CW2

Computer Security Assignment 2

JANUARY 19, 2022

AS2634@KENT.AC.UK

Contents

Part 1
Week 17

1. Introduction

This week covers some password cracking exercises including the following:

- Creating a login system to help understand how hashing and salting works and why it is important when storing passwords on server.
- Demonstrating a simple and personalized dictionary attack using leaked password database.
- Breaking a graphical password i.e. password of an android phone

2. Learning Outcomes

To attempt each exercise , I wrote scripts using Python programming language. Following are some topics that I learned during this exercise:

- **What is hashing and why passwords are hashed?**

I learned about hashing technique in which a given string/key is converted into another value which is of a fixed length irrespective the size of input using a mapping function called the hash function. A good hash function is supposed to be fast, should be irreversible and should not produce collisions. It is a one-way function. It is a common way to store passwords on a server so that if server gets compromised, hackers do not get access to passwords directly but rather a hashed value of passwords otherwise it could comprise user's information on multiple accounts since a number of users use same password on multiple accounts.

- **What are rainbow tables?**

Rainbow tables are databases that store plaintext values of different hashes. Rainbow tables are commonly used by hackers to get the plaintext values of leaked hashes from compromised servers without performing any excessive computation and using comparison function.

- **What is salting and how it prevents from rainbow attacks?**

Salting is a process where a random value is appended at the end of the password before it is hashed to protect the passwords from rainbow table attacks. Salting does not guarantee password protection but increases the time and possibilities for hackers to guess passwords which in turn adds additional protection. Salt is randomly generated for each password and is also stored on server along with password.

Appending salt with password increases computation time for hackers when they perform brute force to guess the passwords. If salt value is not appended then once a hacker gets a leaked password database containing hashes, it can simply compare the hash value of leaked passwords from server and rainbow tables and if a value matches, they can simply get the plaintext/ password for that hash. Appending salt adds additional calculation because now simple comparison is not required to guess the password for any hash but rather require calculating hash by combining plaintext of each password with leaked salt and generating hash and then comparing to match from leaked hash from server.

3. Observations

Computation time and power are important considerations in cryptography. I observed how vulnerable data can be if we do not use strong passwords as it is easy to crack if password is based on some data which is exposed. Brute force needs good computational power as for exercise 3 possibilities were limited but it took time for the brute force program to crack the password. In real world scenarios, possibilities are considerably huge and therefore require more sophisticated or smart brute force algorithms along with additional computational power.

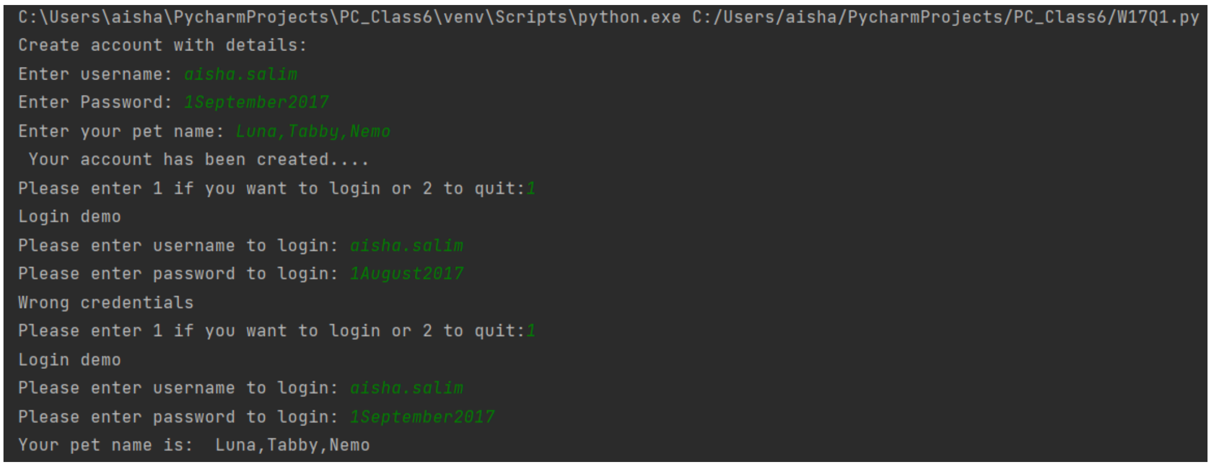
4. Challenges

It was a challenge to generate different combinations for the personal information as manually generating different combinations is not effective and is very time consuming. I found a Python based library 'Permutations' that generated different combinations for me.

5. Proof of Work

Exercise 1

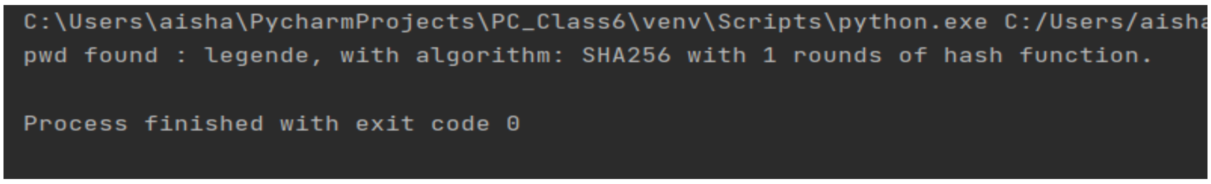
For this exercise, I have attached screenshots for successful case when user is entering correct credentials and for unsuccessful case when user is giving wrong credentials.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W17Q1.py
Create account with details:
Enter username: aisha.salim
Enter Password: 1September2017
Enter your pet name: Luna,Tabby,Nemo
 Your account has been created....
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim
Please enter password to login: 1August2017
Wrong credentials
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim
Please enter password to login: 1September2017
Your pet name is:  Luna,Tabby,Nemo
```

Exercise 2

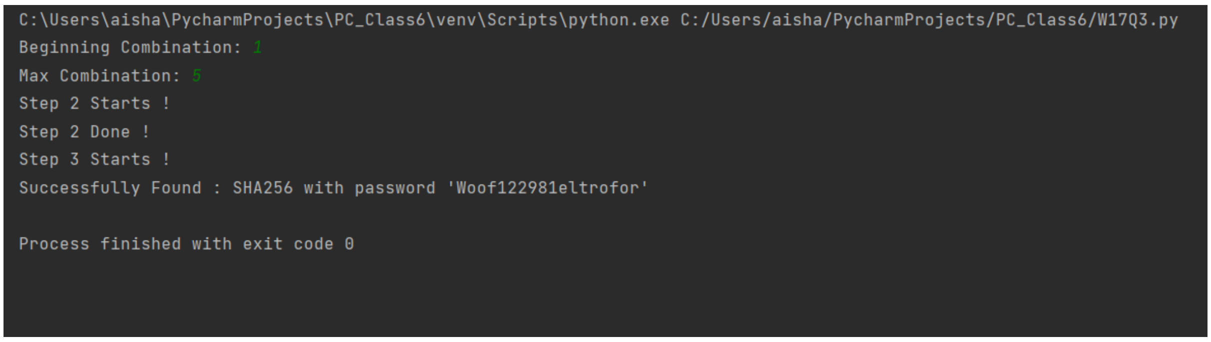
For this question, I checked with hash algorithm SHA-1 and then to SHA-256. For each SHA algorithm, I programmed to checked till 5 rounds of hash function. It is common practice among hackers to release hacked password hashes. It was possible to use passwords released from another website because passwords for this website were not salted but only hashed, hence it was computationally easy to compare hashes from a leaked password website with the hash given and guess the password.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha
pwd found : legende, with algorithm: SHA256 with 1 rounds of hash function.

Process finished with exit code 0
```

Exercise 3

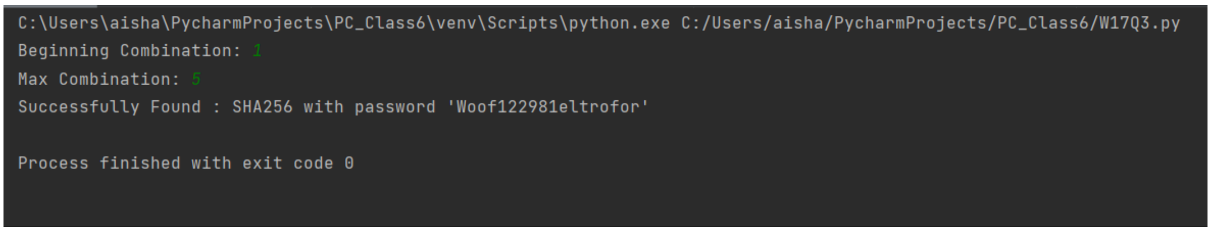
I created a dictionary based on personal information given and used a Permutations (Python) library to make different combinations from the given personal information. My program takes dynamic input for maximum and minimum number of combinations to make from the given personal information. I am converting each personal information into upper case, lower case, handling different date formats and replaced white space to underscore('_') and also trimmed off any white space character to check for diverse combinations. SHA-1 and SHA-256 algorithms are used for one round of hash function.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W17Q3.py
Beginning Combination: 1
Max Combination: 5
Step 2 Starts !
Step 2 Done !
Step 3 Starts !
Successfully Found : SHA256 with password 'Woof122981eltrofor'

Process finished with exit code 0
```

Exercise 3 (Additional Challenge)

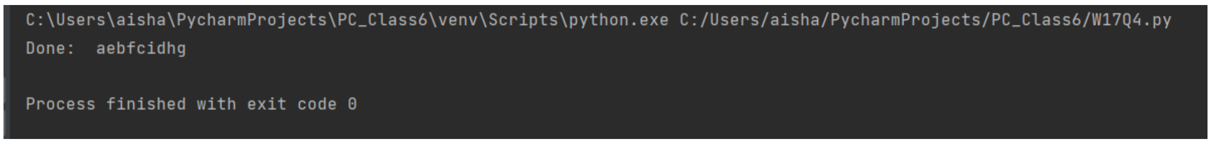
My program was able to check for additional challenge without making any changes as it is already handling case toggling.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W17Q3.py
Beginning Combination: 1
Max Combination: 5
Successfully Found : SHA256 with password 'Woof122981eltrofor'

Process finished with exit code 0
```

Exercise 4

For this exercise length of password is already given which is 9 so I created 9 length permutations of elements [a,b,c,d,e,f,g,h,i] and then hashed each combination with SHA-1 with one round of hash function as it is already specified in the exercise.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W17Q4.py
Done:  aebfcidhg

