RHYTHMIC KEYLOGGING FOR AUTHENTICATION

Matriculation Number: 210027910

***Abstract* – Passwords are not as secure as many would hope and most current generation 2nd factor authentication methods are inconvenient or expensive. The use of typing rhythm as an authentication method is an attractive alternative to existing methods. The background of typing rhythm is explored. Using a custom keylogger and small trial group, an experiment to assess the suitability of typing rhythm as an authentication method is conducted. The performance of simple vs complex typing rhythms are discussed and the effect of username length on authentication performance is determined.**

***Keywords* – Security, Credentials, Username, Password, Authentication, Keylogger, Rhythm, Experiment, Trial, Keystroke Dynamics, Keystroke Analysis, Typing Biometrics, Typing Rhythms.**

***Word Count (excluding references, appendices)*: 4579**

1 INTRODUCTION

This document discusses the planning and execution of a prototypical experiment using a small trial group. Using a custom key logger, timings are taken for a user entering credentials, with the purpose of determining the suitability of using the typing rhythm with which the credentials are entered as a form of authentication. This authentication method is then evaluated with reference to traditional password-based authentication systems and more modern Two Factor Authentication (2FA) systems.

The remainder of this document opens by discussing the problems associated with traditional password-based authentication and drawbacks of the most common 2FA mechanisms (Section 2) followed by a brief exploration of some of the existing and related works in using typing rhythm as an authentication method (Section 3). Section 4 discusses the methodology of the experiment and Section 5 expands on this by discussing the hypotheses, variables and assumptions of the experiment. Section 6 presents the results of the experiment and Section 7 provides a conclusion. Finally the code for the custom keylogger used in this experiment is presented in Appendix A, the scripts used as part of the experiment are presented in Appendix B and any interesting raw data from the experiments presented in Appendix C.

2 PROBLEM STATEMENT

In a computer security context, authentication is the act of verifying a user's identity [1]. It is easy to see why authentication is vital for secure systems. By far, the most common method of authentication is password-based authentication [2] in which a user must supply a username (usually public knowledge) and a password (known only to the user). In a system of this type, the username is used to identify the user, and the password is used to verify that the user is indeed the individual that the username identifies. Unfortunately, password-based authentication suffers from a wide variety of deficiencies, including but not limited to the tendency for users to cycle only a few different passwords [3], the high number of users who write down or forget their password [4] and the vulnerability of short or simple passwords to brute force or dictionary attacks [5].

In response to their perceived weakness, many applications now employ a second authentication method in addition to traditional password-based authentication, and require users to pass both authentication methods in order to gain access. This is known as 2FA and usually manifests as sending

the user a message containing a One Time Passcode (OTP) which they must supply in addition to their password [6]. There are many different mechanisms [7] to send the user the OTP. The most common method is sending the user an SMS containing the OTP. This can be inconvenient and the user must possess a cell phone. Recently, many applications have begun to support the use of a software OTP provider, such as the Microsoft or Google authenticator applications for smartphones. Whilst this may provide some defence against SMS insecurities, it has the same drawbacks from a convenience perspective. A more secure alternative to this is a hardware OTP provider, but these may be even less convenient than a software provider or SMS (if a user doesn't want to carry a phone, they certainly don't want to carry a hardware authenticator). Another possibility is sending the user an OTP or a URL (which acts as a OTP) to the user's email address. This may be more convenient as the user will typically have access to their emails when using a computer or mobile application. However, as discussed previously, many users recycle the same passwords for different applications, and hence if the attacker already has a user's password, they may be able to gain access to their email account, negating the effect of 2FA. A final possibility is the use of a biometric such as fingerprints or facial recognition, but these are typically used as an alternative to passwords rather then in addition to them as they can be inaccurate, if for example the user is wearing gloves or a face covering.

As discussed above, most of the second factor authentication methods in use today are inconvenient to the user, and many of the more secure options require additional hardware. The use of typing rhythm as an authentication method could offer a convenient and inexpensive second factor authentication that does not require any additional hardware and could go largely unnoticed by honest users.

3 BACKGROUND

During World War Two, intelligence staff developed a technique for identifying telegraph operators by the unique typing pattern they used when sending Morse code. This technique was known as "The Fist of the Sender" [8] and found many useful applications in the military intelligence domain.

More recently, security researchers realised that this discovery could be applied to modern keyboards and could be categorised as a form of "soft" or "behavioural" biometrics [9]. Soft biometrics are defined as "physical and behavioural traits, such as gender, height and weight, which are not unique to a specific subject, but are useful for identification, verification, and description of human subjects" [10]. Since soft biometrics are not unique to an individual, they are usually used in combination with some other authentication method as a 2FA system. It is believed that humans use a combination of soft biometrics in order to identify each other [11].

This area of research is known by number of names: keystroke dynamics, keystroke analysis, typing biometrics and typing rhythms [12]. When studying typing rhythm, there are typically two metrics of interest. These are dwell time and flight time. The former is the amount of time that a key is held down for, and the latter is the amount of time in between key presses [13]. Other metrics, such as typing error frequency (how often does the user use backspace), frequency of using certain characters (such as number pad numbers instead of top-row numbers) and how capital letters are generated (which out of shift and the letter is released first or does the user use caps lock) can also be of interest. Typing characteristics can be extracted from structured text such as username/password forms, or unstructured text [13] which is typically gathered by running a general keylogger in the background of a user session.

Much like other biometric systems, when used as part of an authentication system, a typing rhythm component would first need to construct a reference template for each user that a sample could later be matched against [13]. The reference template is built up from a number of features and eventually forms a profile representing a user which is stored in a database. There are many different ways to extract typing features as well as many different comparison methods. The characteristics of a typing rhythm authentication system are determined by the combination of features considered, extraction method, comparison method, data gathering method (structured/unstructured), reference template

density (amount of information contained in the reference template), sample density (amount of information contained in sample to be compared to reference template) and database size (number of users).

In 1997, F. Monrose and A. Rubin [14] constructed a database of 42 keystroke profiles of which 11 were eliminated due to local machine configuration issues invalidating the data. Each profile consisted of an N-Dimensional feature vector where the features are the timing variables of the most common n-grams (such as th, he, nd, re, in, ing…). Reference profiles were constructed from users typing a number of sentences into a dedicated application. These were then organised into clusters based on typing speed in order to speed up the retrieval process. The authors then compared the use of 3 different metrics to compare samples against reference profiles:

- Euclidean Distance between sample and reference.
- Non-Weighted Probability in which a probability is attached to each feature. Higher probabilities are assigned to features closer to the mean, and lower to features further away.
- Weighted Probability in which probabilities are attached to each feature and frequencies are relative to the entire dataset.

Their results showed that Weighted Probability offers the best metric, and also that optimal performance is achieved when comparing two structured samples.

