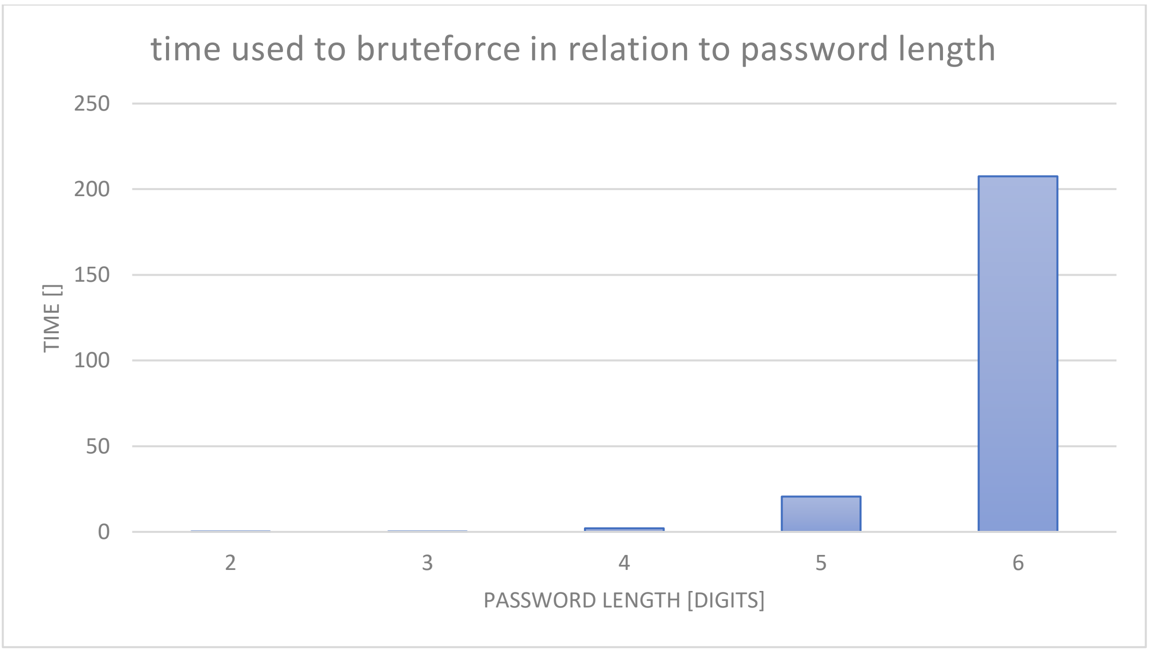


*Figure 8 - time used to brute force in relation to password length*

The results show, that with increasing password the time for brute forcing increases significantly. A sufficiently long password however does not help against dictionary attacks or credential stuffing. To defend against this, other measures need to be implemented. Password minimum length is nonetheless deemed a very good addition to your defence in depth strategy.