Process finished with exit code 0
```

Week 18

1. Introduction

This week covered some exercises about different methods of user authentication:

- User authentication with passwords containing emojis
- Using any file as a password for user authentication
- OTP based password authentication
- Biometric based authentication system

2. Learning Outcomes

Different ways of Authentication

There are multiple ways of user authentication other than textual password-based authentication such as

1.What you know (Knowledge-based authentication example Textual and emoji-based passwords): Authenticating user based on what he/she knows i.e., based on the knowledge a user possesses. Some of its examples include textual based passwords and emoji based passwords.

2. What you have (Example: File based password ; OTP based password)

Such system uses any possession by user as a source of its identification. It may include some security keys which changes its string key based on time, if a user enters the correct key generated at the correct time, it can be used as a source of authentication for that user. Other examples include using any file as your password or using any OTP generated value to proof your identity.

3. Who you are (Example: Biometric based authentication)

Biometric based authentication systems can use a person's **physical attributes** such as fingerprint, retina scan or **behavioral properties** such as handwriting, gestures, gait, or **biological properties** such as perspiration, body odor to identify a user. It works by comparing two sets of data; pre acquired sample/ enrolled template and live data acquired from user to be authenticated. There can be difference in the two data sets, and we need to decide a threshold which is accepted to let user gets successful authentication. Two factors are important when deciding threshold for a biometric system FRR (False Rejection Rate) which specifies the rate to which a user is falsely rejected and FAR (False Acceptance Rate) which specifies the rate to which a user is falsely accepted as another user during authentication. We need to find a threshold where both FRR and FAR are minimized. Usually, biometric based authentication is accompanied by any other way of authentication because if a user is falsely rejected, he can use some other authentication to get authorization.

3. Observations

Textual based passwords are still the most widely used in user authentication and they are not getting obsolete anytime soon. It is also the simplest technique used as compared to other techniques which may require additional equipments such as biometric based user authentication require special equipment to get user sample and then to verify the user every time. Similarly, emoji-based passwords may require a special keyboard that enables users to enter such passwords. These limitations make most of the systems to still use textual based passwords as they are better in usability.

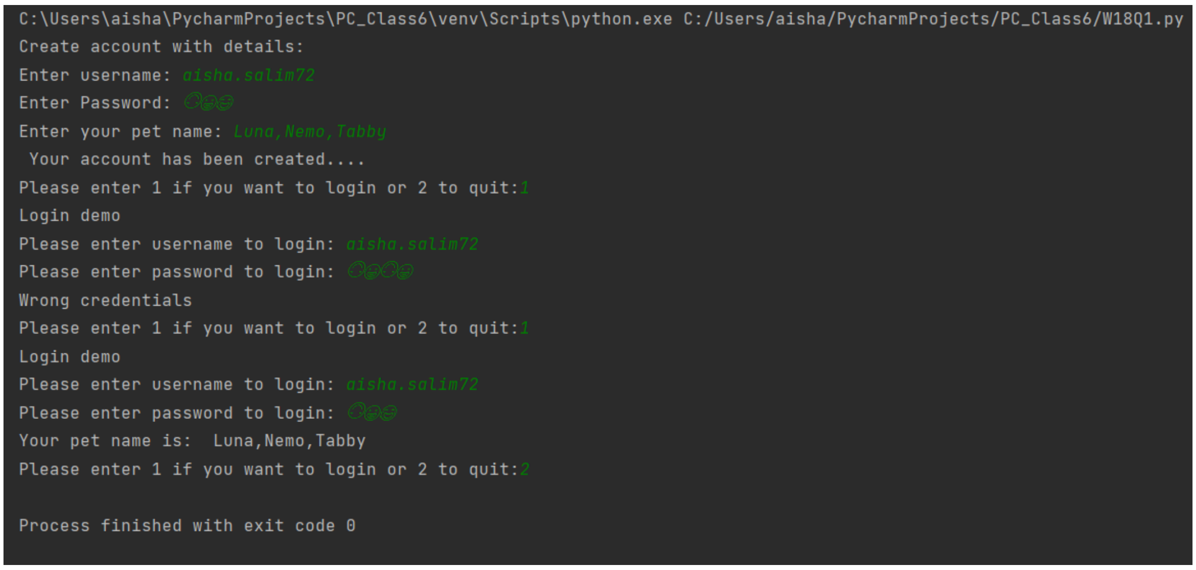
4. Challenges

I found difficulty solving the biometric based authentication system as there is no definite way to decide the threshold of accepted difference at which user is allowed to get authenticated. I used an algorithm to find the threshold value to accept a user at which FRR and FAR are at minimum. The algorithm sums the value of FRR and FAR as a score. It first calculates the averages of values given in the question for the Authentic User (Alice) and Intruder (Eve) and then gets an average of the two. Then algorithm generates scores for all three values and continues rejecting the one value for which score is the highest. It stops when score gets constant. It then calculates the average of the two values and decides the accepted threshold from that. The algorithm is inspired by dichotomy algorithm which keeps re-calculating until value gets constant.

5. Proof of Work

Exercise 1

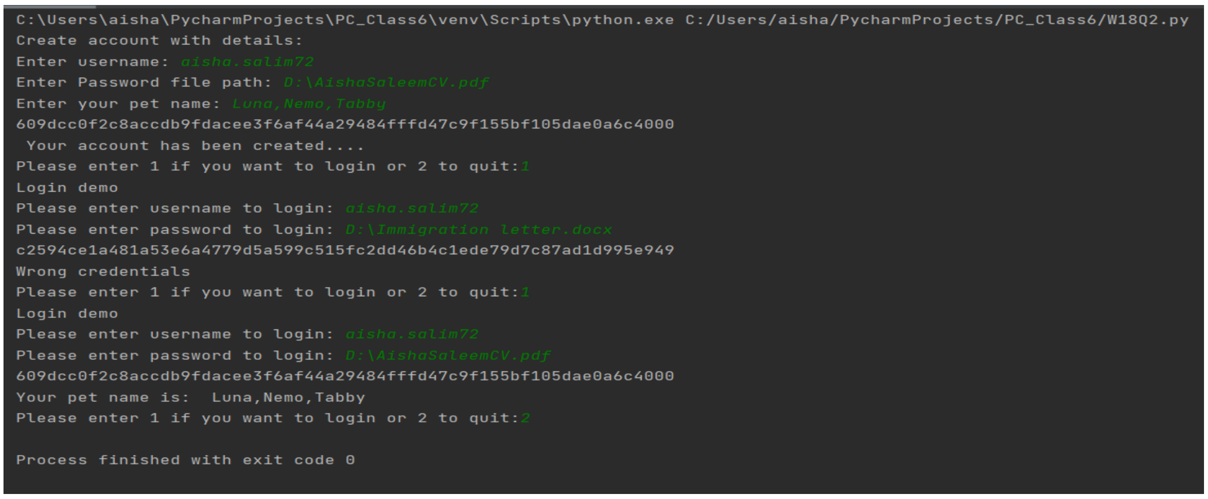
I have used utf-8 encoding to display emojis properly on console. In this simple example I have used three different emojis as a password to register an account.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W18Q1.py
Create account with details:
Enter username: aisha.salim72
Enter Password: 😶😄😊
Enter your pet name: Luna,Nemo,Tabby
 Your account has been created....
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim72
Please enter password to login: 😶😄😶😄
Wrong credentials
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim72
Please enter password to login: 😶😄😊
Your pet name is:  Luna,Nemo,Tabby
Please enter 1 if you want to login or 2 to quit:2

Process finished with exit code 0
```

Exercise 2

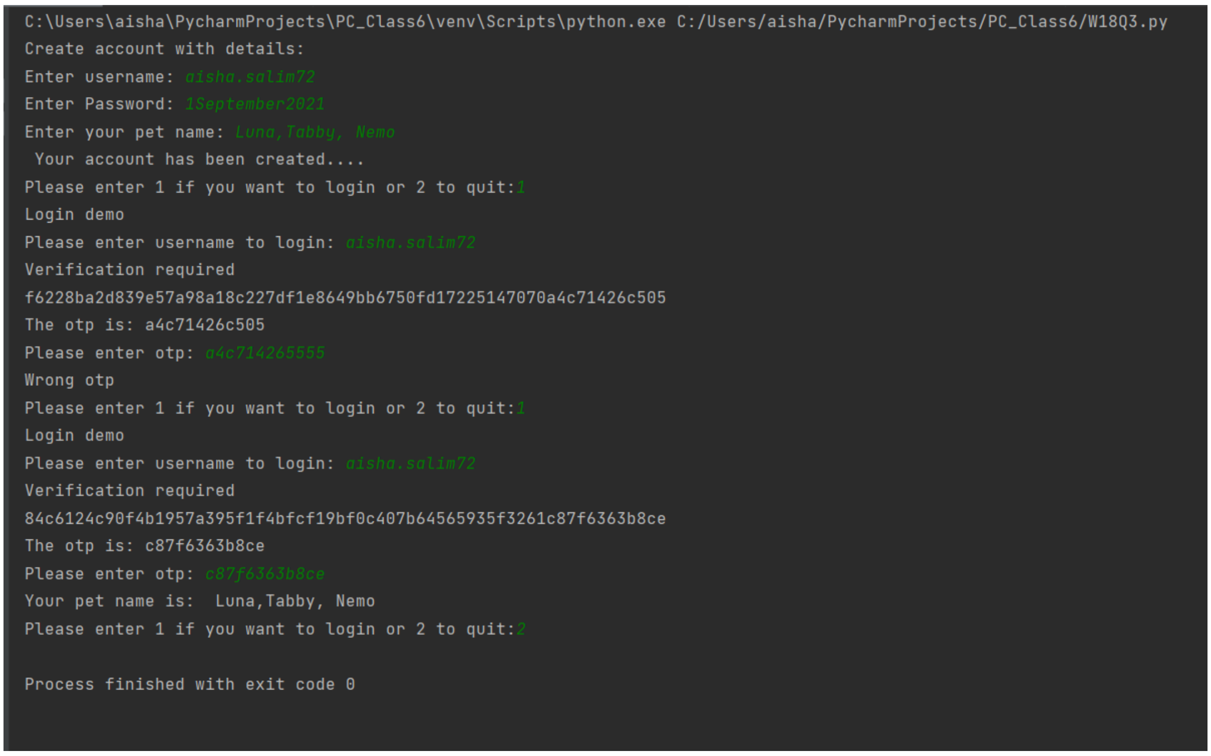
I have created a simple program which takes a file path as input, reads all contents from the file, and generates a SHA-256 hash. When user tries to login and selects a file as a password, program generates and compare two hashes to check if the same file has been uploaded by the user or not.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W18Q2.py
Create account with details:
Enter username: aisha.salim72
Enter Password file path: D:\AishaSaleemCV.pdf
Enter your pet name: Luna,Nemo,Tabby
609dcc0f2c8accdb9fdacee3f6af44a29484fffd47c9f155bf105dae0a6c4000
 Your account has been created....
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim72
Please enter password to login: D:\Immigration letter.docx
c2594ce1a481a53e6a4779d5a599c515fc2dd46b4c1ede79d7c87ad1d995e949
Wrong credentials
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim72
Please enter password to login: D:\AishaSaleemCV.pdf
609dcc0f2c8accdb9fdacee3f6af44a29484fffd47c9f155bf105dae0a6c4000
Your pet name is:  Luna,Nemo,Tabby
Please enter 1 if you want to login or 2 to quit:2

Process finished with exit code 0
```

Exercise 3

I have not implemented SMS service for this exercise as it was out of scope of this assessment. I am displaying the OTP on console for the simplicity and have demonstrated both cases where user entered a correct OTP (successful case) and where user entered an incorrect OTP (unsuccessful case).



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W18Q3.py
Create account with details:
Enter username: aisha.salim72
Enter Password: 1September2021
Enter your pet name: Luna,Tabby, Nemo
 Your account has been created....
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim72
Verification required
f6228ba2d839e57a98a18c227df1e8649bb6750fd17225147070a4c71426c505
The otp is: a4c71426c505
Please enter otp: a4c714265555
Wrong otp
Please enter 1 if you want to login or 2 to quit:1
Login demo
Please enter username to login: aisha.salim72
Verification required
84c6124c90f4b1957a395f1f4bfcf19bf0c407b64565935f3261c87f6363b8ce
The otp is: c87f6363b8ce
Please enter otp: c87f6363b8ce
Your pet name is:  Luna,Tabby, Nemo
Please enter 1 if you want to login or 2 to quit:2

Process finished with exit code 0
```

Additional Question:

1. This scheme is not secure because there is no random factor ; if user gets access to hashed password once; he can easily impersonate the user without any need to break into the server again as he can apply brute force to append different date time formats and hashing algorithms with different rounds to successfully generate an OTP.

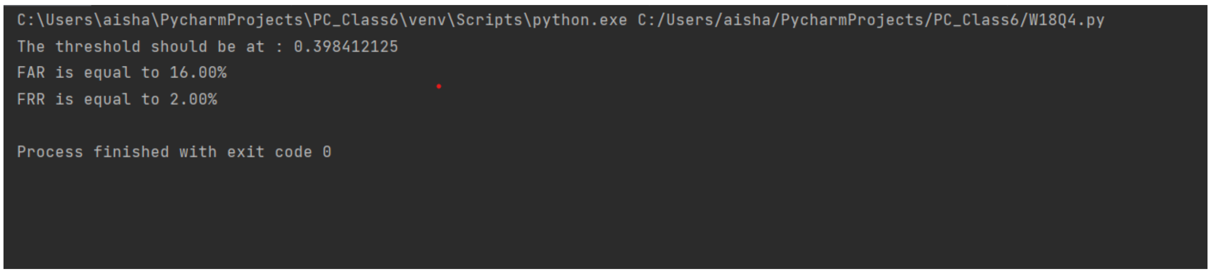
2. To improve the security of OTP scheme , we can use a random factor appended instead of date time as date time is easier to reproduce but a random number with a long length will be difficult to reproduce for the hacker. And we need to also ensure that if a user enters wrong OTP certain number of times, his account should get locked to suspect an attack.
3. For the current scenario, password at the time of registration seems useless but we can use password to increase the security of OTP scheme and use OTP as a second factor of authentication while entering correct password at the time of login still being the first authenticator. In this way first user has to authenticate by entering correct password and then enter correct OTP to login.