In 2000, S. Cho et al. [15] proposed the use of a neural network as a classifier of typing data. They gathered data from 25 participants asking each to generate a 7 character password. Each participant typed their password 150-400 times over a period of up to a week. The last 75 samples for each user was set aside for training and the rest used for training the neural network. Each sample was stored as a 15-dimensional timing vector consisting of $[D_1, F_1, D_2, F_2, \dots, D_7, F_7, D_E]$, where D_x is the dwell time for the x^{th} character and F_x is the flight time between the x^{th} character and the $x+1^{th}$ character. D_E is the dwell time for the enter key. Note that negative flight times are possible if the user presses the next key before releasing the last. The authors discarded any timing vectors containing a feature in the top 10% to eliminate outliers. A multilayer perception neural network using back-propagation for learning was then trained using this data. They were able to identify users based on their typing rhythm with an average error rate of 1% across their small database. In addition, 15 “Imposters” were given all the passwords, and typed each 5 times. None were able to authenticate successfully. In addition to the processing power required to train neural networks, one major drawback of this approach is that the entire network must be retrained to introduce a new user.

In 2011, J. Deluca et al. [16] gathered a dataset consisting of timing data about each key pressed during a session lasting between 30 minutes and 4 hours for 4 users, with an average of 50,000 keystrokes per user. The data was gathered using the Fimbel keylogger and converted by a custom feature extractor developed by the authors. The features that the authors were interested in are key pressed, time pressed, time released. This data was then used to train a K-Nearest-Neighbour classifier. The authors were able to obtain an average accuracy of 89.49%. The authors admitted that this method was not good enough to reliably authenticate user by itself, however noted that the area showed great promise and warranted further research, in particular repeating the experiment with a more suitable key logger and greater number of participants.

In 2016, S. Sznur and S. García [17] proposed the use of unsupervised clustering algorithms to group typing data by user. As part of their research they generated the largest public labelled keystroke dataset available at the time. This dataset consisted of 379 sessions across 17 unique users gathered by custom keylogging software created by the authors. Each session contained all the keystrokes a user entered whilst using the computer and hence is unstructured data. The dataset was then partitioned into smaller datasets to enable the authors to test different hypothesises. Each hypothesis tested a different distance metric (or combination of metrics). The distance metric was combined with the K-Means clustering algorithm to group the data samples into clusters in the hope that each group would

be a unique user. The authors found that the distance metric “A-distance digraphs + R-distance digraph” offered the best performance and was able to demonstrate an accuracy of 78.9% when applied to all 17 users. Its best accuracy was 98.8% and was achieved when data from only 5 users who all had a similar (and large, >15) number of sessions recorded. R-distance considers the degree of disorder of a vector V with regards to its ordered counterpart V’. R-distance is able to capture the typing rhythm of a user, without capturing their overall speed. A-distance captures the absolute typing speed of a user across an n-graph. Digraphs are groups of 2 letters. The authors concluded that as long as there is sufficient data about each user, and the total number of users is known, it is possible to classify typing data into groups corresponding to a single user with a high degree of accuracy. K, the number of clusters, must be set in advance, and so adding a new user would require the clustering process to be repeated.

4 METHODOLOGY

4.1 KEYLOGGER

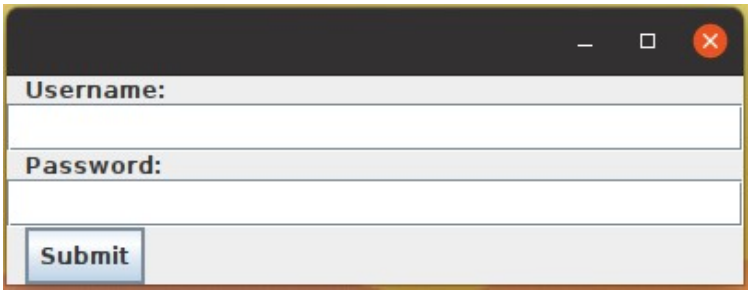

Figure 1: Screenshot of keylogger.

The data for this experiment will be gathered using a purpose built keylogger program. The keylogger is programmed in Java and produces results files, see appendix A for the source code and an example results file. After the starting keylogger, a user is presented with a mock login screen, as shown in figure 1. The login screen consists of a username field, a password field and a submit button. Users enter values into the two fields and then click the submit button (or press the enter key). Once the submit button has been pressed, a results file will be generated. Results files consist of two parts: header and keys. The header contains the complete values for username and password, and the total time elapsed between the first key being pressed and the last key being released. The keys section of the file is in comma separated value (CSV) format where each entry represents one complete key press/release and stores the following information: numerical key code, character represented, clock time the key was pressed, clock time the key was released, time elapsed whilst key was pressed and time elapsed between previous key being released and this key being pressed. The clock times use a JVM clock that is not representative of the real world time these keys were pressed but has a higher resolution than using the system time. As all clock times are relative, the data can be simplified by setting the first clock time that appears in the file to zero and then subtracting the first clock time from every other clock time. The unit for time measurements is nanoseconds (ns). As the data gathered is from a form with set labels, the dataset for this experiment is structured.

4.2 EXPERIMENT DESIGN

Five test subjects were enlisted to provide data for this experiment. Each subject was asked to enter the same login credentials into the keylogger three times, resulting in three data files per test subject and a total of 15 results files. Several days later, the same test subjects were asked to enter the same credentials a further 3 times, resulting in a further 15 results files. All test subjects used the same laptop to operate the keylogger to remove any variance caused by the keyboard.

A sixth test subject was enlisted to play the part of the ‘imposter’. The imposter is given the credentials of all the other test subjects and is asked to type each three times. This generates a further 15 results files. All six subjects were asked to retry any attempt that involved the use of backspace to correct an error, and the credentials were verified to ensure that each subject used identical credentials for each attempt.

All test subjects are friends or family members of the author and will remain anonymous. Due to the privacy concerns of the test subjects, the data gathered from them will not be shared publicly and only timing data will be presented in this document.

The first three samples from each subject will be used to generate a reference profile. Reference profiles will consider the average dwell and flight times on a per keystroke basis, as well as average overall time. For example, the reference profile for the password ‘123’ would store average time 1 is held (across the three samples), average time between 1-2, average time 2 is held, average time between 2-3, average time ‘3’ is held, average time between 1 being pressed and 3 being released (overall time). Both mode and median will be trialled to generate the average values, to investigate the effect this has on performance. A python script will be developed to automate the process of generating the reference profiles and save them in a CSV format (see Appendix B for details).

The final three samples from each subject will be login attempts against the reference profile. Each timing from the attempts will be compared against the reference timing. If a timing falls within a percentage tolerance of the reference timing, that data point will be considered a match. This percentage tolerance will be referred to as timing tolerance. If the number of data points that match reach a threshold, the login attempt will be considered successful. This threshold is referred to as acceptance threshold. Both settings will be varied to investigate the effect on performance.

