# Prevention Against Shoulder Surfing Attacks

*Submitted in Partial Fulfilment of the Requirements for the Course*

**Information Security Management**

**(CSE3502)**

*Under the Guidance of*

**Prof. Ruby D.**

*By*

| | |
|---|---|
| **Saloni Anand** | **18BCE2276** |
| **Sukriti Jaitly** | **18BCE0250** |
| **Divyang Arora** | **18BCE2251** |
| **Arnav Sharma** | **18BCE0868** |

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

**Date of Submission: May 29th, 2021**

## DECLARATION

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 6196

Student Name: Saloni Anand, Sukriti Jaitly, Divyang Arora, Arnav Sharma

Date of Submission: 29th May, 2021

Signature: Saloni Anand, Sukriti Jaitly, Divyang Arora, Arnav Sharma

## TABLE OF CONTENTS

# PROBLEM STATEMENT

## I. Idea

The idea behind the project is to use this algorithm so that the users are safe from key-loggers and eavesdropping by the attackers. Since every time the user enters the password through the **Dynamic Graphical Input**, the pattern changes every time, so the attacker records and enters the wrong password every time. While going through this course we learned that a lot of the websites have password encryptions where the entered password is encrypted using Algorithms like DES or AES.

Shoulder surfing, also known as eavesdropping, is a type of social engineering technique used to obtain information such as Personal Identification Numbers (PINs), login passwords, and other confidential data by looking over the victim's shoulder, either from device keystrokes or sensitive information being spoken and heard. This attack can be carried out from a short distance (by peering over the victim's shoulder) or from a greater distance (by utilising binoculars or equivalent devices). Attackers do not need any technical abilities to use this technique; all they need is a keen inspection of the victims' surroundings and a familiarity with the typing pattern. Instead of peering over one's shoulder, this attack can be carried out through key logging, in which the attackers track the key presses the user makes on the keyboard.

A user enters a password based on the type of input system. If the input system is pattern based, then the user enters a pattern to login. If the input system is code-word based then the user enters a code word to login. In this project, we intend to completely remove the chances of any type of Shoulder surfing attack. We introduce a new algorithm, which is effortless to implement but very effective. The algorithm along with the way the information is stored and fetched from the database makes it so secure, that even if a user tells the person his/her login password, he will never be able to log in to the system.

The methodology used in this project is that instead of using the keyboard as the primary input device, we will be using the mouse and give the password through a dynamic graphical input system. This type of system accepts a pattern based input that is entered in a code word form.

The outcome that we expect through this project is to stop all the attackers who are trying to get the different types of confidential data of the users by either shoulder surfing or key logging.

## II. Scope

Shoulder surfing is a strategy for gathering information that involves glancing over someone's shoulder. In crowded settings, shoulder surfing is a good technique to gather information because it's quite easy to stand next to someone and watch them fill out a form, input a PIN number at an ATM machine, or use a calling card at a public pay phone. Binoculars or other vision-enhancing gear can be used to do long-distance shoulder surfing. Experts propose shielding paperwork or your keypad from view with your body or cupping your hand to avoid shoulder surfing. **Eavesdropping** is another term for shoulder surfing.

Another attack which is discussed in this project is **Key-logging.**

**Keyloggers** are a sort of monitoring software that records a user's keystrokes. These keystroke loggers, one of the oldest types of cyber threat, record the information you type into a website or application and send it to a third party. Keyloggers gather data and transfer it to a third party, which could be a criminal, police enforcement, or IT department. In essence, keyloggers are software programmes that use algorithms to track keystrokes using pattern recognition and other methods.

## III. Novelty

There are mainly two types of **Secure Hash Algorithms**(SHA) and **Message Digest**(MDx).

Secure Hash Algorithms, or SHA, are a group of cryptographic functions that are used to keep data secure. It operates by converting data with a hash function, which is a bitwise operations, modular additions, and compression functions-based method. The hash function then generates a fixed-length string that bears no resemblance to the original.These algorithms are one-way functions, which means that it's nearly impossible to change them back into their original data once they've been changed into their hash values. SHA-1, SHA-2, and SHA-3 are three algorithms of interest, each of which was built with increasingly stronger encryption in response to hacker attempts. Because of its widely known flaws, SHA-0 is currently considered obsolete.

The server side just needs to maintain track of a specific user's hash value, rather than the actual password, when using SHA to encrypt passwords. This is useful in the event that an attacker breaks into the database and only finds the hashed functions, not the actual passwords. If they try to use the hashed value as a password, the hash function will transform it to another string and prevent access.

To generate a unique value from data and a unique symmetric key, message digest methods use cryptographic hash functions. A cryptographic hash function accepts data of any length and returns a single, fixed-length output. Because message digest algorithms always generate an encrypted value (which is never decrypted), they are frequently referred to as encryption-only algorithms.

When a sender and receiver share a unique symmetric key to compute a message digest value, secrecy is ensured, ensuring that the message digest value cannot be easily modified if the data is modified in an illegal or unanticipated manner. For the receiver to construct an identical message digest, both the sender and the receiver of the data (including the sender's message digest) must share the same key.

In this project we have used **MD5 Message Digest Algorithm**

**MD5** message digest algorithm is the 5th version of the **Message Digest Algorithm** developed by Ron Rivest to produce 128 bit message digest. MD5 is quite faster than other versions of message digest which takes the plain text of 512 bit blocks which is further divided into 16 blocks, each of 32 bits and produces the 128 bit message digest which is a set of four blocks, each of 32 bits. MD5 produces the message digest through five steps i.e. padding, append length, divide input into 512 bit blocks, initialize chaining variables a process blocks and 4 rounds, uses different constants in each iteration.

Furthermore, the combination of methods that we have used is completely new. We have combined OTP verification, MD5 algorithm, an arbitrary method of our own to convert the numerical password into a four digit word, and dynamic graphical input that stops any keyloggers. In addition the keypad is jumbled differently every time and a different pattern is to be used to enter the password. All this leads to our system having three layers of security and being so secure that anyone with malicious intent cannot login even if they know your password. This combination of methods also makes our system novel.

## IV. Comparative Statement

| Title | Year | Methodology | Advantage | Limitation |
|---|---|---|---|---|
| Password Encryption for hybrid cloud | 2019 | User credentials connected with an internal application being communicated to an external cloud service may be intercepted by a workspace cloud connection existing within an entity. | Since a workspace cloud connector is used for generating the encryption key, it provides an extra layer of encryption to the password that other methods do not. | If the path of the virtual delivery agent is damaged due to some situation then it would be impossible to check the password authentication since the encrypted password from either internal or external workspace cloud connector would not reach the other. |
| A Shoulder Surfing Resistant Graphical Authenticatio n System | 2018 | This paper's main concern is to prevent the shoulder surfing attacks and smudge attacks – which happen when the attacker is able to identify the smudges caused by the repetitive pattern making the user to unlock his/her mobile device. | As password is not a text-based password, instead an image-based password, this reduces the chances of a person identifying the password from naked eye or through video capture only once which is easy to use. | Since the whole model is based on the user's visual perception and his/her memory, this model would fail if the user is either confused or forgets the positioning of the segments on the image. To rectify this the user will have to re-register |
| Keylogger Is A Hacking Technique | 2018 | In the registration phase the user | Results were good by using and typing on the | Perpetrator can still exploit the process when |

| | | | | |
|---|---|---|---|---|
| That Allows Threatening Information On Mobile Banking User | | provides a unique username and selects a colour from the 8 options given by the website. | virtual keyboard in the smartphone device. | the victim is unaware of the surveillance of his smartphone. |
| A Novel Friendly Jamming Scheme in Industrial Crowdsensing Networks against Eavesdropping Attack | 2018 | Crowdsensing is a methodology that uses the power of the crowd to perform sensing tasks cooperatively and at a low cost. | The results show that our scheme can significantly decrease the eavesdropping risk compared with the no friendly jamming scenario and meanwhile that it maintains a low decrease on the transmission probability. | The whole system depends on the crowdsensing, that is depending on the people for the success of your system, which can include human error. |
| Improved keylogging and shoulder-surfing resistant visual two-factor authentication protocol | 2018 | In this paper, the deficiencies of two visual authentication protocols are demonstrated, then a two factor authentication scheme that eliminates these deficiencies is presented. | The analysis of the whole system shows that the system is secure, efficient and has a high level of usability which makes it easier for the user to understand the system and keep their data secure. | The attacker will be able to penetrate through the authentication process if the attacker is in possession of the two-factor authentication device. |
| IllusionPIN: Shoulder-Surfing Resistant Authentication Using Hybrid Images | 2017 | IllusionPIN (IPIN), a PIN-based authentication solution that works on touchscreen devices, | Based on the analysis, it seems practically almost impossible for a surveillance camera to capture the PIN | Makes a number of simplifying assumptions that limit the accuracy of our calculations. |

| | | | | |
|---|---|---|---|---|
| | | addresses the problem of shoulder-surfing attacks on authentication techniques. | of a smartphone user when IPIN is in use. | |

| | | | | |
|---|---|---|---|---|
| Revolving Flywheel PIN Entry Method to Prevent Shoulder Surfing Attacks | 2018 | Proposes a new scheme named revolving flywheel PIN-Entry method to prevent shoulder-surfing attacks. | Proposed method is easy to adapt and does not have any cognitive burden. This method will be easy to follow by ATM machines or wherever PIN is used for authentication. | In this method they have not used the middle layer, which could have been used to provide more security. |
| Keylogging Resistant Visual Authentication Protocols | 2018 | Their approach to determining the matter is to introduce an associate degree intermediate device that bridges an individual's user and a terminal. | A lot of specifically, their approach visualizes the safety method of authentication employing a smartphone aided increased reality. | This paper is not focused on authentication security and user experience thus making it easy for attackers to enter the system by breaking the authentication process. |
| Text based graphical password system to obscure shoulder surfing | 2018 | The goal of this study is to present a solution based on text-based graphical password approaches, such as the | Since Déjà vu is further replaced by text-based images rather than purely graphics, this takes away the keylogging advantage of the attacker. It | Human Error is very high in this process. |

| | | | | |
|---|---|---|---|---|
| | | Déjà vu and Moveable Frame schemes. | minimized the search time of pass images on the login screen | |