Exercise 4

The challenge is to find a threshold value after which a person should be rejected to get authentication, be it the legitimate person. Because biometric system requires collecting live template and then comparison with the sample saved in database acquired at registration, there is always a chance to get error. That is why we need to have a threshold value which decides at the time of error if user should be authenticated or not.

I designed an algorithm to find the threshold value at which FRR and FAR are at minimum. The algorithm generates score by summing the value of FRR and FAR after each iteration. It first calculates the averages of values given in the question for the Authentic User Alice (Value 1) and Intruder Eve (Value 2) and then gets an average of the two (Value 3). Then algorithm generates scores for all three values and continues rejecting the one value for which score is the highest. It stops when score gets constant. It then calculates the average of the two values for which score remained constant and decides the accepted threshold from that. The algorithm is inspired by dichotomy algorithm which keeps re-calculating until value gets constant.

The **threshold value found is nearest to 0.4** which is also given in the recommended set of thresholds. At this threshold, False Acceptance Rate (FAR) is 16 % and False Rejection Rate is 2 %.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W18Q4.py
The threshold should be at : 0.398412125
FAR is equal to 16.00%
FRR is equal to 2.00%

Process finished with exit code 0
```

Week 19

1. Introduction

This week covers exercises which are beyond user authentication, it covers the following exercises:

- Message Authentication with HMAC (Hash-based message authentication code)
- Needham Schroeder Protocol
- Demonstrating an attack on Needham Schroeder Protocol

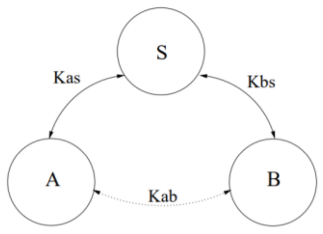
2. Learning Outcomes

- **HMAC (Hash-based Message Authentication Code)**

HMAC is a cryptographic technique used to send message over unsecure channel with integrity. In this server and client agrees on a private key only known to server and client only. The client generates a unique HMAC/ hash per request to the server by hashing the request data with the private key and send it as a part of the request. The server when receives the message, it recalculates the HMAC using the client specific key and matches it with the HMAC received as a part of the request. If both HMACs matches, it shows the integrity of the message that it has securely reached the server without any adversary manipulating the message.

- **Needham Schroeder Protocol**

Needham Schroeder Protocol is a protocol designed to generate and share a session key between two parties intended to start communication in a symmetrically encrypted communication.
Two parties A and B needs to communicate through a server S and need to generate and share a session key. It is assumed that A and B already have a secure communication with the server using keys Kas and Kbs. Following figure explains the background conditions:



The protocol shares the steps how Kab is generated by server and successfully gets shared between A and B so that two parties can start secure communication.

- **Issues in Needham Schroeder Protocol / Replay Attack**

It is possible for an intruder to interrupt messages from an earlier session and use the keys to get successful authentication. It is called a Replay Attack where intruder can reuse the old session key and pass it as a new one. In original protocol, there is no way for party B to know if key is new or old.

3. Observations

I understood how nonces are useful and I can use this concept in my other algorithms as well because it ensures if A generates a random number(nonce) and send it to B and B then sends it back to A. A can be sure that the message is sent by B because only B can send it back the random number that A sent it.

Similarly, timestamp can also be used as a nonce which can improve this concept to identify replay attacks. Or both timestamps and nonce can be used together to increase the security more.

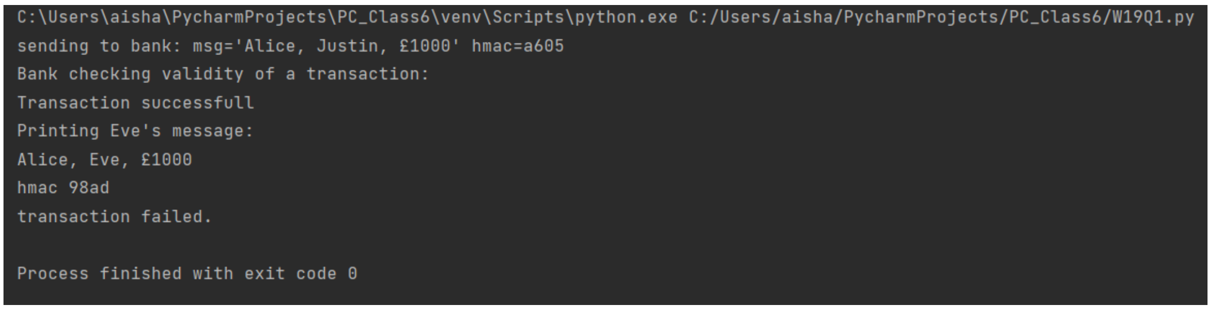
I have also realized how hard it is to make a protocol secure as for N-S protocol , it is important that both parties have secured connection with the server.

4. Proof of Work

Exercise 1

I have demonstrated that bank and Alice are communicating on an unsecure channel. Bank and Alice have already agreed on a key, now Alice is sending a message to bank along with HMAC calculated based on the key, her transaction is successful. Later on, Eve (intruder) intercepts the transaction and changes the message. Now when bank checks the validity of transaction and recalculates the HMAC, it does not match with the HMAC sent with the message.

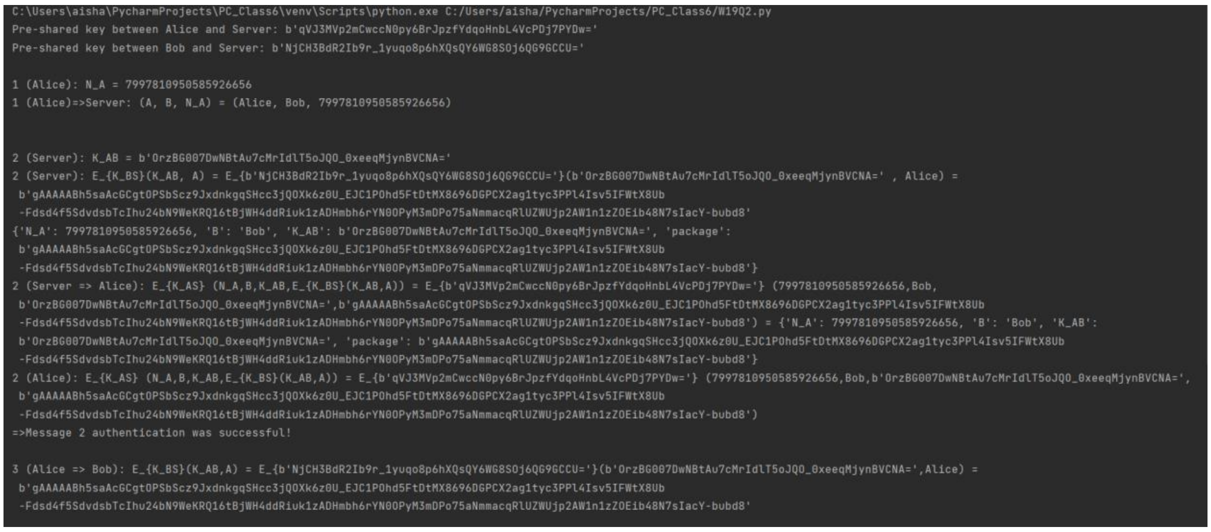
1. Hence, bank server rejects the transaction request successfully inspecting it to be coming from an intruder.
2. 16^4(16 to the power of 4) possibilities for 4 digits of HMAC , each digit is hexadecimal digit, hence EVE needs to try 16^4 possibilities through a brute force to get a successful HMAC. The greater the length of HMAC, more secure it is because it will need more resources and time to crack it which makes it more secure.



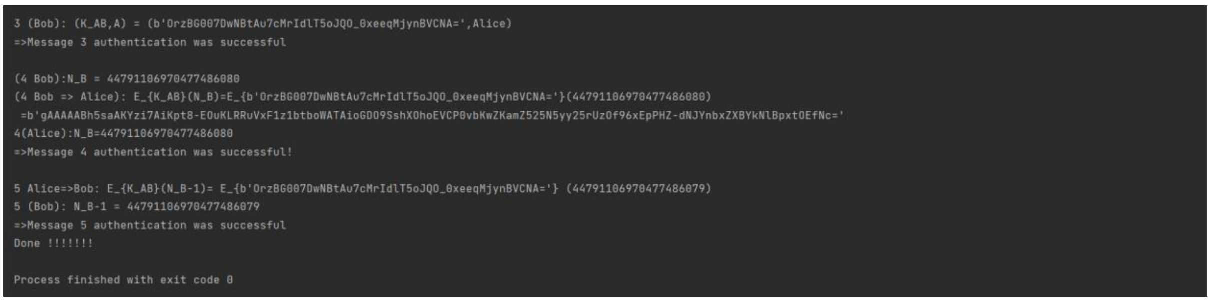
```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W19Q1.py
sending to bank: msg='Alice, Justin, £1000' hmac=a605
Bank checking validity of a transaction:
Transaction successfull
Printing Eve's message:
Alice, Eve, £1000
hmac 98ad
transaction failed.

Process finished with exit code 0
```

Exercise 2

In this I have demonstrated Needham Schroeder Protocol step by step with message authentication at each step. If message is not authenticated at any step, protocol will stop.



```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W19Q2.py
Pre-shared key between Alice and Server: b'qVJ3MVp2mCwccN0py6BrJpzfYdqoHnbL4VcPDj7PYDw='
Pre-shared key between Bob and Server: b'NjCH3BdR2Ib9r_1yuqo8p6hXQsQY6WG8SOj6QG9GCCU='

1 (Alice): N_A = 7997810950585926656
1 (Alice)=>Server: (A, B, N_A) = (Alice, Bob, 7997810950585926656)


2 (Server): K_AB = b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA='
2 (Server): E_{K_BS}(K_AB, A) = E_{b'NjCH3BdR2Ib9r_1yuqo8p6hXQsQY6WG8SOj6QG9GCCU='}(b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=' , Alice) =
 b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8'
{'N_A': 7997810950585926656, 'B': 'Bob', 'K_AB': b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=', 'package':
 b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8'}
2 (Server => Alice): E_{K_AS} (N_A,B,K_AB,E_{K_BS}(K_AB,A)) = E_{b'qVJ3MVp2mCwccN0py6BrJpzfYdqoHnbL4VcPDj7PYDw='} (7997810950585926656,Bob,
 b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=',b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8') = {'N_A': 7997810950585926656, 'B': 'Bob', 'K_AB':
 b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=', 'package': b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8'}
2 (Alice): E_{K_AS} (N_A,B,K_AB,E_{K_BS}(K_AB,A)) = E_{b'qVJ3MVp2mCwccN0py6BrJpzfYdqoHnbL4VcPDj7PYDw='} (7997810950585926656,Bob,b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=',
 b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8')
=>Message 2 authentication was successful!