The samples generated by the imposter are considered to be login attempts against the reference profile of the test subject with the same credentials. The performance of each setting pair can then be evaluated for each test subject in terms of True Positives (authorised user authenticates), False Positives (imposter authenticates), True Negatives (imposter fails to authenticate) and False Negatives (authorised user fails to authenticate). Finally, the performance of the overall system at each tolerance setting can be evaluated by combining all the data from each individual subject and calculating the False Acceptance Rate (FAR) and False Rejection Rate (FRR). A python script will be developed to automate the process of testing each of the login attempts against the corresponding reference profiles at given acceptance threshold and timing tolerance settings and save relevant performance metrics to a CSV file (see Appendix B for details).

5 EXPERIMENT

5.1 HYPOTHESISES

Hypothesis 1 (H1): a simple rhythm can be determined more easily than a more patterned based one. A simple rhythm is defined as 1 or 2 distinct rhythms (such as ta-ta-ta-ta or ta-tum-ta-tum). A pattern is a mix of rhythms with more complex timings. A rhythm is considered easily determined if there is a high False Acceptance Rate.

Hypothesis 2 (H2): incorporating the username rhythm improves performance versus considering the password rhythm alone.

5.2 VARIABLES

Independent variable:

- Test subject and their corresponding credentials

Dependent variables:

- Dwell time per keystrokes
- Flight time in between keystrokes
- Overall time to enter credentials

Control variables:

- Machine and environment used to run keylogger and gather keystroke data (OpenJDK 11.0.11 running on Gigabyte Aero 15, i7-10750H, Ubuntu 20.10 Linux, 16GB RAM)
- Keyboard used to enter samples
- Number of samples per test subject
- Averaging method for reference profiles (tested with mean and median)
- Timing tolerance (tested at 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%)
- Acceptance threshold (tested at 60%, 70%, 80%, 90% and 100%)

5.3 ASSUMPTIONS

- All test subjects correctly enter their credentials on each attempt without using the backspace key
- Rhythm of Tab (used to progress down the form) and Enter key (used to submit the form) are not considered

6 RESULTS

6.1 PARAMETER TUNING

Before the hypothesises can be considered, the optimal values for the parameters needed to be experimentally determined. There are three parameters of interest: averaging method, acceptance threshold and timing tolerance. The optimal parameters were determined by varying the acceptance threshold and timing tolerance and calculating the resultant FAR and FRR for each setting pair across all 30 authorised and imposter (15 of each) login attempts against the reference profiles. This was repeated for each of the averaging methods. The raw results from this can be found in Appendix C. Figures 2, 3, 4 and 5 show 3-dimensional plots of these results. The best FAR and FRR were 3.33% and 0% respectively and were achieved when the averaging mode was median, the acceptance threshold was 60% and the timing tolerance was 40%.

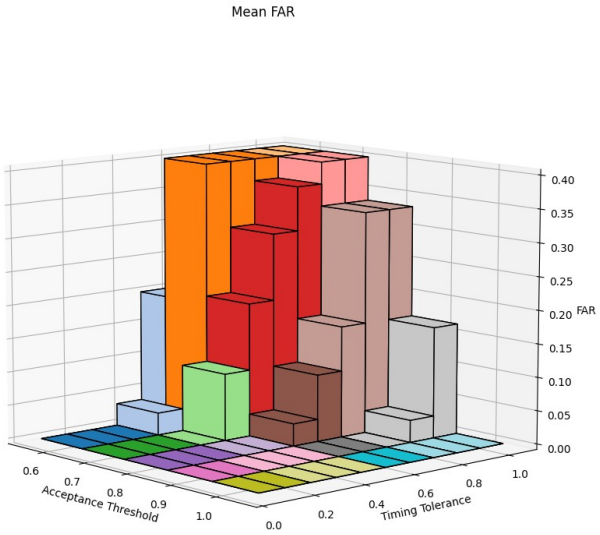

Figure 3: (Acceptance Threshold, Timing Tolerance) vs FAR for Mean averaging.

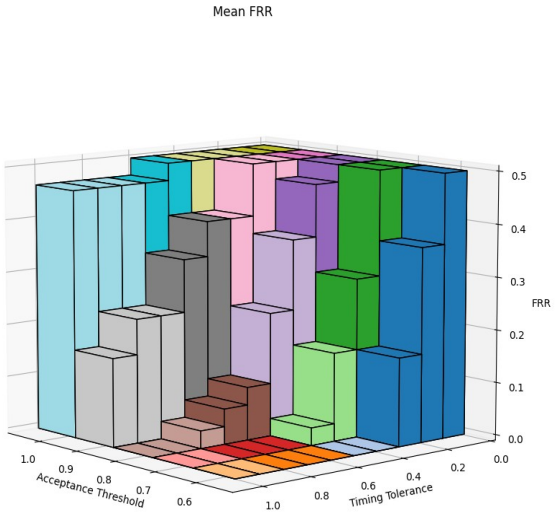

Figure 2: (Acceptance Threshold, Timing Tolerance) vs FRR for Mean averaging.

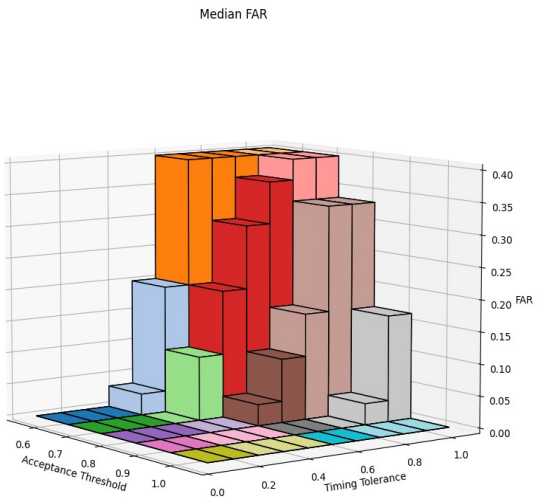

Figure 5: (Acceptance Threshold, Timing Tolerance) vs FAR for Median averaging.

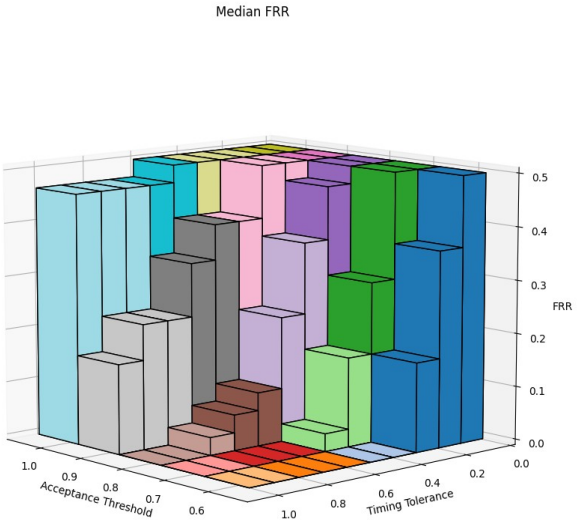

Figure 4: (Acceptance Threshold, Timing Tolerance) vs FRR for Median averaging.