| | | | | |
|---|---|---|---|---|
| EvoPass: Evolvable graphical password against shoulder-surfing attacks | 2017 | Passes drawings as credentials are created from a group of user-selected pass pictures. Users are required to identify their pass sketches from a set of challenge images for user authentication. | Improves password strength over time by degrading pass sketches over time without requiring users to reselect pass images. The evolving feature makes it difficult for observational adversaries to identify the pass sketches, over time. | This strategy may work poorly for passwords that are not used frequently as a degrade in the pass sketch may lead to the user forgetting it. |
| An Improved Graphical Authentication System to Resist the Shoulder Surfing Attack | 2017 | Injecting the indirect pin into the system and using a visual password to authenticate the user's bank account is a clever approach to do it. | When the phone is misplaced, it protects the bank account from hackers by blocking it. The forget password module and banking service module is also very effective. | When a user's password is misused, the existing paper does not protect the user's account from hackers. |
| Image based authentication using zero-knowledge protocol | 2018 | They propose and examine the usability and security of | Hackers who try to eavesdrop the password will fail since the password is not | Since the whole system depends on the memory of the user knowing |

| | | | | |
|---|---|---|---|---|
| | | zero-knowledge protocol, where users prove they know the graphical password. w/o sending it. | sent over an insecure channel such as the Internet. As a result, it is a safe method of preventing undesired interception. | where each point on the image was located, there is a possibility that the user forgets the precise points location on the image and will have to re-register or reset the image points by some method. |

| | | | | |
|---|---|---|---|---|
| DyGazePass: A Gaze Gesture-Based Dynamic Authentication System to Counter Shoulder Surfing and Video Analysis Attacks | 2018 | This paper presents DyGazePass: Dynamic Gaze Passwords, an authentication strategy that uses dynamic gaze gestures. | With a 97.5 percent accuracy, the static-dynamic interface with a transition speed of two seconds is one of the most effective authentication techniques. | Users with colorblindness will have a limited set of colors to choose from as they can not distinguish a few colors. |
| Preventing Shoulder Surfing using Randomized Augmented Reality Keyboards | 2017 | They offer keyboard randomization as a simple yet effective countermeasure to several sorts of keystroke inference attacks in this study. | Acts as a strong defence against keyboard inference attacks from both the side and visual channels. | The camera resolution of the EPSON BT-200 is extremely low (640 × 480 pixels), which makes marker recognition error prone and difficult, especially if the user is at a distance from the keyboard. |

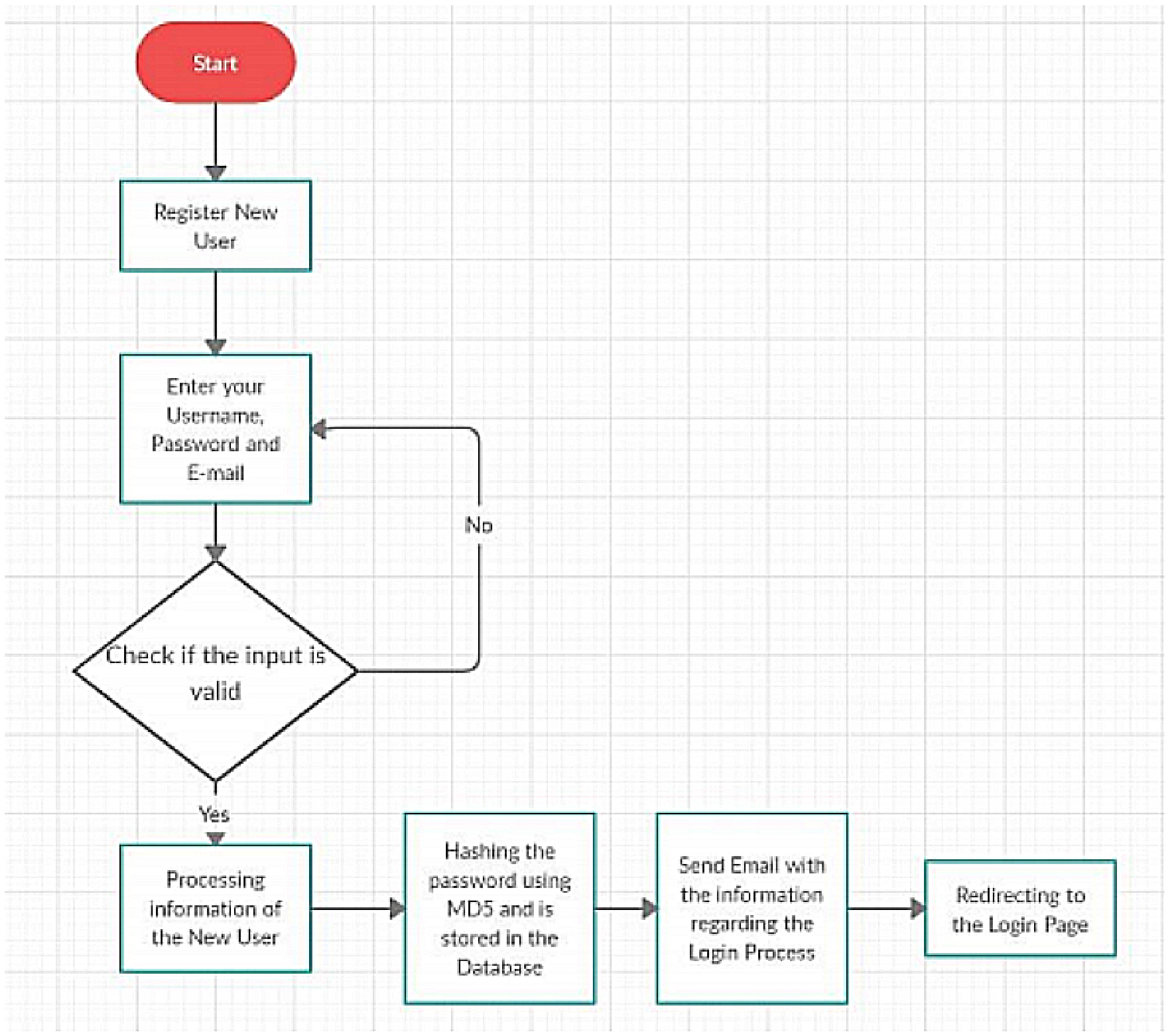
| | | | | |
|---|---|---|---|---|
| Graphical password: Prevent shoulder-surfing attack using digraph substitution rules | 2017 | A user is only required to click on one of the pass-images instead of both pass-images shown in each challenge set for three consecutive sets. | While this activity is simple enough to reduce login time, the photos clicked appear to be random and can only be retrieved by knowing the registration password as well as the activity rules. | During authentication, an adversary can use a shoulder surfing attack to obtain the correct credentials by memorising them. |

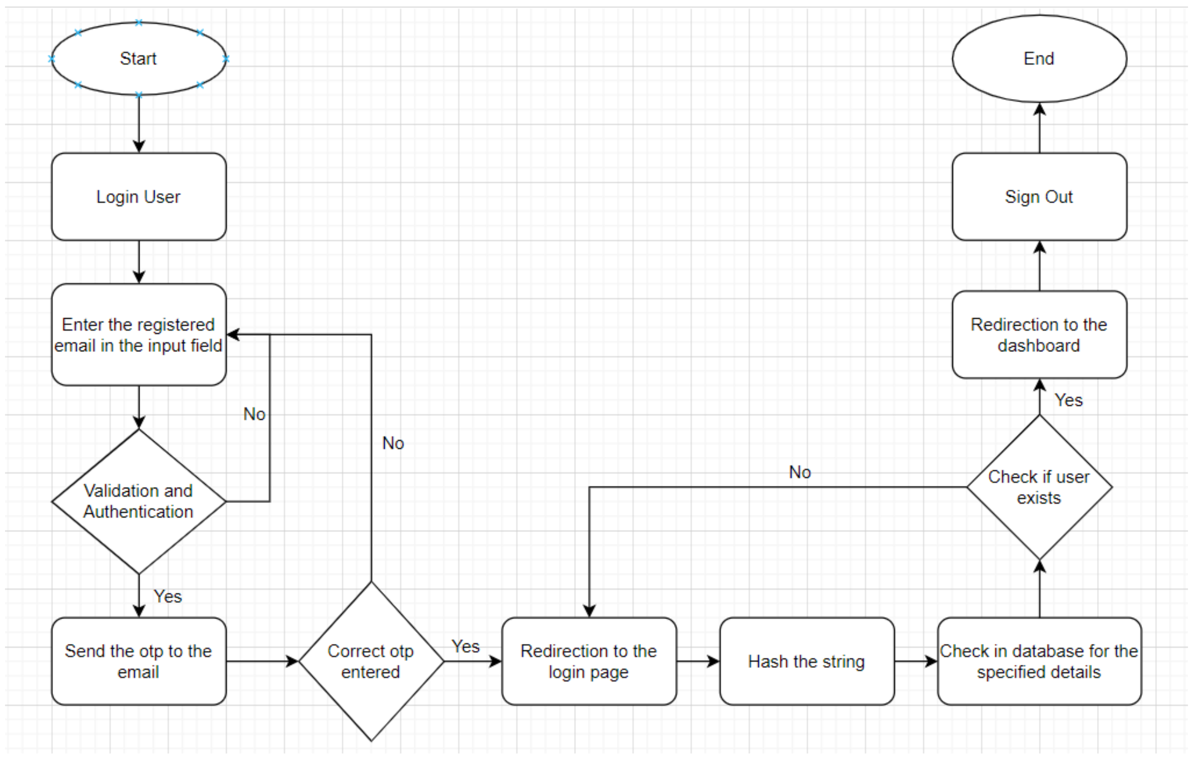
## V. Expected Result

The outcome that we expect through this project is to stop all the attackers who are trying to get the different types of confidential data of the users by either shoulder surfing or key logging.