3 (Alice => Bob): E_{K_BS}(K_AB,A) = E_{b'NjCH3BdR2Ib9r_1yuqo8p6hXQsQY6WG8SOj6QG9GCCU='}(b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=',Alice) =
 b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8'
```

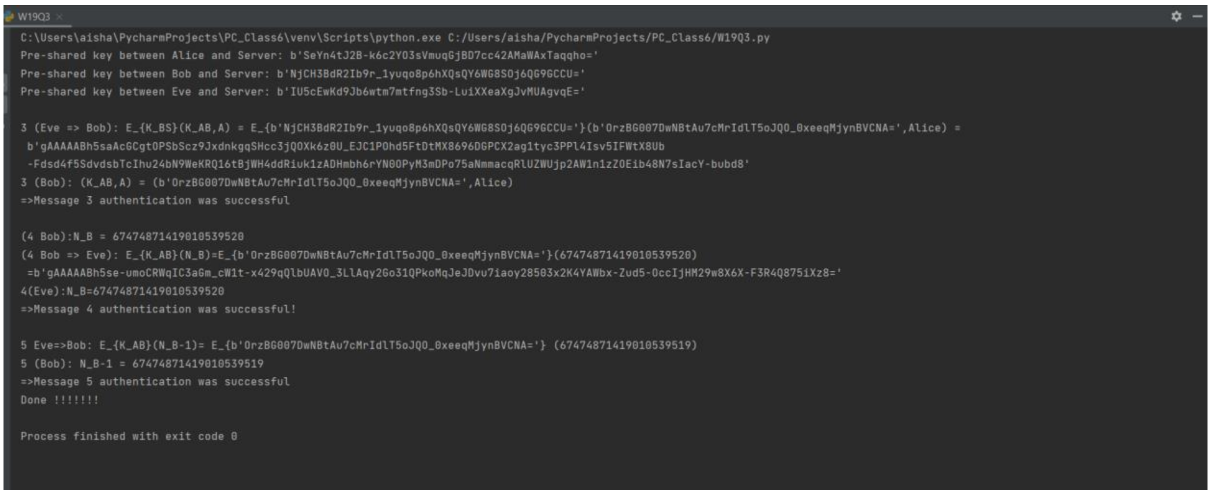


```
3 (Bob): (K_AB,A) = (b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=',Alice)
=>Message 3 authentication was successful

(4 Bob):N_B = 4479110697047748608O
(4 Bob => Alice): E_{K_AB}(N_B)=E_{b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA='}(44791106970477486080)
 =b'gAAAAABh5saAKYzi7AiKpt8-EOuKLRRuVxF1z1btboWATAioGDO9SshXOhoEVCP0vbKwZKamZ525N5yy25rUzOf96xEpPHZ-dNJYnbxZXBYkNlBpxtOEfNc='
4(Alice):N_B=44791106970477486080
=>Message 4 authentication was successful!

5 Alice=>Bob: E_{K_AB}(N_B-1)= E_{b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA='} (44791106970477486079)
5 (Bob): N_B-1 = 44791106970477486079
=>Message 5 authentication was successful
Done !!!!!!!

Process finished with exit code 0
```

Exercise 3

In this I am demonstrating REPLAY attack.

I have reused KAB key, message 3 that Alice sends to BOB and session Key between Bob and Server from previous Exercise to demonstrate how EVE can use this information to plan a successful replay attack:



W19Q3

```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W19Q3.py
Pre-shared key between Alice and Server: b'SeYn4tJ2B-k6c2YO3sVmuqGjBD7cc42AMaWAxTaqqho='
Pre-shared key between Bob and Server: b'NjCH3BdR2Ib9r_1yuqo8p6hXQsQY6WG8SOj6QG9GCCU='
Pre-shared key between Eve and Server: b'IU5cEwKd9Jb6wtm7mtfng3Sb-LuiXXeaXgJvMUAgvqE='

3 (Eve => Bob): E_{K_BS}(K_AB,A) = E_{b'NjCH3BdR2Ib9r_1yuqo8p6hXQsQY6WG8SOj6QG9GCCU='}(b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=',Alice) =
 b'gAAAAABh5saAcGCgtOPSbScz9JxdnkgqSHcc3jQOXk6z0U_EJC1POhd5FtDtMX8696DGPCX2ag1tyc3PPl4Isv5IFWtX8Ub
 -Fdsd4f5SdvdsbTcIhu24bN9WeKRQ16tBjWH4ddRiuk1zADHmbh6rYN0OPyM3mDPo75aNmmacqRlUZWUjp2AW1n1zZOEib48N7sIacY-bubd8'
3 (Bob): (K_AB,A) = (b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA=',Alice)
=>Message 3 authentication was successful

(4 Bob):N_B = 67474871419010539520
(4 Bob => Eve): E_{K_AB}(N_B)=E_{b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA='}(67474871419010539520)
 =b'gAAAAABh5se-umoCRWqIC3aGm_cW1t-x429qQlbUAVO_3LlAqy2Go31QPkoMqJeJDvu7iaoy28503x2K4YAWbx-Zud5-OccIjHM29w8X6X-F3R4Q8751Xz8='
4(Eve):N_B=67474871419010539520
=>Message 4 authentication was successful!

5 Eve=>Bob: E_{K_AB}(N_B-1)= E_{b'OrzBG007DwNBtAu7cMrIdlT5oJQO_0xeeqMjynBVCNA='} (67474871419010539519)
5 (Bob): N_B-1 = 67474871419010539519
=>Message 5 authentication was successful
Done !!!!!!!

Process finished with exit code 0
```

Eve successfully passes Message 3, Message 4 and Message 5 authentication showing how replay attack is possible!

Week 20

1. Introduction

This week discusses the following exercises :

- Setting Linux file and folder permissions
- Investigating Role-based Access Control (RBAC)
- Cross-site Scripting (XSS) Attacks
- Sandbox
- Federated Identity Management System (FIM)

2. Learning Outcomes

- **Role-based Access Control (RBAC)**

Role-based access control (RBAC) determines the resources that a person can access within an organization based on person's role within the organization.

- **Federated Identity Management System (FIM)**

FIM system is like single sign on where the user is requested to sign in only once and then it can be used to login all the apps not only within the organization but also the apps of partnered organization or third-party apps without the need to login repeatedly.

- **Same Origin Policy and Cross Site Scripting Attacks**

Same Origin Policy is a policy enforced by web browsers which forbids websites to load ,read or write contents from pages from different origin.

XSS attacks are injection attacks where malicious scripts are injected into a website. The malicious script is executed inside a victim's browser through which attacker can comprise victim's browser.

3. Observations

In this week I learned concepts which are beyond user authentication but necessary to understand other security aspects that need into consideration. When designing a system, we also need to consider other aspects like Single Origin Policy and Cross Scripting attack and giving correct permissions to different roles and consider if we need Single Sign On policy or not as these things also impact the security of our system and is beyond user authentication.

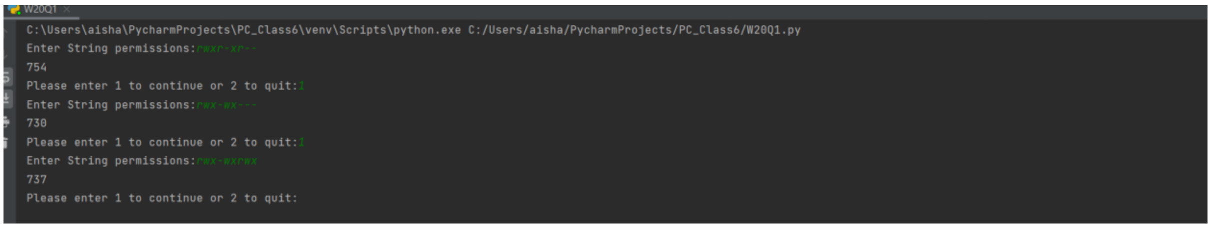
4. Challenges

- I was not able to complete the XSS game because the hints on the page are not clear and I was not able to follow it after level 3.

- Technical documents for FIM systems were not easily available.

5. Proof of work

Exercise 1

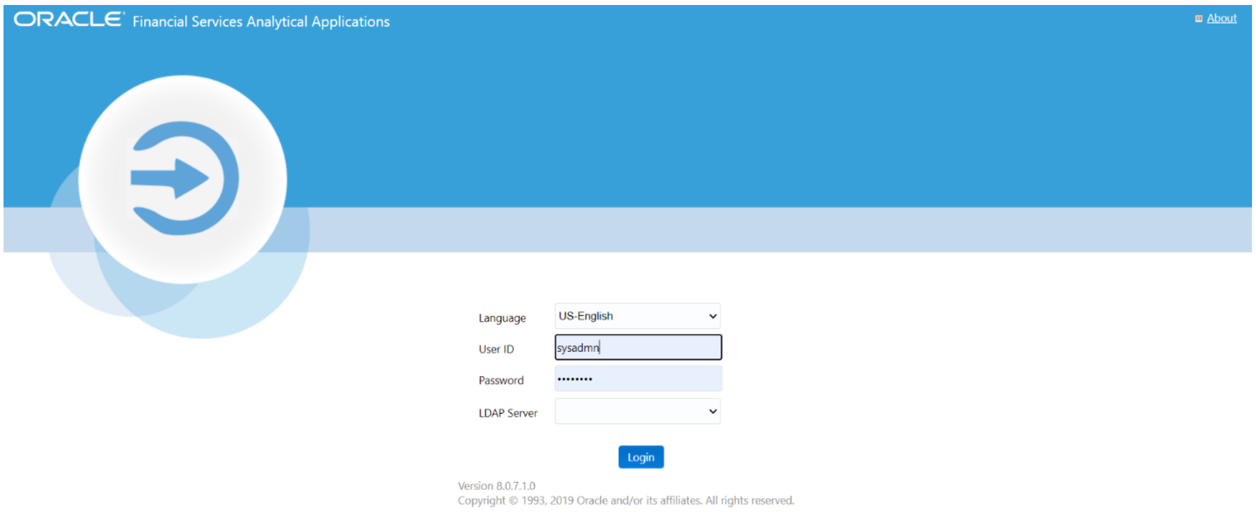


W20Q1