6.2 HYPOTHESIS 1

In order to evaluate H2, the dataset must be split into two partitions. One partition contains simple rhythms and the other more complex rhythms. For the sake of simplicity, when classifying rhythms, only the dwell time is considered, but future works should look at incorporating the flight time into this classification in addition to dwell time. The test subjects will be classified based on their median reference profiles (see Appendix C). The performance will be evaluated based on the FAR when the parameters are set to the values determined in Section 6.1. The dataset is partitioned as shown below:

Simple Rhythms: Subject 1, Subject 2.

Complex Rhythms: Subject 3, Subject 4, Subject 5.

The confusion matrices for the simple rhythms and for the complex rhythms are shown in tables 1 and 2 respectively.

Table 1: Confusion matrix for simple rhythms.

		Predicted	
		Positive	**Negative**
Actual	**Positive**	6	0
	Negative	0	6

Table 2: Confusion matrix for complex rhythms.

		Predicted	
		Positive	**Negative**
Actual	**Positive**	9	0
	Negative	1	8

The corresponding FAR for the simple rhythms is 0% and 5.56% for the complex rhythms. Based on this it would seem that H2 is incorrect, however there is only one false positive result in the entire dataset when the parameters are set to the optimal values, which is against test subject 5. This means that whichever partition of the dataset includes test subject 5 will always perform worse then the other partition. Therefore, the test data used as part of this experiment is not sufficient to definitively prove H1 either way.

6.3 HYPOTHESIS 2

To test H2, a second dataset was generated using a Python script (see Appendix B for details) which removed the keystroke information for the username field from the original dataset. Then, with the parameters set to the optimal values as determined in Section 6.1, confusion matrices across the whole dataset were generated for the original username/password dataset and the password-only dataset. These are presented in tables 3 and 4 respectively.

Table 3: Confusion matrix for username/password dataset.

		Predicted	
		Positive	**Negative**
Actual	**Positive**	15	0
	Negative	1	14

Table 4: Confusion matrix for password-only dataset.

	Predicted

		Positive	Negative
Actual	**Positive**	10	11
	Negative	4	5

The FAR and FRR for the username/password dataset are 3.33% and 0% respectively as calculated in Section 6.1. In comparison, the FAR and FRR for the password-only dataset are 13.33% and 16.67% respectively. However, if the acceptance threshold is set to 90% and the timing tolerance to 100%, the FAR and FRR can be improved to 13.33% and 10% respectively.

By looking at each subject's individual FAR and FRR across the username/password dataset, it is possible to investigate the effect of the username length on performance. Figure 6 is a line graph showing the effect of username length on FAR and FRR, where the parameters are set to the optimal values determined in Section 6.1.

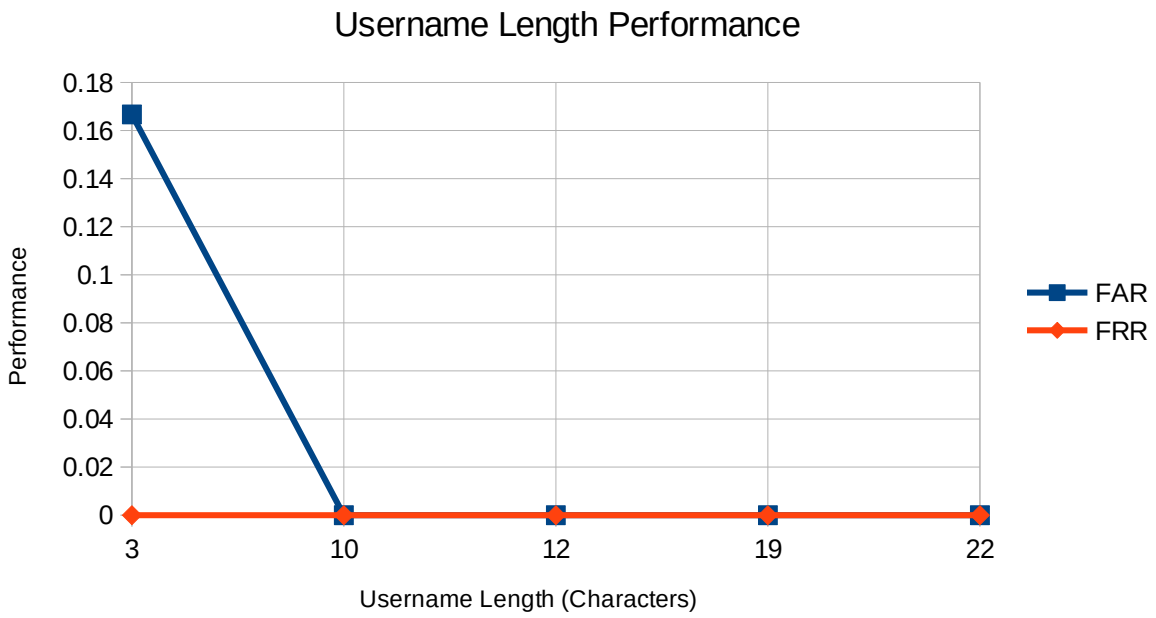

Figure 6: Line graph showing FAR and FRR performance figures against username length.

Based on the results shown above, it would seem that H2 is correct. The FAR and FRR were both dramatically lower for the username/password dataset when compared to the password-only dataset. This is not surprising, as including the username rhythm provides more data points to match against. The test subject who had the shortest username, did have the worst FAR score and was the only subject whom the imposter was able to successfully authenticate against, suggesting that username length does have an effect on performance. However, the size of the trial makes it difficult to draw any conclusive inference and ideally the experiment should be repeated with more test subjects whose username lengths are more evenly distributed.

7 CONCLUSION

The system developed as part of this experiment was able to display relatively good performance once the optimal values for the parameters had been set. For the small number of samples available, the system was able to demonstrate a FAR of 3.33% and a FRR of 0%. Whilst these values are not sufficient to provide a reasonable level of security as a primary authentication mechanism, it is enough

to significantly enhance the security of traditional password protected systems if used as part of a 2FA system.

However, the system presented has a number of deficiencies that should be addressed in future work. The first is the inability for the system to handle backspace or other error corrections. Test subjects were asked to retry any attempts that included an error correction and so the effect of this has not been considered as part of this work. This decision was made because the mechanism the system uses to compare attempts against reference profiles, requires the number of keystrokes to be equal. A second deficiency is that keys held for longer then a certain threshold will generate a repeated event. As a result of this, all three of the imposter's attempts against test subject 4 were immediately rejected as number of keystrokes in the attempt and reference profile were different. This deficiency could be fixed by modifying the keylogger so that it combines duplicate keystroke events together.