## ARCHITECTURE

### I. High Level Design


Start
Register New User
Enter your Username, Password and E-mail
Check if the input is valid
No
Yes
Processing information of the New User
Hashing the password using MD5 and is stored in the Database
Send Email with the information regarding the Login Process
Redirecting to the Login Page


Start
Login User
Enter the registered email in the input field
Validation and Authentication
No
Yes
Send the otp to the email
Correct otp entered
No
Yes
Redirection to the login page
Hash the string
Check in database for the specified details
Check if user exists
No
Yes
Redirection to the dashboard
Sign Out
End

## II. Low Level Design

**Register Module:**

- Input username, email and password. We have included the following constraints on the user details.
  - Username should be unique
  - Password must be an eight digit number

    Eg. Username: Saloni
    Password: 12345678
    Email Id: saloni0408@gmail.com
- Check if entered details are valid.
- Convert password into a string. Following is the algorithm to do that
  - Divide the 8-digit number into four, two digit numbers .
  - E.g. 12345678 = [12, 34, 56, 78]
  - Convert each chunk into corresponding characters to generate a string of four characters. Below is the list of conversions:

○ [ 0-5: ‘a’, 6-11: ‘b’, 12-17: ‘c’, 18- 23: ‘d’, 24-29: ‘e’, 30-35: ‘f’, 36-41: ‘g’, 42- 47: ‘h’, 48-53: ‘i’, 54-59: ‘j’, 60-65: ‘k’, 66-71: ‘l’, 72-77: ‘m’, 78-83: ‘n’, 84-91: ‘#’, 91-99: ‘*’]

○ Therefore: 12345678 = [12, 34, 56, 78] = ‘cfjn’

● Compute every permutation of the string ○ [ ‘cfjn’, ‘fcjn’, ‘jcfn’, ‘cjfn’, ‘fjcn’, ‘jfcn’, ‘nfcj’, ‘fncj’, ‘cnfj’, ‘ncfj’, ‘fcnj’, ‘cfnj’, ‘cjnf’, ‘jcnf’, ‘ncjf’, ‘cnjf’, ‘jncf’, ‘njcf’, ‘njfc’, ‘jnfc’, ‘fnjc’, ‘nfjc’, ‘jfnc’, ‘fjnc’ ] ○ Total 4! = 24 permutations.

● Generate hash of each permutation using any hashing algorithm. We have used the MD5 hash algorithm in this project.

[‘501a93259d8f00aa7e4c2e9eda8e561e’,
‘7edea98f68d1d1e69ba39b6c6788dc45’,
‘2b5444b24e15b955a7455fcd49d7b897’,
‘c3dff954f4b96447ae2dfa96b95d0c17’,
‘99dde74f2613c8a304e0d02a8aeb5907’,
‘f46b40c76c99accb81cee28e01c7af4f’,
‘1ebe262273b1d93cb165726b19051b1b’,
‘439acea80e1d0124b1caf7fdbebc690f’,
‘ff9f526a2d7aed1da71d767d41445409’,
‘5d8999eb19e557dd865d4e5114fb06b9’,
‘cf83685381e1384c721ad9fd1a4879a4’,
‘b5b5e2088cd03f62b3a1d34e12568353’,
‘ac670e2a0e7de505f6f6b02d0949b966’,
‘f5e56809a788ec7baac614b21e6ac6ea’,
‘5398b787cb9e80c2c29c171d7c5f531f’,
‘a7a290d301d09f8aab46d440c1c0bc45’,
‘0f8698e809e4205ae5441f94dfa4b00d’,
‘b6b66056736b722b5e512f0492e1e36e’,
‘3dd8d0eb7c4d67179dd1c22c3d777a8e’,
‘12008658a74e6974ba34ddda2f2eda0a’,
‘3353f6fc45c701c655ad648ef7857815’,
‘545d3e0ae7566fb264df9d02f6cf49d6’,
‘097ee3281992b6f025cbc9a3f5e3b973’,
‘3475bb0f27f3a9614a79eae11cec1cf8’]

● Save the username, email, and each hash in the database.

**Login Module:**

Architecture of Security:

There are three levels of security in this project. All of these levels of work in a cascading manner as described below:

First layer: The user will enter their email, which is registered, to the system. If the email is not registered, the process will not proceed.

Second Layer: If the email entered is registered, then a verification will be processed on the registered device. This way provides two additional sub levels of security and an additional feature.

1. As only the genuine user will have access to the registered email, we will be sure that it is the real user and not the attacker who is accessing the system.

2. In case, the user no longer has access to the registered email. What a normal user does is recover their email using the ‘Forgot Password’ option. In any case, the otp will only be sent on the email and the attacker will not be able to reach the final level of Login.

3. Additional feature: The loophole in the Static Login system is that what if the attacker recognizes and remembers the pattern. The obvious solution is changing the pattern in each Login Session. That gives rise to another problem. We need to inform the user about that current session pattern. We cannot flash it directly onto the screen. When the user gets the otp on the registered email, there will also be an additional piece of information telling the user about the pattern distribution. The user remembers this distribution and enters the password in accordance with that.

Third Layer: The user will be asked to enter the details – Username and password through a unique input system. The user can enter any one password from the set as a code-word but actually, it is a pattern. For demo purpose, we included only two patterns: Horizontal distribution and Vertical distribution, as shown below:

| | | | |
|---|---|---|---|
| R1C1 (a) 0 | R1C2 (b) 1 | R1C3 (c) 2 | R1C4 (d) 3 |
| R2C1 (e) 4 | R2C2 (f) 5 | R2C3 (g) 6 | R2C4 (h) 7 |
| R3C1 (i) 8 | R3C2 (j) 9 | R3C3 (k) 10 | R3C4 (l) 11 |
| R4C1 (m) 12 | R4C2 (n) 13 | R4C3 (#) # | R4C4 (*) * |

Table 1: Horizontal Distribution

| | | | |
|---|---|---|---|
| R1C1 (a) 0 | R1C2 (e) 1 | R1C3 (i) 2 | R1C4 (m) 3 |
| R2C1 (b) 4 | R2C2 (f) 5 | R2C3 (j) 6 | R2C4 (n) 7 |
| R3C1 (c) 8 | R3C2 (g) 9 | R3C3 (k) 10 | R3C4 (#) 11 |
| R4C1 (d) 12 | R4C2 (h) 13 | R4C3 (l) # | R4C4 (*) * |

Table 2: Vertical Distribution

Now,

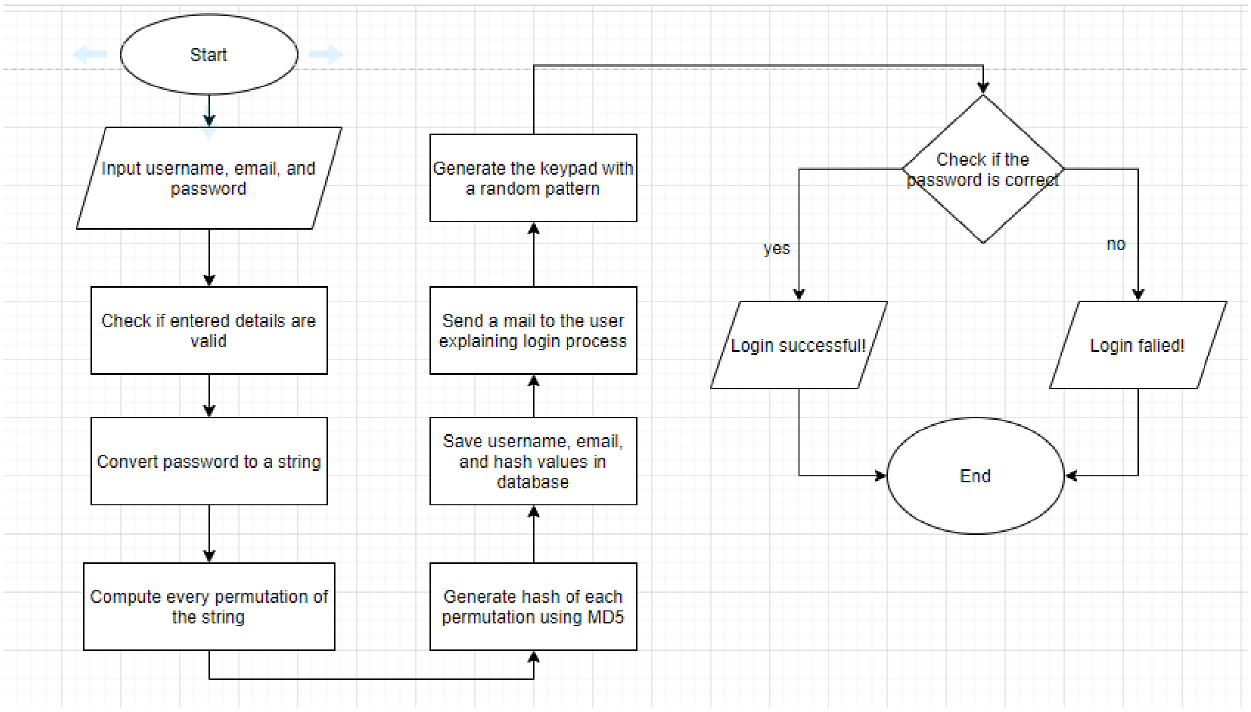
● Take an array of characters to display on the input buttons.

● Generate the pattern from this array as required.

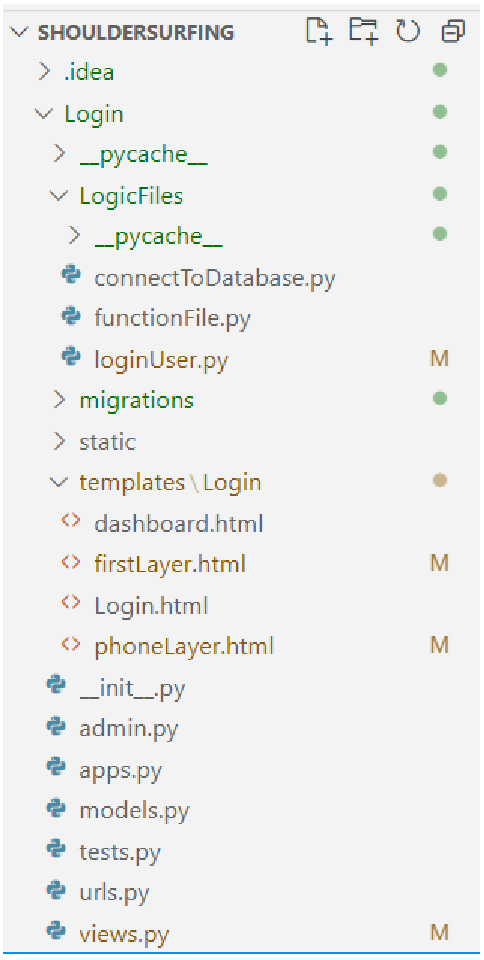
● Check the database for the existing pattern.