```
C:\Users\aisha\PycharmProjects\PC_Class6\venv\Scripts\python.exe C:/Users/aisha/PycharmProjects/PC_Class6/W20Q1.py
Enter String permissions:rwxr-xr--
754
Please enter 1 to continue or 2 to quit:1
Enter String permissions:rwx-wx---
730
Please enter 1 to continue or 2 to quit:1
Enter String permissions:rwx-wxrwx
737
Please enter 1 to continue or 2 to quit:
```

Exercise 2

- I have investigated a web-based application called Oracle FCCM (Financial Crime and Compliance Management) suite which uses RBAC.



ORACLE Financial Services Analytical Applications

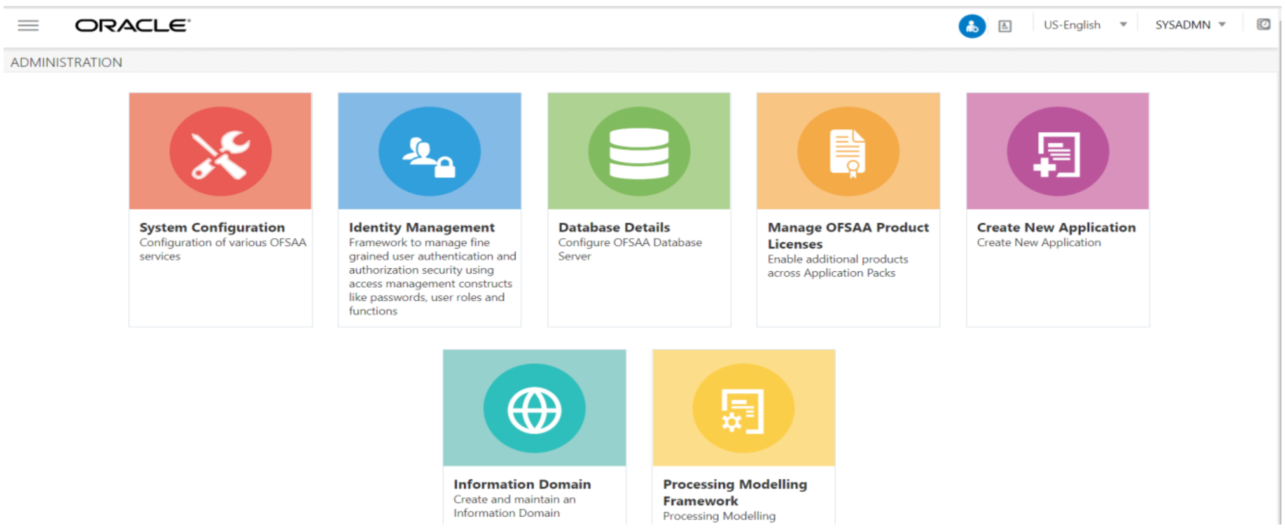
About

Language	US-English
User ID	sysadmn
Password	••••••••
LDAP Server	

Login

Version 8.0.7.1.0
Copyright © 1993, 2019 Oracle and/or its affiliates. All rights reserved.

- It uses an Identity management module through which users and user groups can be created, deleted, and modified.



System Configuration Configuration of various OFSAA services

Identity Management Framework to manage fine grained user authentication and authorization security using access management constructs like passwords, user roles and functions

Database Details Configure OFSAA Database Server

Manage OFSAA Product Licenses Enable additional products across Application Packs

Create New Application Create New Application

Information Domain Create and maintain an Information Domain

Processing Modelling Framework Processing Modelling

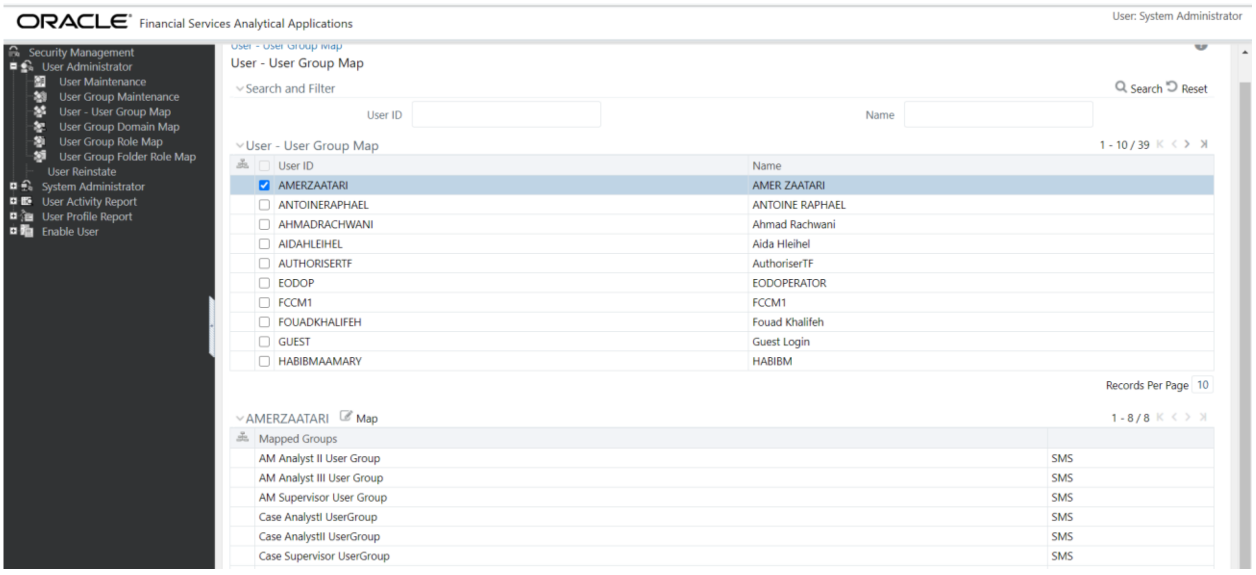
- This is the group maintenance page through which roles can be modified. This application has quite a lot of user groups as it is a type of CRM. So just going through a few of them.

TFLTANALYSTGRP
KYC Investigator User Group
Case Supervisor UserGroup
Case AnalystII UserGroup
Case AnalystI UserGroup
AM Supervisor User Group
AM Analyst III User Group
AM Analyst II User Group
AM Analyst I User Group
AM Data Miner User Group
AM Executive User Group
AM External Auditor User Group
AM Internal Auditor User Group
CM FrontOffice Anlyst GRP
CM Level 1 Analyst GRP
CM Level 1 Supervisor GRP
CS Admin
Case Administrator User Group
Case Executive UserGroup
Case External Auditor UserGroup
Case Initiator User Group
Case Internal Auditor Usergroup
Case Viewer UserGroup
IPEADMN

KYC Administrator User Group
Mantas Administrator User Group
OB KYC Administrator Group
OB KYC Investigator Group
QUESTMAINT
Qtnr Confidential Grp
Questionnaire Admin Grp
Questionnaire Lcalised Admin Grp
Questionnaire Localised Auth Grp
Questionnaire Localised View Grp
Questionnaire Signoff Grp
Questionnaire Template Grp
Questionnaire Template View Grp
Real Time Screening Access Group
TFLTADMINISTATORGRP
TFLTSUPERVISORGRP
WLSUPERVISORUG

1. TFLTANALYSTGRP group can access the real time application module which is used for real time transaction screening.
2. WLSUPERVISORUG gives access to the Watchlist modification control through which blacklisted/sanctioned personal/country names can be edited.
3. AM Analyst I User Group can access the Anti-Money Laundering module of the suite through which the analyst can monitor the suspicious transaction behavior.

- User Groups assigned to one of the users. Now this user will only have access to the modules or accesses these groups allow.



User - User Group Map

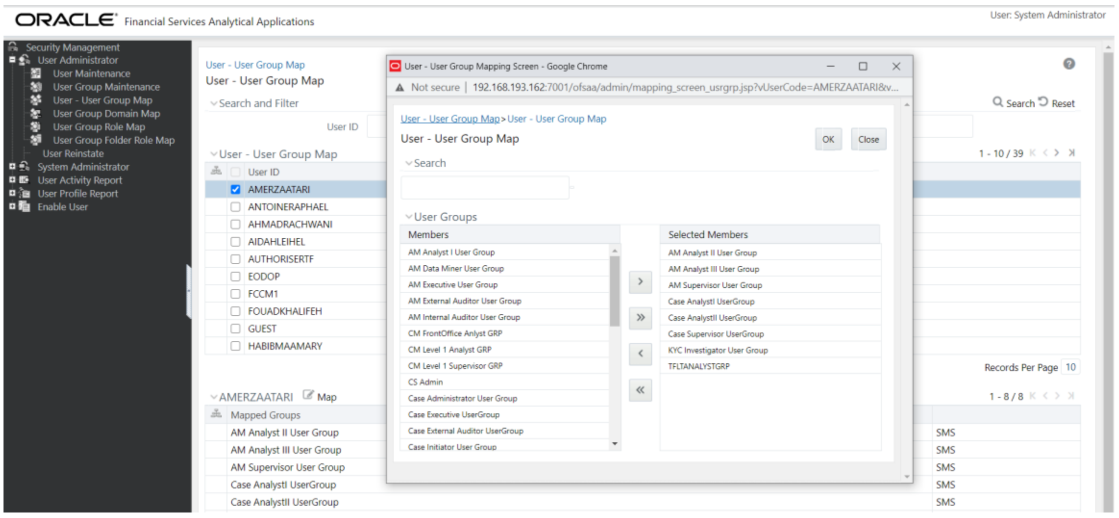
1 - 10 / 39

User ID	Name
✓ AMERZAATARI	AMER ZAATARI
ANTOINERAPHAEL	ANTOINE RAPHAEL
AHMADRACHWANI	Ahmad Rachwani
AIDAHLEIHEL	Aida Hleihel
AUTHORISERTF	AuthoriserTF
EODOP	EODOPERATOR
FCCM1	FCCM1
FOUADKHALIFEH	Fouad Khalifeh
GUEST	Guest Login
HABIBMAAMARY	HABIBM

1 - 8 / 8

Mapped Groups	
AM Analyst II User Group	SMS
AM Analyst III User Group	SMS
AM Supervisor User Group	SMS
Case AnalystI UserGroup	SMS
Case AnalystII UserGroup	SMS
Case Supervisor UserGroup	SMS

- Following is the page through which user group accesses can be modified.



User - User Group Map
User - User Group Map

Search and Filter

User ID

User - User Group Map

User ID
☑ AMERZAATARI
☐ ANTOINERAPHAEL
☐ AHMADRACHWANI
☐ AIDAHLEIHEL
☐ AUTHORISERTF
☐ EODOP
☐ FCCM1
☐ FOUADKHALIFEH
☐ GUEST
☐ HABIBMAAMARY

1 - 10 / 39

Records Per Page 10