Unfortunately, the data gathered as part of this experiment was not sufficient to definitively prove either H1 or H2. In both cases there was not sufficient data to make a conclusive statement either way. This could be alleviated in the future by enlisting significantly more test subjects, and asking each to generate more samples. It could be even further improved by continually recording keystrokes for a fixed period and using this data to generate reference profiles instead of restricting the profile generation to three structured attempts. It may also be interesting to ask all users to use a password of the same length to further control the experiment. For proving H2, it may be useful to assign a username to each user and ensure the username lengths are evenly distributed.

Integrating a typing rhythm into an authentication system would be almost free and should be acceptable to the public as if employed correctly, honest users will not even notice the system, whilst enjoying improved security. Users would require almost no training to utilise a system that incorporated typing rhythm security. Enrolling new users to the system would be straight forward, not requiring an additional process to gather samples as users already need to type as part of most existing enrolment processes. Users could not even be required to repeatedly type the same credentials, as typing rhythms could be established by monitoring keystrokes in the background. Advanced forms of typing rhythms systems could continually verify the identity of the user by monitoring keystroke dynamics in the background which could combat the security issues presented by users forgetting to log out. This could also be useful in high-stress environments such as air traffic control centrers where a typing rhythm system could detect fatigue or distraction through altered typing rhythms.

Typing rhythm is a soft biometric, which means it is not unique to an individual user. This limits is effectiveness as a primary authentication method. A user's typing rhythm could change for a wide variety of reasons including fatigue, injury, using only one hand to type, temperature, intoxication, or improving/declining typing skills, and as a result, a user may be unable to access their system. There are also privacy concerns with this technology, especially if a version that continually monitors keystrokes is employed. Depending on how secure (or ethical) the keylogging system is, the technology could be used to steal sensitive information. Another challenge is the effect that different keyboards have on typing biometrics. Whilst the data from this experiment was all gathered using the same keyboard, this is unrealistic in a real world environment where users may wish to login from various different devices. Test subjects also complained about using an unfamiliar keyboard and many had to practise before being able to enter credentials without error.

REFERENCES

[1] D. Gollman, “Computer Security”, John Wiley and Sons, 1999.

[2] P. Elftmann, “Secure Alternatives to Password-based Authentication Mechanisms”, RWTH Aachen University, 2006.

[3] E. Stobert & R. Biddle, “The Password Life Cycle: User Behaviour in Managing Passwords”, USENIX Association Symposium on Usable Privacy and Security (SOUPS), 2014.

[4] M. Zviran & W.J. Haga, “A comparison of password techniques for multilevel authentication mechanisms”, The Computer Journal, pages 227-237, 1993.

[5] A. Adams & M. A. Sasse, “Users are not the enemy: Why users compromise computer security mechanisms and how to take remedial measures”, Communications of the ACM, pages 41-46, 1999.

[6] T. Petsas, G. Tsirantonakis, E. Athanasopoulos & S. Ioannidis, “Two-factor authentication: is the world ready?: quantifying 2FA adoption”, EuroSec: Proceedings of the Eighth European Workshop on System Security, 2015.

[7] I. Velásquez, A. Caro & A. Rodríguez, “Authentication schemes and methods: A systematic literature review”, Information and Software Technology 94, pages 30-37, 2018.

[8] “Origins - Keystroke Dynamics”, Wikipedia, https://en.wikipedia.org/wiki/Keystroke_dynamics accessed on 08/11/2021

[9] J. Jenkins, Q. Nguyen, J. Reynolds, W. Horner & H. Szu, “ The physiology of keystroke dynamics”, Proc. SPIE 8058, Independent Component Analyses, Wavelets, Neural Networks, Biosystems, and Nanoengineering IX, 80581N, 2011

[10] G. Guo, M. Nixon, A. Ross & M. Vatsa, “Soft Biometrics: Extraction and Applications based on Images and Videos”, Springer Open paper collection.

[11] A. Dantcheva, C. Velardo, A. D’Angelo & J. Dugelay, “Bag of soft biometrics for person identification”, Multimedia Tools and Applications 51, pages 739-777, 2011.

[12] J. Ilonen, “Keystroke dynamics”, Advanced Topics in Information Processing, 2003.

[13] P. Švenda, “Keystroke Dynamics”, Masaryk University, 2001.

[14] F Monrose & A Rubin, “Authentication via keystroke dynamics”, 4th ACM conference on Computer and communication security, 1997.

[15] S. Cho, C. Han, D. H. Han, H. Kim, “Web-Based Keystroke Dynamics Identity Verification Using Neural Network”, Journal of Organizational Computing and Electronic Commerce, 2000.

[16] J. Deluca, D. R. Worley, H. Henry, P. Folkes, N. Bakelman, “A System-Wide Keystroke Biometric System”, Proceedings of Student-Faculty Research Day, CSIS, Pace University, 2011.

[17] S. Sznur & S. García, “Advances in Keystroke Dynamics Techniques to Group User Sessions”, International Journal of Information Security Science, Vol 4, No. 2, 2016.”

APPENDIX A: KEYLOGGER

The keylogger program is written in Java and does not have any external dependencies. It consists of five classes, and produces a new results file with name [current system time].csv each time the submit button is pressed. The directory to store the results should be specified as the first command line parameter to the program (or can be set by changing the DEFAULT_PATH constant in Window.java).

PartialKey.java

```
package RythmicKeylogger;


/**
 * Stores some of the attributes of a key
 * Used for storing partial keys in a buffer
 */
public class PartialKey {
    public int keyCode;
    public long timeDown;

    public PartialKey(int keyCode, long down) {
        this.keyCode = keyCode;
        this.timeDown = down;
    }
}
```