● If the pattern matches: Login Successful

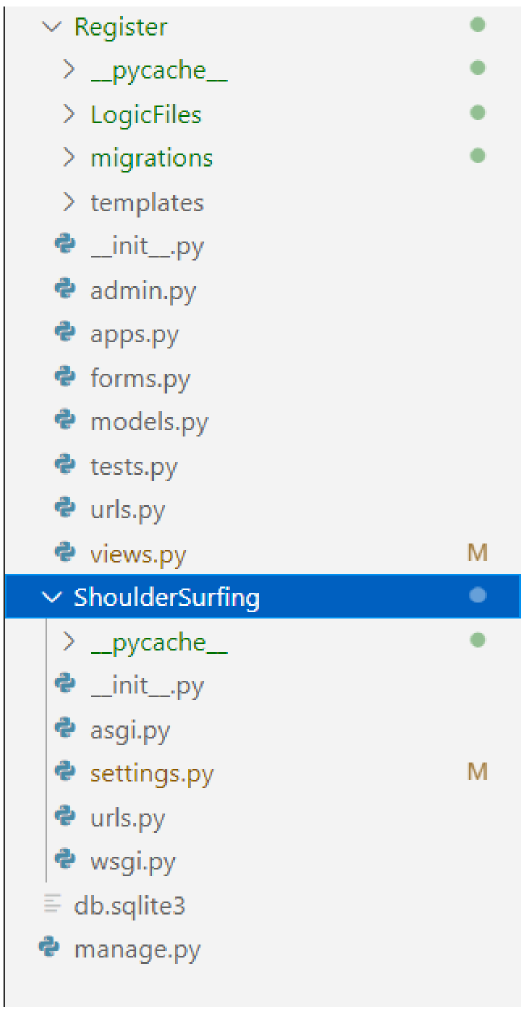
● Next Login Session, Change the display values of the grid.


Start
Input username, email, and password
Check if entered details are valid
Convert password to a string
Compute every permutation of the string
Generate hash of each permutation using MD5
Save username, email, and hash values in database
Send a mail to the user explaining login process
Generate the keypad with a random pattern
Check if the password is correct
yes
no
Login successful!
Login falied!
End

## File Structure:



```
SHOULDERSURFING
  .idea
  Login
    __pycache__
    LogicFiles
      __pycache__
      connectToDatabase.py
      functionFile.py
      loginUser.py                M
    migrations
    static
    templates\Login
      dashboard.html
      firstLayer.html             M
      Login.html
      phoneLayer.html             M
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py                      M
```

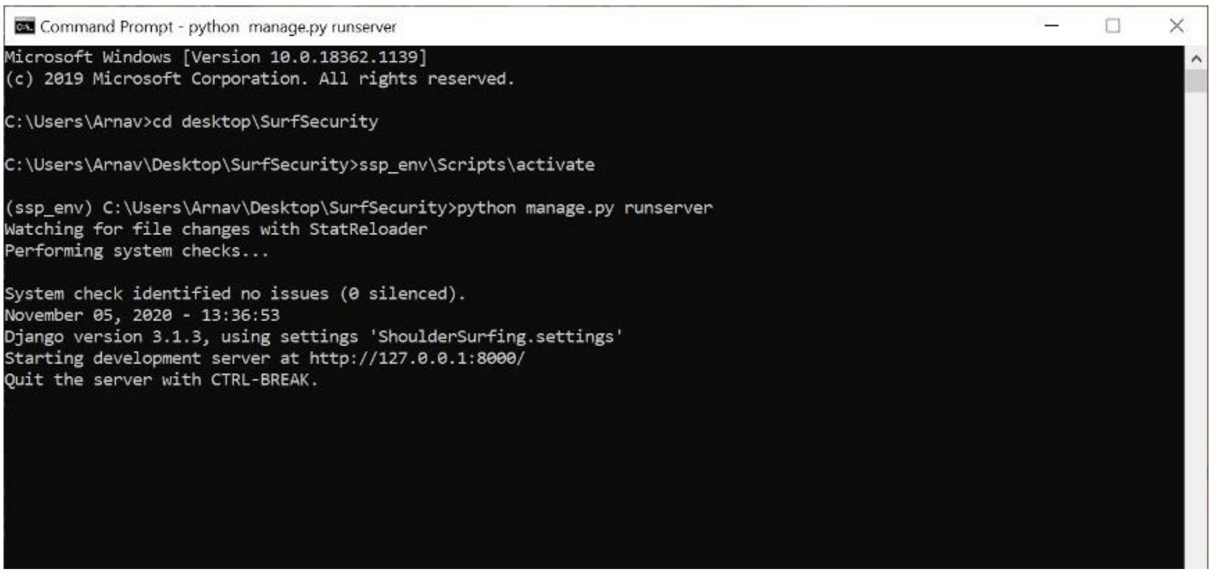


```
Register
  __pycache__
  LogicFiles
  migrations
  templates
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py                M
ShoulderSurfing
  __pycache__
  __init__.py
  asgi.py
  settings.py             M
  urls.py
  wsgi.py
db.sqlite3
manage.py
```

## IMPLEMENTATION

This project is based on python. We have used a Django based web app to implement the frontend mechanism. The project works in the following manner:

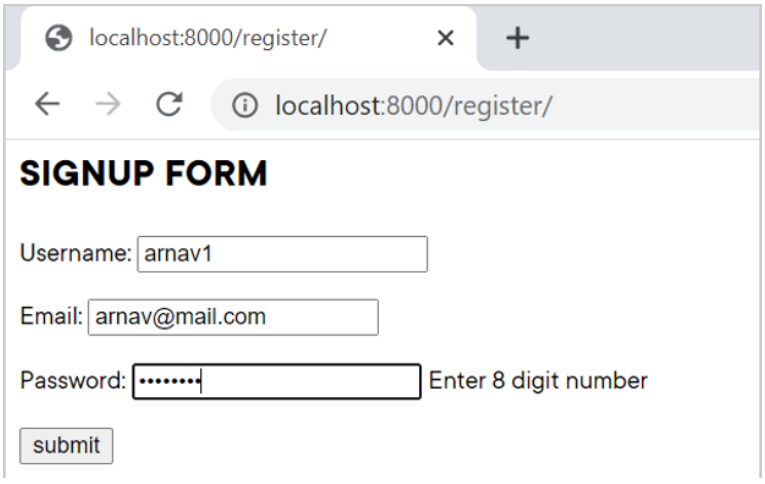
First, we run the python-based server on Django. Make sure Apache server and mysql are running on Xampp.



```
Command Prompt - python  manage.py runserver
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Arnav>cd desktop\SurfSecurity

C:\Users\Arnav\Desktop\SurfSecurity>ssp_env\Scripts\activate

(ssp_env) C:\Users\Arnav\Desktop\SurfSecurity>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 05, 2020 - 13:36:53
Django version 3.1.3, using settings 'ShoulderSurfing.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Fig: Running Django server

Now, we open the register page. This is where the user will register for the portal. Here they will set an 8 digit number as their password. For the sake of demonstration, the password set here is *12345678*.



localhost:8000/register/

SIGNUP FORM

Username: arnav1

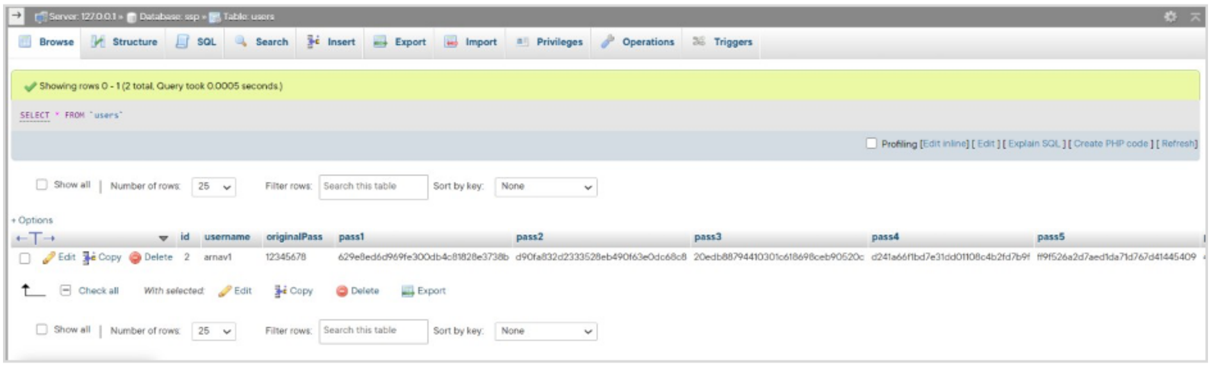
Email: arnav@mail.com

Password: •••••••• Enter 8 digit number

submit

As we can see in our database this account has been registered. The database stored the username,original password, and the 24 hashes of the code that will be their actual password. The user can use any one of those 24 codes.

It should be noted that since the password will be condensed to a 4 digit code. So even if the database is attacked, the attacker would not be able to login with the original 8 digit password as the login field will only accept a 4 digit code.



Server: 127.0.0.1 » Database: ssp » Table: users

Showing rows 0 - 1 (2 total, Query took 0.0005 seconds.)

```
SELECT * FROM `users`
```

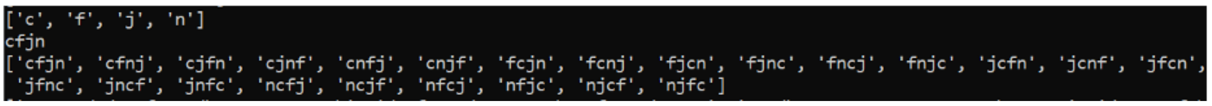
| id | username | originalPass | pass1 | pass2 | pass3 | pass4 | pass5 |
|---|---|---|---|---|---|---|---|
| 2 | arnav1 | 12345678 | 629e8ed6d969fe300db4c81828e3738b | d90fa832d2333528eb490f63e0dc68c8 | 20edb88794410301c618698ceb90520c | d241a66f1bd7e31dd01108c4b2fd7b9f | ff9f526a2d7aed1da71d767d41445409 |

Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Triggers
Profiling [Edit inline] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh]
Show all | Number of rows: 25
Filter rows: Search this table
Sort by key: None
+ Options
Edit Copy Delete
Check all With selected: Edit Copy Delete Export

Fig: Data is stored in Database

Now when the user goes to log in, they have to provide an email, on which device verification will be done as follows:

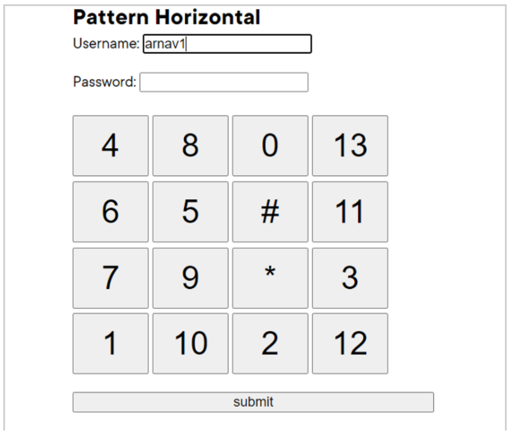
After this step, the user will be asked to give their password. As you can see below, when registering, the user's codes were generated in the backend. The user can now use any of these 24 codes to log in.