AMERZAATARI Map

1 - 8 / 8

Mapped Groups	
AM Analyst II User Group	SMS
AM Analyst III User Group	SMS
AM Supervisor User Group	SMS
Case AnalystI UserGroup	SMS
Case AnalystII UserGroup	SMS

User - User Group Mapping Screen - Google Chrome

Not secure | 192.168.193.162:7001/ofsaa/admin/mapping_screen_usrgrp.jsp?vUserCode=AMERZAATARI&v...

User - User Group Map > User - User Group Map

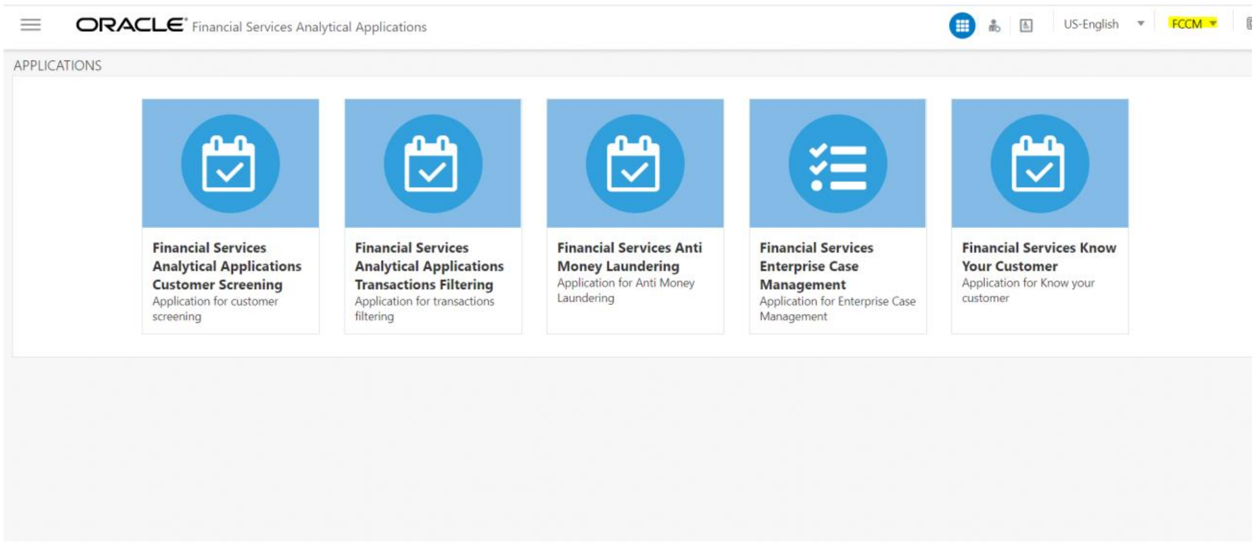
User - User Group Map

Search

User Groups

Members	Selected Members
AM Analyst I User Group	AM Analyst II User Group
AM Data Miner User Group	AM Analyst III User Group
AM Executive User Group	AM Supervisor User Group
AM External Auditor User Group	Case AnalystI UserGroup
AM Internal Auditor User Group	Case AnalystII UserGroup
CM FrontOffice Anlyst GRP	Case Supervisor UserGroup
CM Level 1 Analyst GRP	KYC Investigator User Group
CM Level 1 Supervisor GRP	TFLTANALYSTGRP
CS Admin	
Case Administrator User Group	
Case Executive UserGroup	
Case External Auditor UserGroup	
Case Initiator User Group	

- This user FCCM can access 5 applications from this suite as it has access to these user groups.



Financial Services Analytical Applications Customer Screening
Application for customer screening

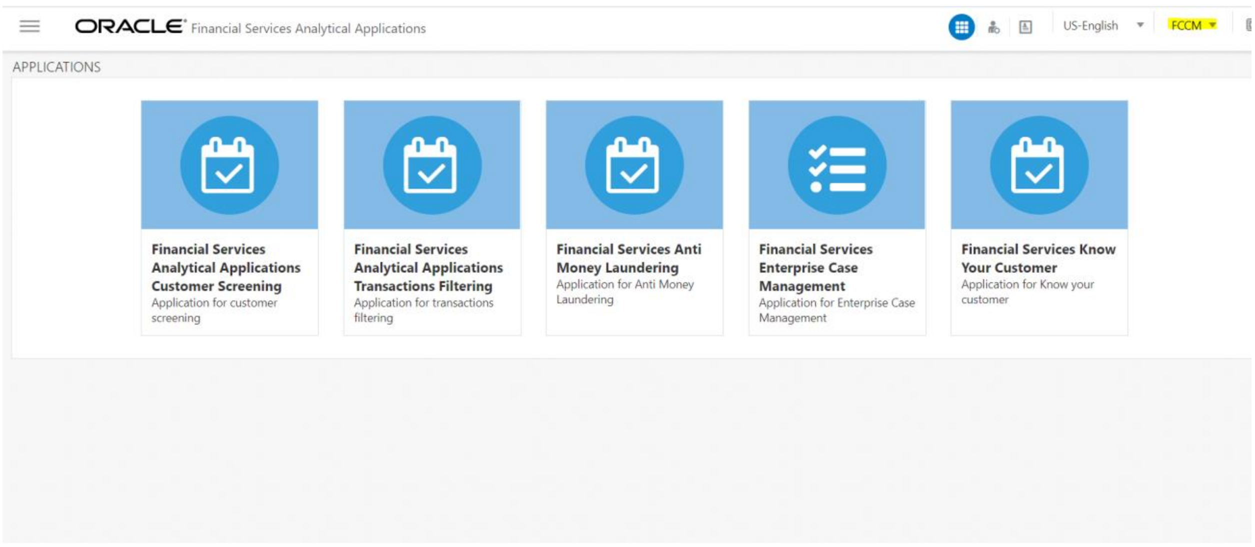
Financial Services Analytical Applications Transactions Filtering
Application for transactions filtering

Financial Services Anti Money Laundering
Application for Anti Money Laundering

Financial Services Enterprise Case Management
Application for Enterprise Case Management

Financial Services Know Your Customer
Application for Know your customer

- However, the following user FCCMTEST1 can access only 4 of these applications due to limited accessibility.


ORACLE Financial Services Analytical Applications
US-English
FCCM
APPLICATIONS
Financial Services Analytical Applications Customer Screening
Application for customer screening
Financial Services Analytical Applications Transactions Filtering
Application for transactions filtering
Financial Services Anti Money Laundering
Application for Anti Money Laundering
Financial Services Enterprise Case Management
Application for Enterprise Case Management
Financial Services Know Your Customer
Application for Know your customer

Exercise 3

Same Origin Policy is a policy enforced by web browsers which forbids websites to load ,read or write contents from pages from different origin.

Origin is a combination of three parts:

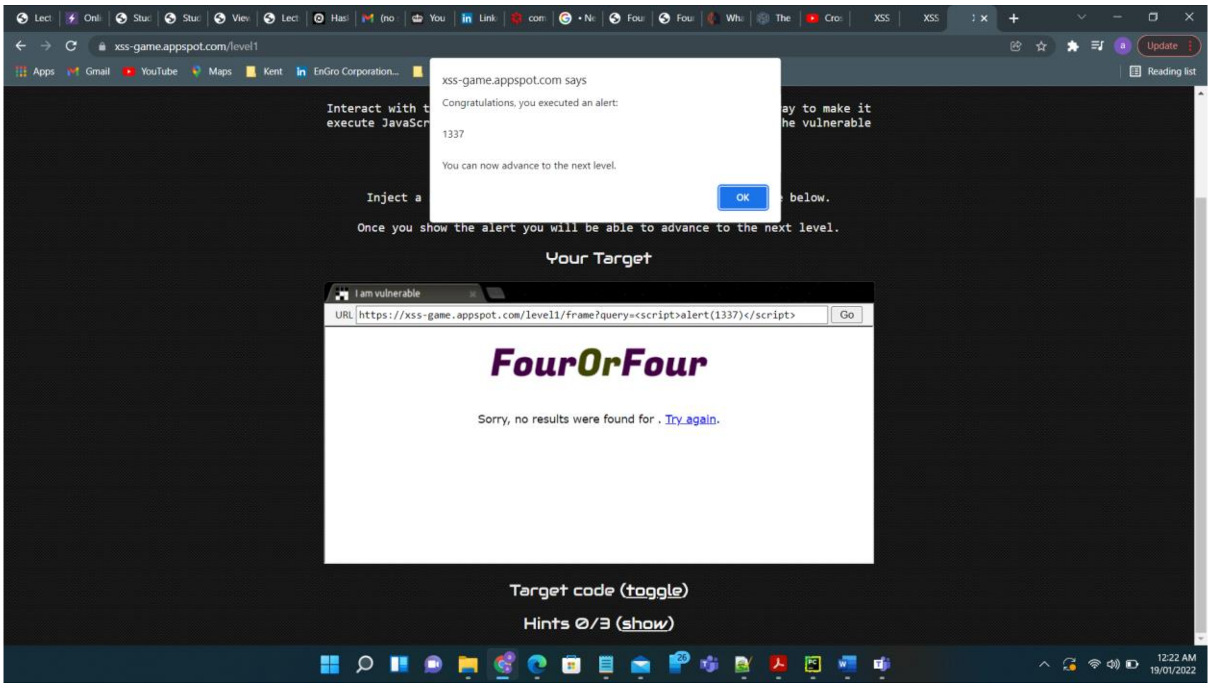
1. Schema
2. Hostname (Domain /subdomain)
3. Port number

So, all the resources having same above three properties are identified from the same origin and the browser only allows reading or writing between the two.
What if you can get control over Java Script of a website?
JavaScript has access to HTML elements of a website such as DOM elements and it can be used to access different DOM elements such as reading cookies. It can be very problematic.

XSS is an attack in which attacker is able to inject Java Script into a web page.

Cross-Site Scripting attack bypasses the Same Origin Policy or SOP. When user input in taken and it is not properly checked, an attacker can take advantage of the situation and insert any JavaScript script which can be used to get access to content which is forbidden for the attacker.
Following are different types of XSS Attacks:

1. **Reflected XSS** where input is reflected back in response and identified as script block and gets executed.



xss-game.appspot.com says

Congratulations, you executed an alert:

1337

You can now advance to the next level.

Interact with t… ay to make it
execute JavaScr… he vulnerable

Inject a … below.

Once you show the alert you will be able to advance to the next level.

Your Target

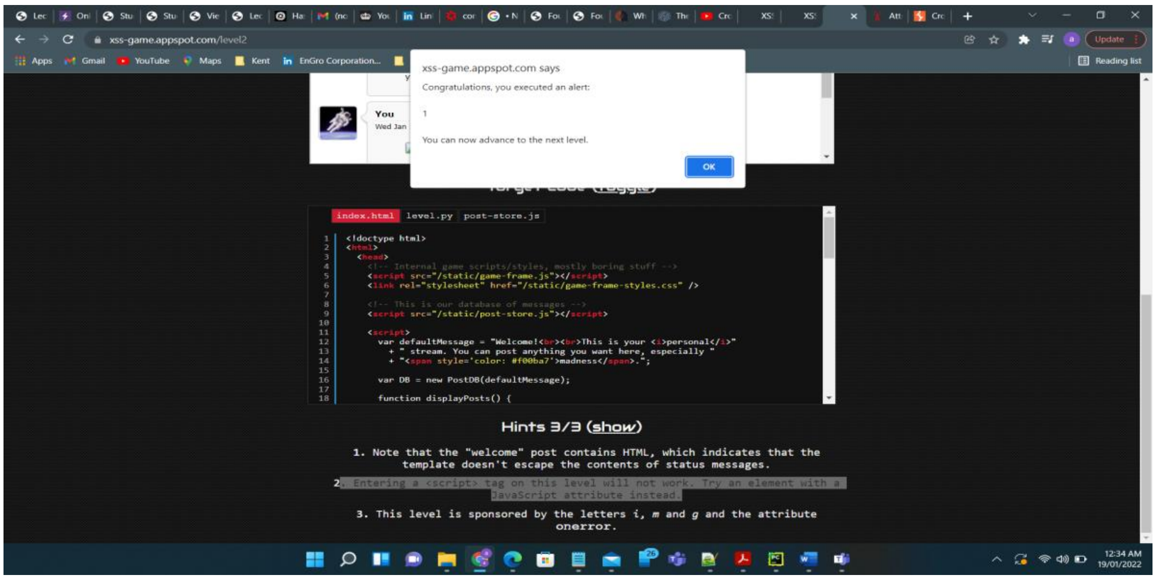
URL https://xss-game.appspot.com/level1/frame?query=<script>alert(1337)</script>

FourOrFour

Sorry, no results were found for . Try again.

Target code (toggle)

Hints 0/3 (show)



xss-game.appspot.com says

Congratulations, you executed an alert:

1

You can now advance to the next level.

index.html level.py post-store.js