Key.java

```
package RythmicKeylogger;


import java.awt.event.KeyEvent;


/**
 * CS4203 Computer Security - P2: Practical Applications, Part 1: Rhythmic Keylogger for
Authentication.
 * Stores information about a single keypress
 * @author 210027910
 * 14/10/2021
 */
public class Key {
    private int keyCode;
    private long timeDown;
    private long timeUp;
```

```
    private Key previousKey;

    /**
     * Constructor
     * @param character character that was pressed
     * @param down system time (in ns) when key was pressed
     * @param up system time (in ns) when key was released
     * @param previous key that was pressed before this key
     */
    public Key(int keyCode, long down, long up, Key previous) {
        this.keyCode = keyCode;
        this.timeDown = down;
        this.timeUp = up;
        this.previousKey = previous;
    }

    /**
     * Construct from partial key
     * @param key Partial key (contains character and timeUp)
     * @param up system time (in ns) when key was released
     * @param previous key that was pressed before this key
     */
    public Key(PartialKey key, long up, Key previous) {
        this(key.keyCode, key.timeDown, up, previous);
    }

    /**
     * Getter for keyCode
     * @return code of character that was pressed
     */
    public int getKeyCode() {
        return this.keyCode;
    }

    /**
     * Convert keycode into character
     * @return keycode in character form
```

```
     */
    public String getCharacter() {
        return KeyEvent.getKeyText(this.keyCode);
    }

    /**
     * Getter for timeDown
     * @return system time (in ns) key was pressed
     */
    public long getTimeDown() {
        return this.timeDown;
    }

    /**
     * Getter for timeUp
     * @return system time (in ns) key was released
     */
    public long getTimeUp() {
        return this.timeUp;
    }

    /**
     * Getter for previousKey
     * @return key that was pressed before this key
     */
    public Key getPreviousKey() {
        return this.previousKey;
    }

    /**
     * Calculate and return time the key was pressed for (in ns)
     */
    public long timeElapsed() {
        return this.timeUp - this.timeDown;
    }
}
```

Chain.java

```
package RythmicKeylogger;
```

```java
import java.util.List;
import java.util.ArrayList;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

/**
 * CS4203 Computer Security - P2: Practical Applications, Part 1: Rhythmic Keylogger for Authentication.
 * Manages a chain of keypresses.
 * @author 210027910
 * 14/10/2021
 */
public class Chain {
    //key data
    private Key firstKey;
    private Key latestKey;
    private int chainLength;

    //chain data
    private String username;
    private String password;

    /**
     * Constructor.
     */
    public Chain() {
        this.firstKey = null;
        this.latestKey = null;
        this.chainLength = 0;
        this.username = null;
        this.password = null;
    }

    /**
     * Add a new key to the keychain.
     * @param key key to be added
```

```
     */
    public void addKey(Key key) {
        if (key.getPreviousKey() == this.latestKey) {
            if (this.latestKey == null) {
                this.firstKey = key;
            }

            this.latestKey = key;
            chainLength++;
        }
        else {
            throw new IllegalArgumentException("Supplied key is not part of this keychain.
Start a new keychain.");
        }
    }

    /**
     * Getter for chainLength.
     * @return length of the chain in keys
     */
    public int length() {
        return this.chainLength;
    }

    /**
     * Represent the chain as List.
     * @return List form of chain
     */
    public final List<Key> getChain() {
        List<Key> chain = new ArrayList<Key>();
        Key key = this.latestKey;

        while(key != null) {
            chain.add(0, key);
            key = key.getPreviousKey();
        }

        return chain;
    }
```

```java
/**
 * Setter for username.
 * @param username new username
 */
public void setUsername(String username) {
    this.username = username;
}

/**
 * Getter for username.
 * @return username
 */
public String getUsername() {
    return this.username;
}

/**
 * Setter for password.
 * @param password new password
 */
public void setPassword(String password) {
    this.password = password;
}

/**
 * Getter for password.
 * @return password
 */
public String getPassword() {
    return this.password;
}

/**
 * Calculates the total time elapsed to type the chain (in ns).
 * From first key down to last key up.
 * @return total time elapsed to type the chain (in ns)
 */
```

```
    public long timeElapsed() {
        return latestKey.getTimeUp() - firstKey.getTimeDown();
    }

    /**
     * Print a summary of the chain.
     */
    public void printSummary() {
        System.out.println("Username: " + this.getUsername());
        System.out.println("Password: " + this.getPassword());
        System.out.println("Time Elapsed: " + this.timeElapsed() + " ns");

        System.out.println("\nKeys: ");

        for (Key key : this.getChain()) {
            System.out.println("Key Code: " + key.getKeyCode());
            System.out.println("Character: " + key.getCharacter());
            System.out.println("Time held: " + key.timeElapsed() + " ns");
            if (key.getPreviousKey() != null) System.out.println("Time between: " +
Math.abs((key.getTimeDown() - key.getPreviousKey().getTimeUp())) + " ns");
        }
    }

    /**
     * Save chain to disk.
     * KeyLogger.endChain() should be called first
     * @param filepath path to save chain too
     * @throws IOException
     */
    public void save(String path) throws IOException {
        String filepath = path + "chain" + System.currentTimeMillis() + ".csv";
        File file = new File(filepath);
        FileWriter writer = new FileWriter(file);

        writer.write("Username: " + this.getUsername() + "\n");
        writer.write("Password: " + this.getPassword() + "\n");
        writer.write("Total Time Elapsed (ns): " + this.timeElapsed() + "\n");
        writer.write("\nKeys:\n");
```

```
writer.write("key_code,character,time_pressed,time_released,time_elapsed,time_between\n");


        for (Key key : this.getChain()) {
            long timeBetween = 0;
            if (key.getPreviousKey() != null) {
                timeBetween = Math.abs((key.getTimeDown() -
key.getPreviousKey().getTimeUp()));
            }


            String line = key.getKeyCode() + "," + key.getCharacter() + "," +
key.getTimeDown() + "," + key.getTimeUp() + "," + key.timeElapsed() + "," + timeBetween +
"\n";
            writer.write(line);
        }


        writer.close();
    }
}
```

KeyLogger.java

```
package RythmicKeylogger;


import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;


import java.util.ArrayList;


/**
 * CS4203 Computer Security - P2: Practical Applications, Part 1: Rhythmic Keylogger for
Authentication.
 * Intercepts key presses and records them in key classes
 * @author 210027910
 * 14/10/2021
 */
public class KeyLogger extends KeyAdapter {
    private Chain activeChain;
    private Key previousKey;


    private ArrayList<PartialKey> buffer;
```

```
    /**
     * Constructor
     * @param activeChain initially active chain
     */
    public KeyLogger(Chain activeChain) {
        super();
        this.previousKey = null;
        this.activeChain = activeChain;
        this.buffer = new ArrayList<PartialKey>();
    }

    /**
     * Intercepts key press events, records the time they occurred and the character
pressed.
     */
    @Override
    public void keyPressed(KeyEvent event) {
        int keyCode = event.getKeyCode();
        long timeDown = System.nanoTime();
        buffer.add(new PartialKey(keyCode, timeDown));
    }

    /**
     * Intercepts key release events and records the time they occured and add this
information to the active chain
     */
    @Override
    public void keyReleased(KeyEvent event) {
        for (PartialKey partial : this.buffer) {
            if (partial.keyCode == event.getKeyCode()) {
                Key key = new Key(partial, System.nanoTime(), this.previousKey);
                activeChain.addKey(key);

                this.previousKey = key;
                this.buffer.remove(partial);
                break;
            }
        }
```

```
    }

    /**
     * To be called before chaning active buffer or printing.
     * Finishes any partial keys even if they have not been released.
     */
    public void endChain() {
        for (PartialKey partial : this.buffer) {
            Key key = new Key(partial, System.nanoTime(), this.previousKey);
            activeChain.addKey(key);

            this.previousKey = key;
            this.buffer.remove(partial);
            break;
        }
    }

    /**
     * Chain the currently active chain where info is recorded
     * @param newChain new chain to use for data recording
     */
    public void changeActiveChain(Chain newChain) {
        this.activeChain = newChain;
        this.previousKey = null;
        this.buffer = new ArrayList<PartialKey>();
    }
}
```