```
['c', 'f', 'j', 'n']
cfjn
['cfjn', 'cfnj', 'cjfn', 'cjnf', 'cnfj', 'cnjf', 'fcjn', 'fcnj', 'fjcn', 'fjnc', 'fncj', 'fnjc', 'jcfn', 'jcnf', 'jfcn',
 'jfnc', 'jncf', 'jnfc', 'ncfj', 'ncjf', 'nfcj', 'nfjc', 'njcf', 'njfc']
```

Fig: 24 permutations of the code

The user will also be notified about the pattern distribution being followed, via email. But for demonstration purposes we have displayed it on the screen (Horizontal in this example).



**Pattern Horizontal**

Username: arnav1

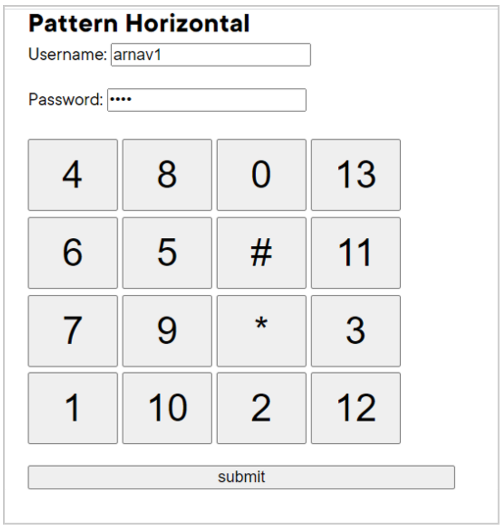
Password:

| | | | |
|---|---|---|---|
| 4 | 8 | 0 | 13 |
| 6 | 5 | # | 11 |
| 7 | 9 | * | 3 |
| 1 | 10 | 2 | 12 |

No as per the earlier mention architecture, the original password is divided into 4 parts:

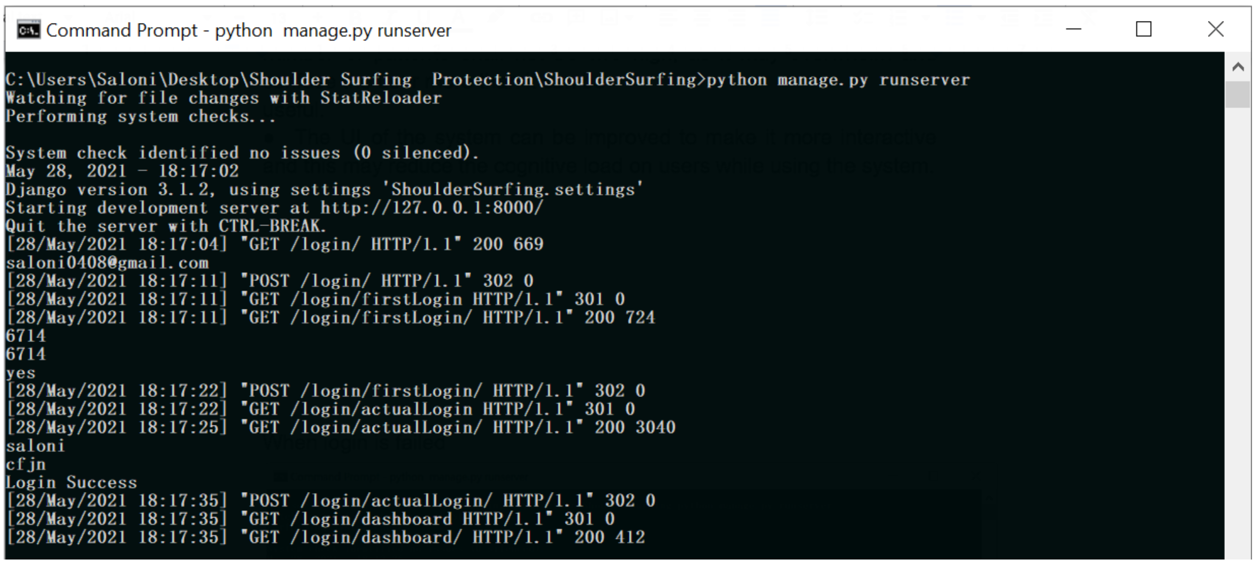
12:c 34:f 56:j 78:n

The 24 permutations of cfjn can all work as the passcode. As the pattern is horizontal, we put the passcode accordingly.



The same pattern is to be used every time, according to the distribution (horizontal or vertical).

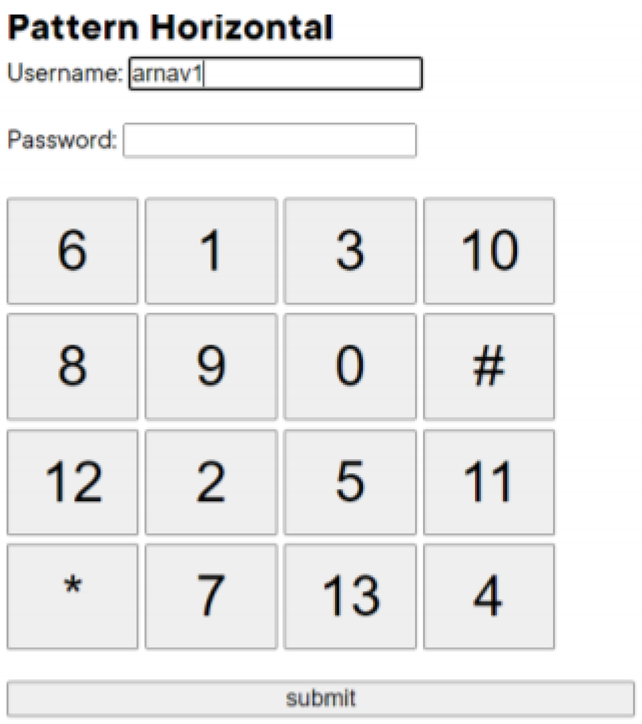
So the user was able to successfully sign in as shown below:



```
C:\Users\Saloni\Desktop\Shoulder Surfing  Protection\ShoulderSurfing>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 28, 2021 - 18:17:02
Django version 3.1.2, using settings 'ShoulderSurfing.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[28/May/2021 18:17:04] "GET /login/ HTTP/1.1" 200 669
saloni0408@gmail.com
[28/May/2021 18:17:11] "POST /login/ HTTP/1.1" 302 0
[28/May/2021 18:17:11] "GET /login/firstLogin HTTP/1.1" 301 0
[28/May/2021 18:17:11] "GET /login/firstLogin/ HTTP/1.1" 200 724
6714
6714
yes
[28/May/2021 18:17:22] "POST /login/firstLogin/ HTTP/1.1" 302 0
[28/May/2021 18:17:22] "GET /login/actualLogin HTTP/1.1" 301 0
[28/May/2021 18:17:25] "GET /login/actualLogin/ HTTP/1.1" 200 3040
saloni
cfjn
Login Success
[28/May/2021 18:17:35] "POST /login/actualLogin/ HTTP/1.1" 302 0
[28/May/2021 18:17:35] "GET /login/dashboard HTTP/1.1" 301 0
[28/May/2021 18:17:35] "GET /login/dashboard/ HTTP/1.1" 200 412
```

When we put the password as cfjn, a shoulder surfing attacker will think it is: 0 5 9 10, as can be seen above. As per human psychology, the attacker will remember the password as 0 5 9 10, not knowing that the actual password is actually the pattern in which the numbers were typed.

So when they try to log in using the password they know in the following keypad:



**Pattern Horizontal**

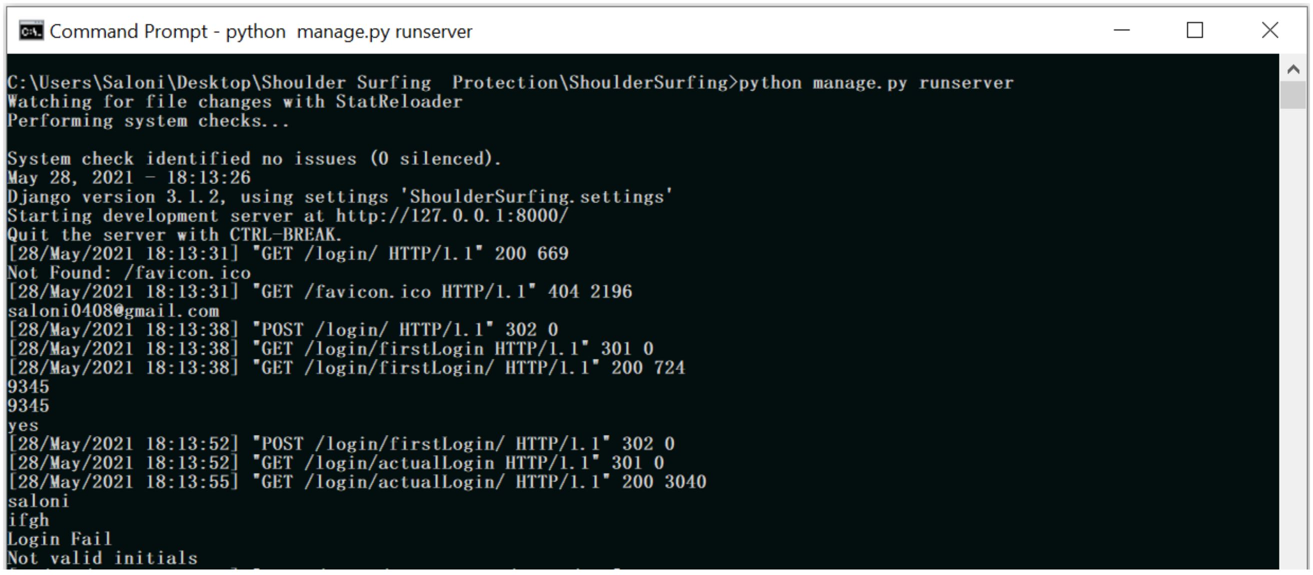
Username: arnav1

Password:

| | | | |
|---|---|---|---|
| 6 | 1 | 3 | 10 |
| 8 | 9 | 0 | # |
| 12 | 2 | 5 | 11 |
| * | 7 | 13 | 4 |

submit

Their login would fail as they would be putting the wrong patten, as shown below:



Command Prompt - python manage.py runserver