```
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />

    <!-- This is our database of messages -->
    <script src="/static/post-store.js"></script>

    <script>
      var defaultMessage = "Welcome!<br><br>This is your <i>personal</i>"
        + " stream. You can post anything you want here, especially "
        + "<span style='color: #f00ba7'>madness</span>.";

      var DB = new PostDB(defaultMessage);

      function displayPosts() {
```

Hints 3/3 (show)

1. Note that the "welcome" post contains HTML, which indicates that the template doesn't escape the contents of status messages.
2. Entering a <script> tag on this level will not work. Try an element with a JavaScript attribute instead.
3. This level is sponsored by the letters *i*, *m* and *g* and the attribute onerror.

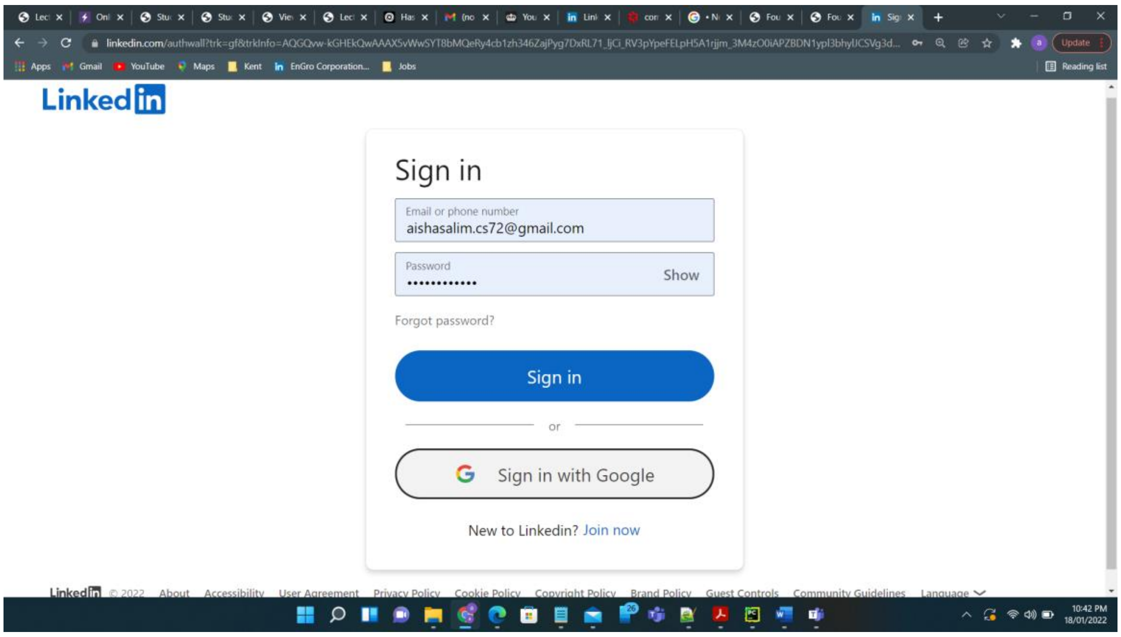
2. Stored XSS in which script is persisted or saved in database and injects everyone who access it.

Exercise 5

FIM system is like single sign on where the user is requested to sign in only once and then it can be used to login all the apps not only within the organization but also the apps of partnered organization or third-party apps without the need to login repeatedly. However, the identity of the user remains confidential. The organizations have some standards used for authentication like OAuth, SAML2 etc.

As a real-life example Google can act as an identity provider for LinkedIn. If the user is logged in to google account LinkedIn can be accessed without the need of different sign in account or entering the information again. The process requires authentication which is done using Google OAuth 2.0 and as an Identity provider google shares some attribute to authenticate the user and provide access to LinkedIn without sharing the actual identity details.

The technique used behind LinkedIn FIM is Google Single Sign-On (SSO) where Google serves as Identity Provider (IdP).

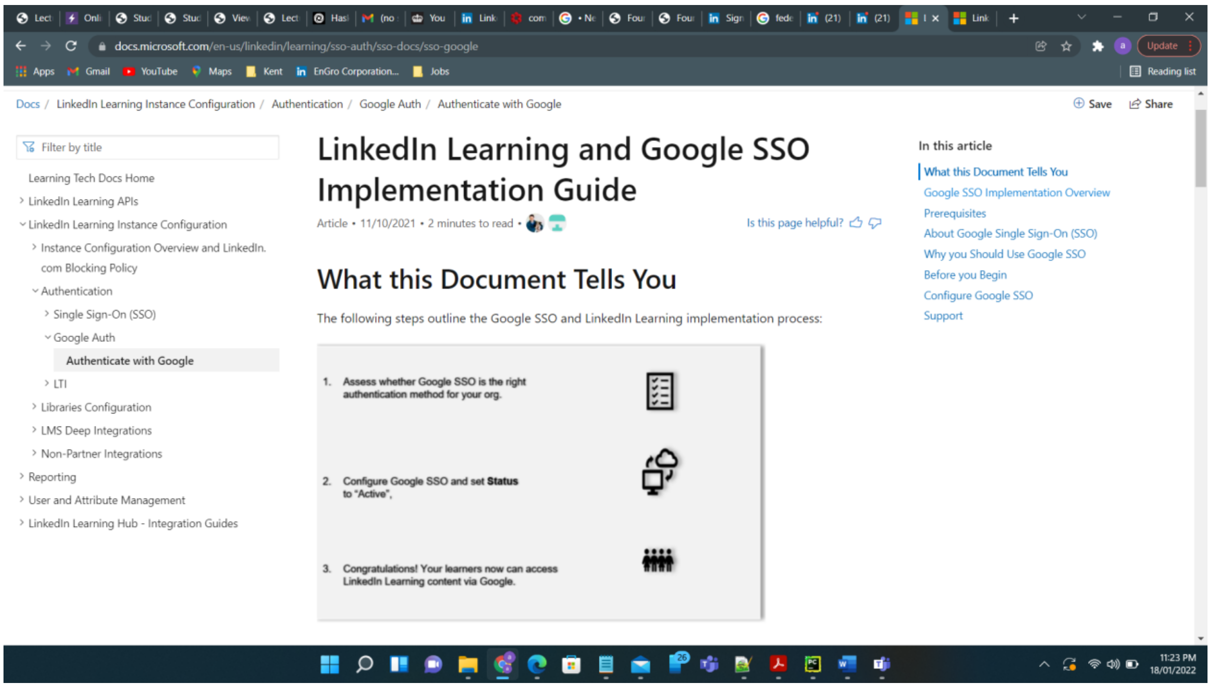


linkedin.com/authwall?trk=gf&trkInfo=AQGQvw-kGHEkQwAAAXSvWwSYT8bMQeRy4cb1zh346ZajPyg7DxRL71_ljCi_RV3pYpeFELpH5A1rjjm_3M4zO0iAPZBDN1ypl3bhylJCSVg3d...
Linked in
Sign in
Sign in
or
Sign in with Google
New to Linkedin? Join now
Linked in © 2022 About Accessibility User Agreement Privacy Policy Cookie Policy Copyright Policy Brand Policy Guest Controls Community Guidelines Language
10:42 PM
18/01/2022

Email or phone number
aishasalim.cs72@gmail.com

Password
•••••••••••• Show

Forgot password?



docs.microsoft.com/en-us/linkedin/learning/sso-auth/sso-docs/sso-google
Docs / LinkedIn Learning Instance Configuration / Authentication / Google Auth / Authenticate with Google
Save
Share
Filter by title
Learning Tech Docs Home
LinkedIn Learning APIs
LinkedIn Learning Instance Configuration
Instance Configuration Overview and LinkedIn.com Blocking Policy
Authentication
Single Sign-On (SSO)
Google Auth
Authenticate with Google
LTI
Libraries Configuration
LMS Deep Integrations
Non-Partner Integrations
Reporting
User and Attribute Management
LinkedIn Learning Hub - Integration Guides
In this article
What this Document Tells You
Google SSO Implementation Overview
Prerequisites
About Google Single Sign-On (SSO)
Why you Should Use Google SSO
Before you Begin
Configure Google SSO
Support
11:23 PM
18/01/2022

LinkedIn Learning and Google SSO Implementation Guide

Article • 11/10/2021 • 2 minutes to read •

Is this page helpful?

What this Document Tells You

The following steps outline the Google SSO and LinkedIn Learning implementation process:

1. Assess whether Google SSO is the right authentication method for your org.
2. Configure Google SSO and set **Status** to "Active".
3. Congratulations! Your learners now can access LinkedIn Learning content via Google.

Account Centre Employee Database Integration:

Employee Database Integration (EDI) allows Google to integrate its users data into LinkedIn Applications. This data can now be used to authenticate google users while signing into LinkedIn. Google has also configured SSO (Single Sign in). This provides an additional layer of security. The integration then allows:

- Uploading all Google's user data into LinkedIn system
- Configuring Single Sign-On

Information Source: LinkedIn Official Documentation:
https://help.linkedin.com/cc/custom_fattach/get/8684157/0/filename/Privacy%20and%20Security%20Whitepaper%20-%20Account%20Center%20Employee%20Database%20Integration.pdf

Part 2
Week 19 Advanced Exercise

Introduction of Advanced Exercise

I have chosen to attempt advanced exercise of Week 19 i.e. Fixing Needham-Schroeder protocol.

What Advanced exercise did I do?

I have chosen this exercise because I implemented naïve Needham-Schroeder Protocol in basic exercise of week 19 and then I also modified scripts to perform replay attack to find vulnerability in this protocol. Performing these basic exercises developed my interest to explore this protocol in detail and to study how fixes have been suggested to overcome these vulnerabilities.

Apart from this, studying this protocol is of keen importance for me as being a programmer I often have to design algorithms to solve similar problems. It is of interest to me to know how I can make this algorithm secure by reducing replay attacks. The knowledge or the technique that I learn in this exercise can be useful in solving similar challenges in my professional life. This is the reason why I preferred doing this advanced exercise and enhance my knowledge on this topic rather than self-designing any new exercise.

How did I do the exercise?

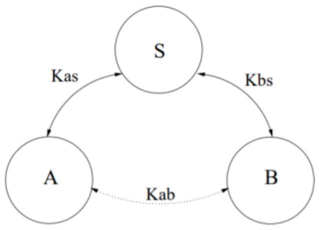
To understand the advanced exercise, it is important to understand how protocol works which then leads to discuss what makes it is vulnerable and how I attempted to fix it.

Following is a brief description of the protocol:

- **What is Needham Schroeder Protocol?**

Needham Schroeder Protocol aims to establish a session key between two parties on an unsecure network to secure the future communication.
It assumes that a centralized key distribution facility called Server is responsible for generation and distribution of all keys and that each user has a registered private key with the server.
Following figure explains the protocol:

- It is assumed that S is a server which is trusted by both parties A and B intending to communicate and both parties already have a secure communication with server S and already have their private keys with the server.
- A has private key Kas with the server S known only between A and S.
- B has private key Kbs with the server S known only between B and S.



The protocol shares the steps how Kab is generated by server and successfully gets shared between A and B so that two parties can start secure communication.

- **Important Terms**

  Nonces: Nonces are random numbers which are assumed with very high probability not been used before in any earlier transactions. Nonces are not Timestamps but nonces are also used to ensure a level of security. If A sends a message to B containing a random number Na and B replies back with the same random number , it ensures that it is B that is replying back as it sends the random number which is only known by either A or B.

  Identity: A and B are identities of the two parties Alice and Bob communicating.
  The protocol as described in the assignment in one of the basic questions is as follows:

  Steps:
  The Needham-Schroeder protocol is:
  1. A → S : A,B, Na

  Alice sends a message to server sending her identity, Bob's identity B and a random number Nonce Na.

  2. S → A : {Na,B,Kab, {Kab,A}Kbs } Kas

  Server generates a session key Kab and sends it back to Alice. It also sends a package with session key Kab and Alice's nonce Na to be sent to B encrypted in Kbs for Alice to forward it to Bob.

  3. A → B : {Kab,A}Kbs

  Alice then forwards that whole package received from Server S to Bob. Bob is able to decrypt it as it is encrypted Kbs is the shared key between Bob and Server.

  4. B → A : {Nb}Kab
  Bob now decrypts the package and has gotten Kab. Bob now generates a random number Nonce Nb and encrypt is using Kab (session key Bob has just received) to show to Alice that he has successfully gotten the session key.

  5. A → B : {Nb − 1}Kab
  Now Alice performs a simple operation on the Nonce Nb and send it back to Bob showing Alice is still available to start communication.

  Here Na and Nb are nonces.

- **Issue in Needham Schroeder Protocol / Replay Attack**

It is possible for an intruder to interrupt messages from an earlier session. An intruder can intercept KAB from an older session and use the key to get successful authentication as Bob. There is no mechanism at Bob to ensure that Kab which is sent to Bob at step 3 is fresh or an old session being sent.It is called a Replay Attack where intruder can reuse the old session key and pass it as a new one.

I demonstrated a replay attack in Week 19 Q3 basic exercise how Eve(intruder) interpreted Kab and then used a pre recorded session between Alice and Bob to get successful authentication and impersonation as Alice.

This is the issue in naïve Needham Schroeder Protocol that needs to be fixed.

- **How do I fix that issue?**

To fix the replay attack, I have added following two fixes:

1. Adding authentication of Alice by including a new Nonce:

Following are the steps how adding a new Nonce can solve this issue. Adding new steps for the protocol:

1. A → B : A

In the beginning of the protocol now Alice is sending a message to Bob containing its identity A.

2. B → A : {A, N'B} Kbs

Bob responds back to Alice with a nonce N'B encrypted with Bob's key with the Server Kbs.

3. A → S : A,B,NA,{A,N'B}Kbs

Alice sends a message to server sending her identity, Bob's identity B and a random number Nonce Na. It also sends the package sent back by B containing the new nonce N'B to the server.

4. S → A : {Na,B,Kab, {Kab, A, N'B }Kbs } Kas

Server generates a session key Kab and sends it back to Alice. It also sends a package with session key Kab and Alice's nonce Na and the nonce N'B sent earlier by Bob to Alice encrypted in Kbs for Alice to forward it to Bob.

5. A → B : {Kab, A, N'B } Kbs

Now when Alice sends this package to Bob, the presence of new nonce N'B ensures that it is actually Alice that is sending message to Bob and not any intruder. The remaining steps are same as the naïve algorithm shared above.

2. Adding timestamps to avoid Replay Attacks

Adding timestamp is a common fix in security protocols which is useful to identify replaying of old session key. I have introduced timestamps at the step 4 and 5 of the protocol. The new protocol is as follows which is a further improvement of the protocol described in above section where New Nonce is introduced:

4. S → A : {Na,B,Kab, {Kab, A, N'B,T }Kbs ,T} Kas

Server generates a session key Kab and sends it back to Alice along with Nonce of A Na, identity of Bob B and a **Timestamp T**. It also sends a package with session key Kab and Alice's nonce Na and the nonce N'B sent earlier by Bob to Alice and a **Timestamp T** encrypted in Kbs for Alice to forward it to Bob.

5. A → B : {Kab, A, N'B,T } Kbs

Where T refers to Timestamp.

Now when Alice receives this message (step 4) from server it and when B receives this message from Alice (step 5), they can compare the timestamps and reject the authentication if timestamp has exceeded the threshold.

Reject if [Current Time - Timestamp T > = Threshold]

What libraries have I used for the Demonstration

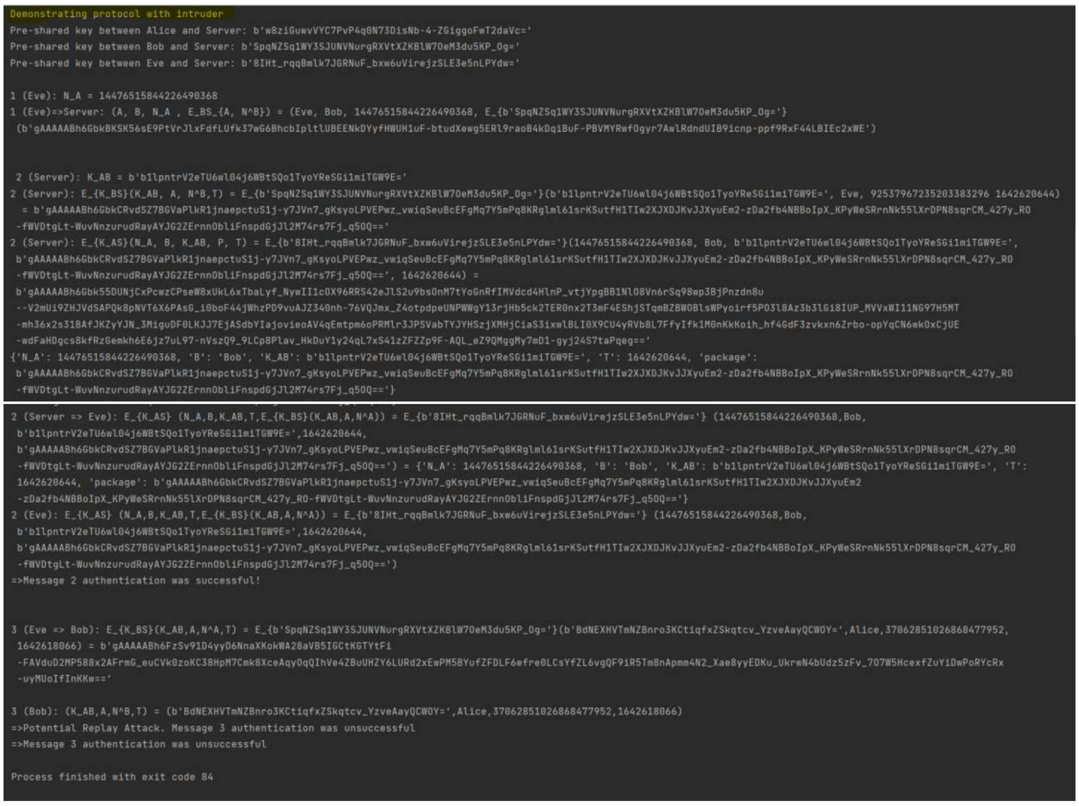
- I have used Python as it is a cross-platform programming language.
- I have created a class for the Server which is responsible for generating and distribution of the keys.
- I have created a User class demonstrating Alice and Bob.
- I have used Fernet (https://pypi.org/project/fernet/) for generating keys in my source code.
- For the simplicity I have kept threshold at 1 minute for the demonstration in my code.
- I have used Epoc Unix time as Timestamp in my demonstration code.

What results did I get and what did I learn?

I added both the fixes suggested above and demonstrated Needham's protocol for an intruder using previous session information for a replay attack.

Security Fix	**Results**
Extra Nonce to authenticate User 1	Protocol fails the authentication if Nonce is not sent back.
Timestamp added at step 4	Protocol fails the authentication if threshold exceeded
Timestamp added at step 5	Protocol fails the authentication if threshold exceeded

have also attached screenshot for timestamp demonstration:



```
Demonstrating protocol with intruder
Pre-shared key between Alice and Server: b'w8ziGuwvVYC7PvP4q0N73DisNb-4-ZGiggoFwT2daVc='
Pre-shared key between Bob and Server: b'SpqNZSq1WY3SJUNVNurgRXVtXZKBlW7OeM3du5KP_Og='
Pre-shared key between Eve and Server: b'8IHt_rqqBmlk7JGRNuF_bxw6uVirejzSLE3e5nLPYdw='

1 (Eve): N_A = 14476515844226490368
1 (Eve)=>Server: (A, B, N_A , E_BS_{A, N^B}) = (Eve, Bob, 14476515844226490368, E_{b'SpqNZSq1WY3SJUNVNurgRXVtXZKBlW7OeM3du5KP_Og='}
 (b'gAAAAABh6GbkBKSK56sE9PtVrJlxFdfLUfk37wG6BhcbIpltlUBEENkDYyfHWUH1uF-btudXewg5ERl9raoB4kDqiBuF-PBVMYRwfOgyr7AwlRdndUIB91cnp-ppf9RxF44LBIEc2xWE')


 2 (Server): K_AB = b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E='
2 (Server): E_{K_BS}(K_AB, A, N^B,T) = E_{b'SpqNZSq1WY3SJUNVNurgRXVtXZKBlW7OeM3du5KP_Og='}(b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E=', Eve, 9253796723520338329б 1642620644)
  = b'gAAAAABh6GbkCRvdSZ7BGVaPlkR1jnaepctuS1j-y7JVn7_gKsyoLPVEPwz_vwiqSeuBcEFgMq7Y5mPq8KRglml61srKSutfH1TIw2XJXDJKvJJXyuEm2-zDa2fb4NBBoIpX_KPyWeSRrnNk55lXrDPN8sqrCM_427y_RO
 -fWVDtgLt-WuvNnzurudRayAYJG2ZErnnObliFnspdGjJl2M74rs7Fj_q5OQ=='
2 (Server): E_{K_AS}(N_A, B, K_AB, P, T) = E_{b'8IHt_rqqBmlk7JGRNuF_bxw6uVirejzSLE3e5nLPYdw='}(14476515844226490368, Bob, b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E=',
 b'gAAAAABh6GbkCRvdSZ7BGVaPlkR1jnaepctuS1j-y7JVn7_gKsyoLPVEPwz_vwiqSeuBcEFgMq7Y5mPq8KRglml61srKSutfH1TIw2XJXDJKvJJXyuEm2-zDa2fb4NBBoIpX_KPyWeSRrnNk55lXrDPN8sqrCM_427y_RO
 -fWVDtgLt-WuvNnzurudRayAYJG2ZErnnObliFnspdGjJl2M74rs7Fj_q5OQ==', 1642620644) =
 b'gAAAAABh6Gbk55DUNjCxPcwzCPseW8xUkL6xTbaLyf_NywII1cOX96RRS42eJlS2u9bsOnM7tYoGnRfIMVdcd4HlnP_vtjYpgBB1NlO8Vn6rSq98wp3BjPnzdn8u
 --V2mUi9ZHJVdSAPQk8pNVT6X6PAsG_i0boF44jWhzPD9vuAJZ340nh-76VQJmx_Z4otpdpeUNPWWgY13rjHb5ck2TER0nx2T3mF4EShjSTqmBZBWOBlsWPyoirf5PO3l8Az3b3lGi8IUP_MVVxWI11NG97H5MT
 -mh36x2s31BAfJKZyYJN_3MiguDF0LKJJ7EjASdbYIajovieoAV4qEmtpm6oPRMlr3JPSVabTYJYHSzjXMHjCiaS3ixwlBLI0X9CU4yRVb8L7FfyIfk1M0nKkKoih_hf4GdF3zvkxn6Zrbo-opYqCN6wkOxCjUE
 -wdFaHDgcs8kfRzGemkh6E6jz7uL97-nVszQ9_9LCp8Plav_HkDuY1y24qL7xS41zZFZZp9F-AQL_eZ9QMggMy7mD1-gyj24S7taPqeg=='
{'N_A': 14476515844226490368, 'B': 'Bob', 'K_AB': b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E=', 'T': 1642620644, 'package':
 b'gAAAAABh6GbkCRvdSZ7BGVaPlkR1jnaepctuS1j-y7JVn7_gKsyoLPVEPwz_vwiqSeuBcEFgMq7Y5mPq8KRglml61srKSutfH1TIw2XJXDJKvJJXyuEm2-zDa2fb4NBBoIpX_KPyWeSRrnNk55lXrDPN8sqrCM_427y_RO
 -fWVDtgLt-WuvNnzurudRayAYJG2ZErnnObliFnspdGjJl2M74rs7Fj_q5OQ=='}
2 (Server => Eve): E_{K_AS} (N_A,B,K_AB,T,E_{K_BS}(K_AB,A,N^A)) = E_{b'8IHt_rqqBmlk7JGRNuF_bxw6uVirejzSLE3e5nLPYdw='} (14476515844226490368,Bob,
 b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E=',1642620644,
 b'gAAAAABh6GbkCRvdSZ7BGVaPlkR1jnaepctuS1j-y7JVn7_gKsyoLPVEPwz_vwiqSeuBcEFgMq7Y5mPq8KRglml61srKSutfH1TIw2XJXDJKvJJXyuEm2-zDa2fb4NBBoIpX_KPyWeSRrnNk55lXrDPN8sqrCM_427y_RO
 -fWVDtgLt-WuvNnzurudRayAYJG2ZErnnObliFnspdGjJl2M74rs7Fj_q5OQ==') = {'N_A': 14476515844226490368, 'B': 'Bob', 'K_AB': b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E=', 'T':
 1642620644, 'package': b'gAAAAABh6GbkCRvdSZ7BGVaPlkR1jnaepctuS1j-y7JVn7_gKsyoLPVEPwz_vwiqSeuBcEFgMq7Y5mPq8KRglml61srKSutfH1TIw2XJXDJKvJJXyuEm2
 -zDa2fb4NBBoIpX_KPyWeSRrnNk55lXrDPN8sqrCM_427y_RO-fWVDtgLt-WuvNnzurudRayAYJG2ZErnnObliFnspdGjJl2M74rs7Fj_q5OQ=='}
2 (Eve): E_{K_AS} (N_A,B,K_AB,T,E_{K_BS}(K_AB,A,N^A)) = E_{b'8IHt_rqqBmlk7JGRNuF_bxw6uVirejzSLE3e5nLPYdw='} (14476515844226490368,Bob,
 b'b1lpntrV2eTU6wl04j6WBtSQo1TyoYReSGi1miTGW9E=',1642620644,
 b'gAAAAABh6GbkCRvdSZ7BGVaPlkR1jnaepctuS1j-y7JVn7_gKsyoLPVEPwz_vwiqSeuBcEFgMq7Y5mPq8KRglml61srKSutfH1TIw2XJXDJKvJJXyuEm2-zDa2fb4NBBoIpX_KPyWeSRrnNk55lXrDPN8sqrCM_427y_RO
 -fWVDtgLt-WuvNnzurudRayAYJG2ZErnnObliFnspdGjJl2M74rs7Fj_q5OQ==')
=>Message 2 authentication was successful!


3 (Eve => Bob): E_{K_BS}(K_AB,A,N^A,T) = E_{b'SpqNZSq1WY3SJUNVNurgRXVtXZKBlW7OeM3du5KP_Og='}(b'BdNEXHVTmNZBnro3KCtiqfxZSkqtcv_YzveAayQCWOY=',Alice,37062851026868477952,
 1642618066) = b'gAAAAABh6FzSv91D4yyD6NnaXKokWA28aVB5IGCtKGTYtFi
 -FAVduD2MP588x2AFrmG_euCVk0zoKC38HpM7Cmk8XceAqyOqQIhVe4ZBuUHZY6LURd2xEwPM5BYufZFDLF6efre0LCsYfZL6vgQF9iR5Tm8nApmm4N2_Xae8yyEDKu_UkrwN4bUdz5zFv_707W5HcexfZuYiDwPoRYcRx
 -uyMUoIfInKKw=='

3 (Bob): (K_AB,A,N^B,T) = (b'BdNEXHVTmNZBnro3KCtiqfxZSkqtcv_YzveAayQCWOY=',Alice,37062851026868477952,1642618066)
=>Potential Replay Attack. Message 3 authentication was unsuccessful
=>Message 3 authentication was unsuccessful

Process finished with exit code 84
```

Learning Outcomes

I understood how nonces are useful and I can use this concept in my other algorithms as well because it ensures if A generates a random number(nonce) and send it to B and B then sends it back to A. A can be sure that the message is sent by B because only B can send it back the random number that A sent it.

Similarly, timestamp can also be used as a nonce which can improve this concept to identify replay attacks. Or both timestamps and nonce can be used together to increase the security more.

I have also realized how hard it is to make a protocol secure as for N-S protocol , it is important that both parties have secured connection with the server.

What went wrong or what could be improved?

- The demonstration here is based on a small scale. The demonstration needs to be evaluated on more complicated scenarios to find out more vulnerabilities and test multiple scenarios.
- Apart from this I think more sophisticated algorithm can be used to test the time stamps and to generate nonces to increase the probability that random number generated for Nonce has never been used before.

Source Code

The source code is provided as a separate zip file.