Window.java

```
package RythmicKeylogger;

import java.awt.Container;
import java.awt.Insets;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.awt.event.ActionEvent;

import javax.swing.*;
```

```java
import java.util.ArrayList;


/**
 * CS4203 Computer Security - P2: Practical Applications, Part 1: Rhythmic Keylogger for Authentication.
 * Main class.
 * @author 210027910
 * 14/10/2021
 */
public class Window {
    private static final boolean VERBOSE = false;
    private static final String DEFAULT_PATH = "/media/jbm/JBM-UNI/CompSciMSc/Security/Assignments/Assignment 2/Part1/results/";
    private static String filepath;


    /**
     * Entry point for program. Contains main method and responsible for creating main GUI and attaching keylogger.
     * @param args Command line arguments
     */
    public static void main(String[] args) {
        //check if filepath has been set
        if (args.length > 0) {
            filepath = args[0];


            //add trailing slash
            if (filepath.charAt(filepath.length() - 1) != '/')  {
                filepath += "/";
            }
        }
        else {
            filepath = DEFAULT_PATH;
        }


        //chain tracking
        ArrayList<Chain> chains = new ArrayList<Chain>();
        chains.add(new Chain());


        //key logger
```

```
KeyLogger logger = new KeyLogger(chains.get(0));

//declare GUI elements
JFrame frame = new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400, 150);

JLabel usernameLabel = new JLabel("Username:");
JLabel passwordLabel = new JLabel("Password:");

JTextField usernameField = new JTextField();
JPasswordField passwordField = new JPasswordField();

JButton submitButton = new JButton("Submit");
submitButton.setMargin(new Insets(5,5,5,5));

//Add elements to frame
Container pane = frame.getContentPane();
pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
pane.add(usernameLabel);
pane.add(usernameField);
pane.add(passwordLabel);
pane.add(passwordField);
pane.add(submitButton);

frame.getRootPane().setDefaultButton(submitButton);

//add logger to textfields
usernameField.addKeyListener(logger);
passwordField.addKeyListener(logger);

//Add listener for button
submitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //record textfield value in chain
        chains.get(chains.size() - 1).setUsername(usernameField.getText());
```

```
                chains.get(chains.size() -
1).setPassword(String.valueOf(passwordField.getPassword()));

                //instruct keylogger to end the chain
                logger.endChain();

                //print summary
                if (VERBOSE) chains.get(chains.size() - 1).printSummary();

                //save chain
                try {
                    chains.get(chains.size() - 1).save(filepath);
                } catch (IOException exception) {
                    exception.printStackTrace();
                }

                //move to new chain
                chains.add(new Chain());
                logger.changeActiveChain(chains.get(chains.size() - 1));

                //clear fields
                usernameField.setText("");
                passwordField.setText("");
            }
        });

        frame.setVisible(true);
    }
}
```

Example results file

Username: example

Password: examplepassword

Total Time Elapsed (ns): 5002049521

Keys:

key_code,character,time_pressed,time_released,time_elapsed,time_between

69,E,1872991105405,1873078345476,87240071,0

88,X,1873257375184,1873401926701,144551517,179029708

65,A,1873334063382,1873503895967,169832585,67863319

77,M,1873534945538,1873603240862,68295324,31049571

80,P,1873743907129,1873781245919,37338790,140666267

76,L,1873851786205,1873921514036,69727831,70540286

69,E,1873958332355,1874090593752,132261397,36818319

69,E,1875008928403,1875145660461,136732058,918334651

88,X,1875286385181,1875450835100,164449919,140724720

65,A,1875382994344,1875522942855,139948511,67840756

77,M,1875695317998,1875768144256,72826258,172375143

80,P,1875867487323,1875936874664,69387341,99343067

76,L,1876010055430,1876077541537,67486107,73180766

69,E,1876111129782,1876216296963,105167181,33588245

80,P,1876286344639,1876353152621,66807982,70047676

65,A,1876426337460,1876529581365,103243905,73184839

83,S,1876495153137,1876597639997,102486860,34428228

83,S,1876632466832,1876700067797,67600965,34826835

87,W,1877079641610,1877209862947,130221337,379573813

79,O,1877274370182,1877341865072,67494890,64507235

82,R,1877411093075,1877481501218,70408143,69228003

68,D,1877585313911,1877687877142,102563231,103812693

10,Enter,1877964885855,1877993154926,28269071,277008713

APPENDIX B: SCRIPTS

strip_usernames.py

This script operates on a copy of the original dataset and iterates through each of the results files stripping username keystroke data from the keys section of the file.

```
#! /bin/python3


#10/11/2021

#CS3203 - Security: Practical 2, Part 1

#Quick and dirty script to remove usernames from results files in order to test hypothesis
2

#Will not work if the subject's username contains any special characters other then space,
full stop and @

#Modify replace_special() to add support for further special characters

#Running this script more then once on any single results file will remove ALL keystrokes


import os


ROOT = "/media/jbm/JBM-UNI/CompSciMSc/Security/Assignments/Assignment
2/Part1/password_only"


def replace_special(text):
    text = text.replace("Space", " ")
    text = text.replace("Period", ".")
    text = text.replace("ShiftQuote", "@")
    text = text.replace("QuoteShift", "@")

    return text


def process_file(path):
    print("Processing " + path)

    replacement = ""
    username = ""

    progress = ""

    i = 0
    f = open(path, 'r')
```

```
    for line in f.readlines():
        if i < 6:
            if "Username: " in line:
                username = line[10:].strip().upper()

            replacement += line
        else:
            if progress != username:
                char = line.split(",")[1]
                progress += char
                progress = replace_special(progress)
            else:
                replacement += line

        i += 1

    f.close()

    f = open(path, 'w')
    f.write(replacement)
    f.close()


def scan_dir(root):
    for filename in os.listdir(root):
        path = os.path.join(root, filename)

        if os.path.isdir(path):
            scan_dir(path)
        elif os.path.isfile(path) and ".csv" in path:
            process_file(path)

if __name__ == "__main__":
    scan_dir(ROOT)
```

reference_profiles.py

Iterates through every directory, starting in the directory specified by ROOT, looking for directories called “reference”. Once it finds one it generates one reference profile for each averaging method using the results files inside the “reference” folder. This is saved as a CSV in the directory one level