```
C:\Users\Saloni\Desktop\Shoulder Surfing  Protection\ShoulderSurfing>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 28, 2021 - 18:13:26
Django version 3.1.2, using settings 'ShoulderSurfing.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[28/May/2021 18:13:31] "GET /login/ HTTP/1.1" 200 669
Not Found: /favicon.ico
[28/May/2021 18:13:31] "GET /favicon.ico HTTP/1.1" 404 2196
saloni0408@gmail.com
[28/May/2021 18:13:38] "POST /login/ HTTP/1.1" 302 0
[28/May/2021 18:13:38] "GET /login/firstLogin HTTP/1.1" 301 0
[28/May/2021 18:13:38] "GET /login/firstLogin/ HTTP/1.1" 200 724
9345
9345
yes
[28/May/2021 18:13:52] "POST /login/firstLogin/ HTTP/1.1" 302 0
[28/May/2021 18:13:52] "GET /login/actualLogin HTTP/1.1" 301 0
[28/May/2021 18:13:55] "GET /login/actualLogin/ HTTP/1.1" 200 3040
saloni
ifgh
Login Fail
Not valid initials
```

So, the results obtained show that a shoulder surfing attack has been prevented successfully as the attacker will not be able to recognise the fact that the password is not the numbers, but the pattern in which the numbers have been typed.

## RESULTS AND DISCUSSION

### I. Implementation with Code

#### connectToDatabase.py

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="ssp"
)
cursor = mydb.cursor()

# createTableQuery = "CREATE TABLE users(id INT AUTO_INCREMENT PRIMARY
KEY, username VARCHAR (50) UNIQUE, " \
#                      "originalPass VARCHAR(50), pass1 VARCHAR(50), " \
#                      "pass2 VARCHAR(50), pass3 VARCHAR(50), pass4
VARCHAR(50), pass5 VARCHAR(50), pass6 VARCHAR(50), " \
#                      "pass7 VARCHAR(50), pass8 VARCHAR(50), pass9
VARCHAR(50), pass10 VARCHAR(50), pass11 VARCHAR(50), " \
#                      "pass12 VARCHAR(50), pass13 VARCHAR(50), pass14
VARCHAR(50), pass15 VARCHAR(50), pass16 VARCHAR(50), " \
#                      "pass17 VARCHAR(50), pass18 VARCHAR(50), pass19
VARCHAR(50), pass20 VARCHAR(50), pass21 VARCHAR(50), " \
#                      "pass22 VARCHAR(50), pass23 VARCHAR(50), pass24
VARCHAR(50))"
#
# cursor.execute(createTableQuery)
```

#### functionFile.py

```
from itertools import permutations
import hashlib

def convertToString(list):
    string = ""
    for element in list:
        string += element
    return string
```

```python
def createDefaultIteration(chunkList):
    defaultIteration = []
    for number in chunkList:
        if (number >= 0 and number <= 5):
            defaultIteration.append('a')
        elif (number >= 6 and number <= 11):
            defaultIteration.append('b')
        elif (number >= 12 and number <= 17):
            defaultIteration.append('c')
        elif (number >= 18 and number <= 23):
            defaultIteration.append('d')
        elif (number >= 24 and number <= 29):
            defaultIteration.append('e')
        elif (number >= 30 and number <= 35):
            defaultIteration.append('f')
        elif (number >= 36 and number <= 41):
            defaultIteration.append('g')
        elif (number >= 42 and number <= 47):
            defaultIteration.append('h')
        elif (number >= 48 and number <= 53):
            defaultIteration.append('i')
        elif (number >= 54 and number <= 59):
            defaultIteration.append('j')
        elif (number >= 60 and number <= 65):
            defaultIteration.append('k')
        elif (number >= 66 and number <= 71):
            defaultIteration.append('l')
        elif (number >= 72 and number <= 77):
            defaultIteration.append('m')
        elif (number >= 78 and number <= 83):
            defaultIteration.append('n')
        elif (number >= 84 and number <= 91):
            defaultIteration.append('#')
        elif (number >= 92 and number <= 99):
            defaultIteration.append('*')
        else:
            print("Not a valid chunk")
    return defaultIteration

def convertEachPermutationListToString(everyPermutation):
    everyPermutationList = []
    for permutation in everyPermutation:
        permutation = convertToString(permutation)
        everyPermutationList.append(permutation)
    return everyPermutationList

def hashEveryPermutation(everyPermutationList):
    everyHashedPermutationList = []
    for permutation in everyPermutationList:
        result = hashlib.md5(permutation.encode())
        everyHashedPermutationList.append(result.hexdigest())
    return everyHashedPermutationList

def hashLoginPassword(passwordLogin):
    passwordLogin = hashlib.md5(passwordLogin.encode())
    passwordLogin = passwordLogin.hexdigest()
    return passwordLogin

def makePasswordEven(dummyPassword):
    integerList = [int(x) for x in dummyPassword]
    print(integerList)
    print(type(integerList))
    integerList.append(9)
    print(integerList)
```

**loginUser.py**

```
from .connectToDatabase import *
from .functionFile import *

def signin(usernameLogin, passwordLogin):

    searchSQLQuery = "SELECT * from  users WHERE username = %s"
    usernameLogin = (usernameLogin,)
    cursor.execute(searchSQLQuery, usernameLogin)

    result = cursor.fetchall()

    passwordLogin = hashLoginPassword(passwordLogin)

    count = 0
    for row in result:
        for element in row:
            if (element == passwordLogin):
                count = 1
                break
    if(count == 1):
        print("Login Success")
        return True
    else:
        print("Login Fail")
        return False
```

**login views.py**

```
from django.shortcuts import render, redirect
import random
from .LogicFiles.loginUser import signin
from django.conf import settings
from django.core.mail import send_mail
current = ''
username = ''
phoneNo = ''
otp = 0


def dashboard(request):
      context = {}
      context['username'] = username

      if request.GET.get("logout"):
            return redirect("/login/")

      return render(request, "Login/dashboard.html", context)


def LoginUser(request):
      context = {}
      displayValue = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
      '10', '11', '12', '13', '#', '*']
      random.shuffle(displayValue)

      choiceList = [0, 1]
      choice = random.choice(choiceList)
```

```
  if (choice == 0):
        horver = "horizontal"
  else:
        horver = "vertical"

subject = 'choice'
message = f'Hi, thank you. your choice is {horver}.'
email_from = settings.EMAIL_HOST_USER
recipient_list = [phoneNo, ]
send_mail( subject, message, email_from, recipient_list )

context['displayvaluebuttonone'] = displayValue[0]
context['displayvaluebuttontwo'] = displayValue[1]
context['displayvaluebuttonthree'] = displayValue[2]
context['displayvaluebuttonfour'] = displayValue[3]
context['displayvaluebuttonfive'] = displayValue[4]
context['displayvaluebuttonsix'] = displayValue[5]
context['displayvaluebuttonseven'] = displayValue[6]
context['displayvaluebuttoneight'] = displayValue[7]
context['displayvaluebuttonnine'] = displayValue[8]
context['displayvaluebuttonten'] = displayValue[9]
context['displayvaluebuttoneleven'] = displayValue[10]
context['displayvaluebuttontwelve'] = displayValue[11]
context['displayvaluebuttonthirteen'] = displayValue[12]
context['displayvaluebuttonfourteen'] = displayValue[13]
context['displayvaluebuttonfifteen'] = displayValue[14]
context['displayvaluebuttonsixteen'] = displayValue[15]

if (choice == 0):
    context['pattern'] = 'Pattern Horizontal'
    context['buttononevalue'] = 'a'
    context['buttontwovalue'] = 'b'
    context['buttonthreevalue'] = 'c'
    context['buttonfourvalue'] = 'd'
    context['buttonfivevalue'] = 'e'
    context['buttonsixvalue'] = 'f'
    context['buttonsevenvalue'] = 'g'
    context['buttoneightvalue'] = 'h'
    context['buttonninevalue'] = 'i'
    context['buttontenvalue'] = 'j'
    context['buttonelevenvalue'] = 'k'
    context['buttontwelvevalue'] = 'l'
    context['buttonthirteenvalue'] = 'm'
```

```
        context['buttonfourteenvalue'] = 'n'
        context['buttonfifteenvalue'] = '#'
        context['buttonsixteenvalue'] = '*'

    elif(choice ==1):
        context['pattern'] = 'Pattern Vertical'
        context['buttononevalue'] = 'a'
        context['buttontwovalue'] = 'e'
        context['buttonthreevalue'] = 'i'
        context['buttonfourvalue'] = 'm'
        context['buttonfivevalue'] = 'b'
        context['buttonsixvalue'] = 'f'
        context['buttonsevenvalue'] = 'j'
        context['buttoneightvalue'] = 'n'
        context['buttonninevalue'] = 'c'
        context['buttontenvalue'] = 'g'
        context['buttonelevenvalue'] = 'k'
        context['buttontwelvevalue'] = '#'
        context['buttonthirteenvalue'] = 'd'
        context['buttonfourteenvalue'] = 'h'
        context['buttonfifteenvalue'] = 'l'
        context['buttonsixteenvalue'] = '*'

    logindata = request.POST or None
    if request.POST.get("pass"):
        global current
        current = current + logindata['pass']

    if request.POST.get("save"):
        print(logindata['username'])
        print(logindata['password'])
        global username
        username = logindata['username']
        permission = signin(logindata['username'],
logindata['password'])
        if (permission):
            return redirect("/login/dashboard")
        else:
            print("Not valid initials");

    return render(request, "Login/Login.html", context)

def firstLayerAuthentication(request):
    firstLayerData = request.POST or None

    if request.POST.get("firstLayerSave"):
        answer = firstLayerData['answer']
        print(otp)
        print(answer)
        if(answer == str(otp)):
            print('yes')
            return redirect('/login/actualLogin')

    return render(request, "Login/firstLayer.html")

def phoneLayer(request):
    global otp
    otp = random.randint(1111,9999)
    phoneLayerData = request.POST or None
    if request.POST.get("phoneLayerSave"):
        global phoneNo
        phoneNo = phoneLayerData["phone"]
        subject = 'Project OTP'
        message = f'Hi, thank you. your otp is {otp}.'
        email_from = settings.EMAIL_HOST_USER
        recipient_list = [phoneNo, ]
        send_mail( subject, message, email_from, recipient_list )
        print(phoneNo)
        return redirect('/login/firstLogin')
    return render(request, "Login/phoneLayer.html")
```

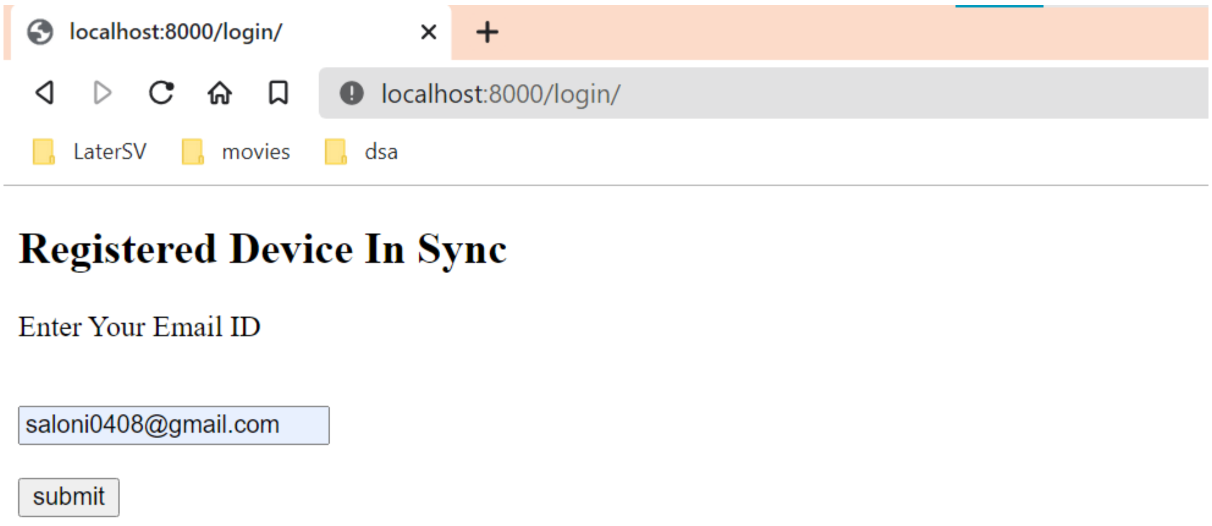
**template login.html**

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <script
      src="https://code.jquery.com/jquery-3.4.1.min.js"
      integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
      crossorigin="anonymous">
    </script>
    <link rel="stylesheet" href="{% static 'css/Login.css' %}">
</head>
```

```
<body>

    <form method="POST" action="" id="form">
        {% csrf_token %}

        <h2>{{ pattern }}</h2>

        <label for="username" class="labeluser">Username:</label>
        <input type="text" name="username" id="username"> <br><br>

        <label for="passfield" class="labeluser">Password:</label>
        <input type="password" id="passfield" name="password"> <br><br>

        <button type="button" value= '{{ buttononevalue }}' name="pass"
onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonone }}</button>
        <button type="button" value= '{{ buttontwovalue }}' name="pass"
onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttontwo }}</button>
        <button type="button" value= '{{ buttonthreevalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonthree }}</button>
        <button type="button" value= '{{ buttonfourvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonfour }}</button>

        <br>
        <button type="button" value= '{{ buttonfivevalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonfive }}</button>
        <button type="button" value= '{{ buttonsixvalue }}' name="pass"
onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonsix }}</button>
        <button type="button" value= '{{ buttonsevenvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonseven }}</button>
        <button type="button" value= '{{ buttoneightvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttoneight }}</button>
        <br>
        <button type="button" value= '{{ buttonninevalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonnine }}</button>
```

```
        <button type="button" value= '{{ buttontenvalue }}' name="pass"
onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonten }}</button>
        <button type="button" value= '{{ buttonelevenvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttoneleven }}</button>
        <button type="button" value= '{{ buttontwelvevalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttontwelve }}</button>
        <br>
        <button type="button" value= '{{ buttonthirteenvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonthirteen }}</button>
        <button type="button" value= '{{ buttonfourteenvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonfourteen }}</button>
        <button type="button" value= '{{ buttonfifteenvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonfifteen }}</button>
        <button type="button" value= '{{ buttonsixteenvalue }}'
name="pass" onclick = "add(this.value)" class="passwordbutton">{{
displayvaluebuttonsixteen }}</button>

        <br><br>

        <input type="submit" value="submit" name="save">
    </form>

    <script>
        function add(buttonValue){
            currentvalue = document.getElementById('passfield').value;
            updatedValue = currentvalue + buttonValue;
            document.getElementById('passfield').value = updatedValue;
        }
    </script>

</body>
</html>
```

## II. Results

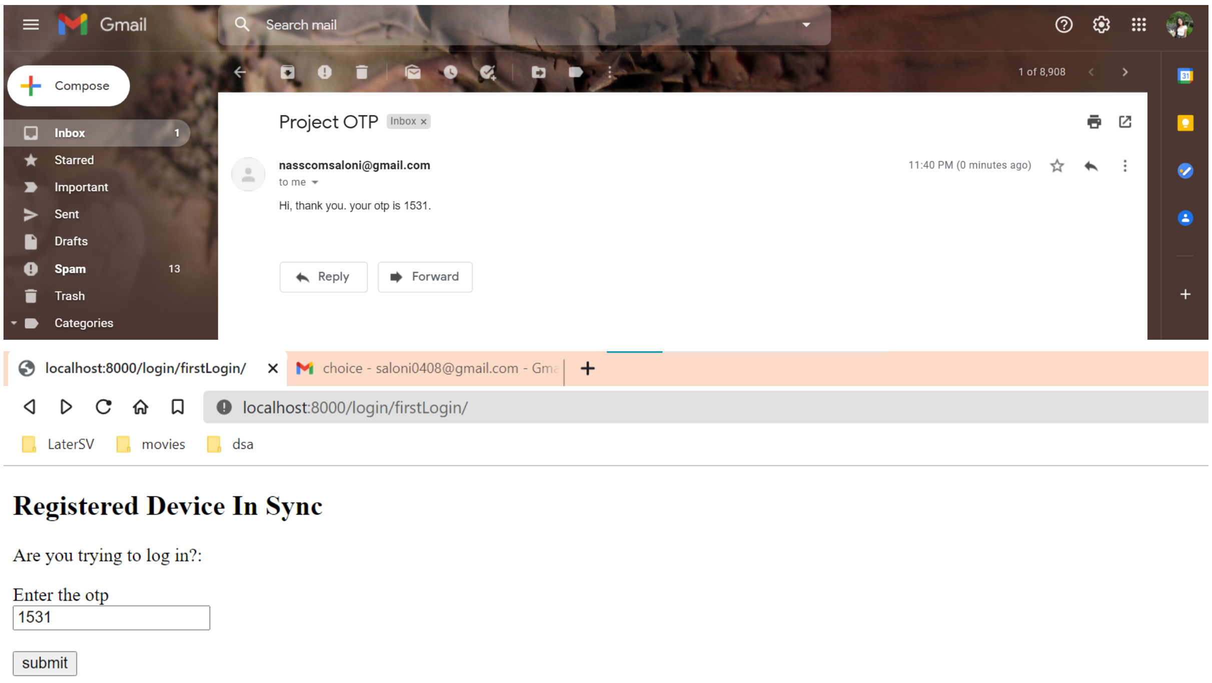

localhost:8000/login/

# Registered Device In Sync

Enter Your Email ID

saloni0408@gmail.com

submit


Project OTP

nasscomsaloni@gmail.com

to me

11:40 PM (0 minutes ago)

Hi, thank you. your otp is 1531.

localhost:8000/login/firstLogin/

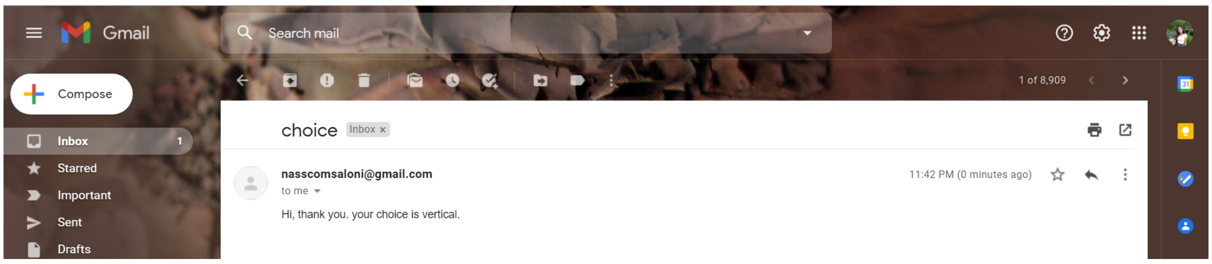
# Registered Device In Sync

Are you trying to log in?:

Enter the otp

1531

submit

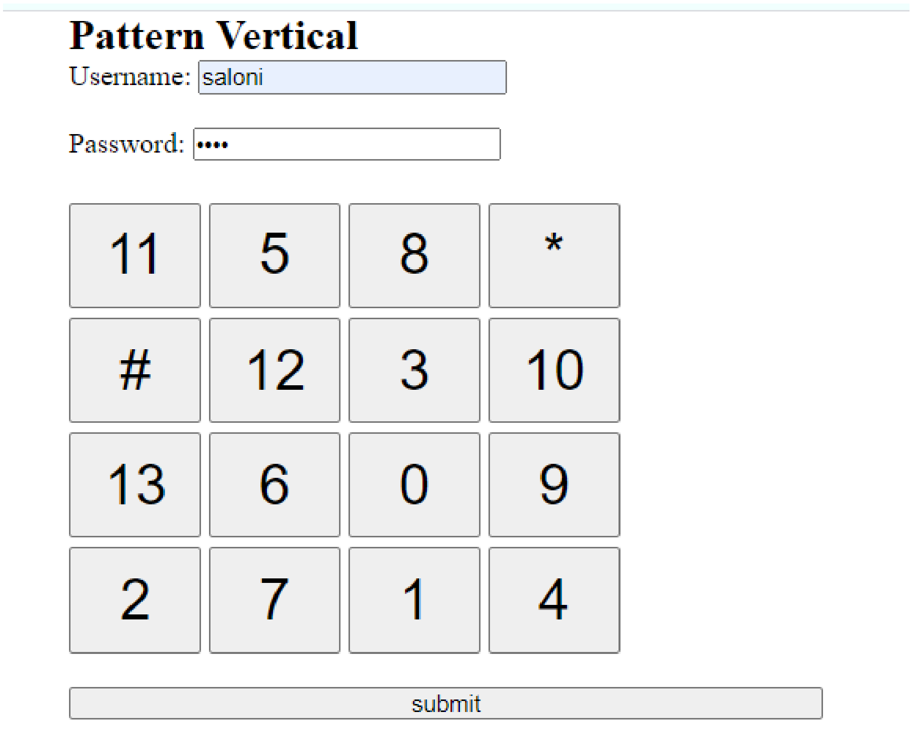


choice Inbox ×

nasscomsaloni@gmail.com
to me

11:42 PM (0 minutes ago)

Hi, thank you. your choice is vertical.

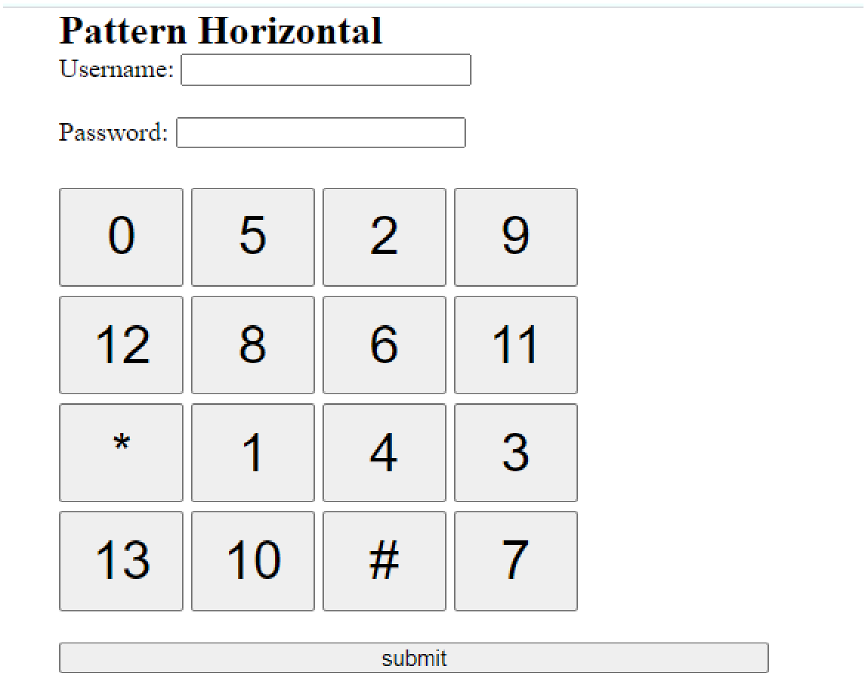


Pattern Vertical

Username: saloni

Password: ••••

| | | | |
|---|---|---|---|
| 11 | 5 | 8 | * |
| # | 12 | 3 | 10 |
| 13 | 6 | 0 | 9 |
| 2 | 7 | 1 | 4 |



Pattern Horizontal

Username:

Password:

| | | | |
|---|---|---|---|
| 0 | 5 | 2 | 9 |
| 12 | 8 | 6 | 11 |
| * | 1 | 4 | 3 |
| 13 | 10 | # | 7 |

## Analysis of the Results Obtained

Since the major aim of this project was to use an algorithm for password that can neither be understood by a hacker, a mouse heatmap etc nor by a person standing beside you trying to see the password we thought it would be best to ask people via screen share if they understood what the password is.

No one could login to our account even when they saw the password being typed. We believe this could be an important password input mechanism in the future

## Comparison of the obtained results with the already existing results

Since this is a novel idea we could not find any such ideas done before

## Efficiency obtained along with the metrics used

This project is not that algorithmically complex and the only encryption essentially being used is the hashing algorithm.

· **Case 1** – UUID is cached when string has a length of 36 characters

· **Case 2** – 49 characters length string, UUID is cached and system time stamp is calculated each iteration

· **Case 3** – 49 characters length string, new UUID is generated on each iteration and system timestamp is calculated each iteration

· **Case 4** – UUID is cached when string has a length of 72 characters.

· **Case 5** – 85 characters length string, UUID is cached and system time stamp is calculated each iteration

· **Case 6** – 85 characters length string, new UUID is generated on each iteration and system timestamp is calculated each iteration

· SHA-256 is 31% faster than SHA-512 only when hashing small strings. When the string is longer, SHA-512 is 2.9 percent faster.

· Per 1 million iterations, it takes ~121.6 milliseconds to get a system timestamp.

· Time to generate UUID is ~670.4 ms per 1M iterations.

· SHA-1 is the fastest hashing function with ~587.9 ms per 1M operations for short strings and 881.7 ms per 1M for longer strings.

· MD5 is 7.6% slower than SHA-1 for short strings and 1.3% for longer strings.

· For small strings, SHA-256 is 15.5 percent slower than SHA-1, and for larger strings, it is 23.4 percent slower.

· SHA-512 is 51.7% slower than SHA-1 for short strings and 20% for longer.

## III. Mapping the results with problem statement and existing systems

In the duration of this project, we have formulated a system which helps in preventing shoulder surfing attacks on users while they log in to their accounts in public places. In our project, the user signs up with an 8 digit password which is then converted to 4 digit code, whose permutations are stored in the database. Then the user is notified about the way their new log in works and are told the actual codes via mail. It is interesting to note that this system is also prone to attacks on the database and keyloggers as the password shown in the database is not the actual password, nor is the data that is entered as the password. It has also been demonstrated how a shoulder surfer would fail at logging and an analysis of the results has been prevented.

The project can be extended to add various other features.

- The project can be incorporated with actual apps to see how it works in real time.
- For the purpose of demonstration, only two patterns are included in it, but multiple patterns can be added to increase the complexity but the number of patterns shall not be two high, as it may overwhelm and confuse the user more than it can be useful.
- The UI of the system can be improved to make it more interactive and this may reduce the cognitive load on users while using the system.

## REFERENCES

1. Password Encryption for hybrid cloud services, Singleton IV, L. C., & Cooper, A. (2019). U.S. Patent No. 10,432,592. Washington, DC: U.S. Patent and Trademark Office.

2. H. Sun, S. Chen, J. Yeh and C. Cheng, "A Shoulder Surfing Resistant Graphical Authentication System," in IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 2, pp. 180-193, 1 March-April 2018, doi: 10.1109/TDSC.2016.2539942.

3. A. P. Kuncoro and B. A. Kusuma, "Keylogger Is A Hacking Technique That Allows Threatening Information On Mobile Banking User," 2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 2018, pp. 141-145, doi: 10.1109/ICITISEE.2018.8721028.

4. Li, Xuran & Wang, Qiu & Dai, Hong-Ning & Wang, Hao. (2018). A Novel Friendly Jamming Scheme in Industrial Crowdsensing Networks against Eavesdropping Attack. Sensors. 18. 1938. 10.3390/s18061938.

5. Khedr, Walid. (2018). Improved keylogging and shoulder-surfing resistant visual two-factor authentication protocol. Journal of Information Security and Applications. 39. 41-57. 10.1016/j.jisa.2018.02.003.

6. A. Papadopoulos, T. Nguyen, E. Durmus and N. Memon, "IllusionPIN: Shoulder-Surfing Resistant Authentication Using Hybrid Images," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 12, pp. 2875-2889, Dec. 2017, doi: 10.1109/TIFS.2017.2725199.

7. O. K. Kasat and U. S. Bhadade, "Revolving Flywheel PIN Entry Method to Prevent Shoulder Surfing Attacks," 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, India, 2018, pp. 1-5, doi: 10.1109/I2CT.2018.8529758.

8. D. Nyang, A. Mohaisen and J. Kang, "Keylogging-Resistant Visual Authentication Protocols," in IEEE Transactions on Mobile Computing, vol. 13, no. 11, pp. 2566-2579, Nov. 2014, doi: 10.1109/TMC.2014.2307331.

9. K. Irfan, A. Anas, S. Malik and S. Amir, "Text based graphical password system to obscure shoulder surfing," 2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 2018, pp. 422-426, doi: 10.1109/IBCAST.2018.8312258.

10. Yu, Xingjie & Wang, Zhan & Li, Yingjiu & Li, Liang & Zhu, Wen & Song, Li. (2017). EvoPass: Evolvable graphical password against shoulder-surfing attacks. Computers & Security. 70. 10.1016/j.cose.2017.05.006.

11. R. Sudha and M. Shanmuganathan, "An Improved Graphical Authentication System to Resist the Shoulder Surfing Attack," 2017 International Conference on Technical Advancements in Computers and Communications (ICTACC), Melmaurvathur, 2017, pp. 53-55, doi: 10.1109/ICTACC.2017.23.

12. Z. Mohamad, L. Y. Thong, A. H. Zakaria and W. S. W. Awang, "Image based authentication using zero-knowledge protocol," 2018 4th International Conference on Computer and Technology Applications (ICCTA), Istanbul, Turkey, 2018, pp. 202-210, doi: 10.1109/CATA.2018.8398683.

13. V. Rajanna, A. H. Malla, R. A. Bhagat and T. Hammond, "DyGazePass: A gaze gesture-based dynamic authentication system to counter shoulder surfing and video analysis attacks," 2018 IEEE 4th International Conference on Identity, Security, and Behavior Analysis (ISBA), Singapore, 2018, pp. 1-8, doi: 10.1109/ISBA.2018.8311458.

14. A. Maiti, M. Jadliwala and C. Weber, "Preventing shoulder surfing using randomized augmented reality keyboards," 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, 2017, pp. 630-635, doi: 10.1109/PERCOMW.2017.7917636.

15. Por, L.Y., Ku, C.S., Islam, A. et al. Graphical password: prevent shoulder-surfing attack using digraph substitution rules. Front. Comput. Sci. 11, 1098–1108 (2017) https://doi.org/10.1007/s11704-016-5472-z