above the “reference” directory. It then proceeds to the next “reference” folder until profiles have been generated for every test subject. This script is run against both the username/password dataset and the password-only dataset.

```
#! /bin/python3


#10/11/2021
#CS3203 - Security: Practical 2, Part 1
#Script to generate reference profiles using both mean and median for each subject


import os
import pandas as pd
import statistics
import re


ROOT = "/media/jbm/JBM-UNI/CompSciMSc/Security/Assignments/Assignment 2/Part1/"


def main(root):
    for filename in os.listdir(root):
        path = os.path.join(root, filename)


        if os.path.isdir(path):
            if "subject" in path:
                name = path.rpartition("/")[-1]
                print("Processing " + name + "...")


                subject_path = os.path.join(path, "reference")
                sample_paths = [os.path.join(subject_path, s) for s in
os.listdir(subject_path) if ".csv" in s]


                samples = {}


                for i,sample_path in zip(range(len(sample_paths)), sample_paths):
                    samples[i] = pd.read_csv(sample_path, skiprows=5)


                median_ref = ""
                mean_ref = ""


                end = 0
                for i in range(samples[0].shape[0]):
```

```
                    if samples[0]["character"][i] == "Enter":
                        end = i - 1
                        break

                    dwells = [float(samples[j]["time_elapsed"][i]) for j in samples.keys()]
                    mean_ref += str(statistics.mean(dwells)) + ","
                    median_ref += str(statistics.median(dwells)) + ","

                    flights = [samples[j]["time_between"][i+1] for j in samples.keys()]
                    mean_ref += str(statistics.mean(flights)) + ","
                    median_ref += str(statistics.median(flights)) + ","

                overall_times = [samples[j]["time_released"][end] - samples[j]
["time_pressed"][0] for j in samples.keys()]
                mean_ref += str(statistics.mean(overall_times)) + ","
                median_ref += str(statistics.median(overall_times)) + ","

                mean_file = open(os.path.join(path, name + "_mean_reference_profile.csv"),
'w')
                mean_file.write(mean_ref)
                mean_file.close()

                median_file = open(os.path.join(path, name +
"_median_reference_profile.csv"), 'w')
                median_file.write(median_ref)
                median_file.close()

if __name__ == "__main__":
    main(ROOT + "results")
    main(ROOT + "password_only")
```

test_login.py

Starting in the directory specified by ROOT, finds the folder corresponding to each subject and tests each of the login attempts found in the “attempt” folder against the reference profile generated by the previous script. The script also finds the “imposter” folder and tests each attempt against the relevant test subject. This is done for each combination of the acceptance thresholds and timing tolerances specified in ACCEPTANCE_THRESHOLDS and TIMING_TOLERANCES respectively. These tests are used to generate a confusion matrix for each of the test subjects, and one for the overall system which are saved as CSV files. This script is run against both the username/password dataset and the password-only dataset.

```
#! /bin/python3
```

```
#10/11/2021

#CS3203 - Security: Practical 2, Part 1

#Script to generate confusion matrix results CSVs for each test subject and the overall systems

#runs for each combination of the tolerances specified in TIMING_TOLERANCES and acceptance_thresholdS on each subject


import os

from numpy import mat

from prettytable import PrettyTable

import pandas as pd


ROOT = "/media/jbm/JBM-UNI/CompSciMSc/Security/Assignments/Assignment 2/Part1/"

TIMING_TOLERANCES = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

ACCEPTANCE_THRESHOLDS = [0.6, 0.7, 0.8, 0.9, 1.0]

AVERAGING_METHODS = ["mean", "median"]


def login(reference_profile, login_attempt, timing_tolerance, acceptance_threshold, verbose=False):

    if len(reference_profile) != len(login_attempt):

        print("Reference profile and login attempt are of different length, are you sure the credentials match?")

        return False


    table = PrettyTable()

    table.field_names = ["Point #", "Reference Value", "Attempt Value", "Within Tolerance", "Needed tolerance"]


    result = []


    for i, ref, attempt in zip(range(len(reference_profile)), reference_profile, login_attempt):

        minimum = ref - (ref * timing_tolerance)

        maximum = ref + (ref * timing_tolerance)


        result.append((attempt >= minimum) and (attempt <= maximum))


        needed_tolerance = "-"


        if not result[i]:
```

```
            needed_tolerance = str(abs(ref - attempt)/ref)

        table.add_row([i, ref, attempt, result[i], needed_tolerance])

    match_percentage = result.count(True) / len(result)
    success = match_percentage >= acceptance_threshold

    if verbose:
        print("Timing Tolerance: " + str(timing_tolerance) + ", Acceptance threshold: " +
str (acceptance_threshold))
        print(table)
        print("Matched on " + str(result.count(True)) + " data points.")
        print("Failed to match on " + str(result.count(False)) + " data points.")
        print("Match percentage: " + str(match_percentage))

        if success:
            print ("LOGIN SUCCESSFUL")
        else:
            print("LOGIN FAILED")

    return success

def extract_attempt(results_file):
    attempt = []

    df = pd.read_csv(results_file, skiprows=5)

    end = 0
    for i in range(df.shape[0]):
        if df["character"][i] == "Enter":
            end = i - 1
            break

        attempt.append(float(df["time_elapsed"][i]))
        attempt.append(float(df["time_between"][i+1]))

    attempt.append(float((df["time_released"][end] - df["time_pressed"][0])))

    return attempt
```

```
def load_reference_profile(file_path):
    file = open(file_path, 'r')

    profile = [float(x) for x in file.readline()[:-1].split(",")]

    file.close()

    return profile

def main(root, timing_tolerances, acceptance_thresholds, averaging_methods):
    #initialise system results data structure
    system_results = {}
    for method in averaging_methods:
        system_results[method] = {}

        for acceptance_threshold in acceptance_thresholds:
            for timing_tolerance in timing_tolerances:
                system_results[method][str(acceptance_threshold) + "," +
str(timing_tolerance)] = {
                    "true_positives":0,
                    "false_positives":0,
                    "true_negatives":0,
                    "false_negatives":0,
                }

    for filename in os.listdir(root):
        path = os.path.join(root, filename)

        if os.path.isdir(path):
            if "subject" in path:
                name = path.rpartition("/")[-1]
                print("Processing " + name + "...")

                for method in averaging_methods:
                    profile_name = name + "_" + method + "_reference_profile.csv"
                    profile = load_reference_profile(os.path.join(path, profile_name))
```

```
                    output =
"acceptance_threshold,timing_tolerance,true_positives,false_positives,true_negatives,false_
negatives\n"


                    for acceptance_threshold in acceptance_thresholds:
                        for timing_tolerance in timing_tolerances:
                            true_positives = 0
                            false_positives = 0
                            true_negatives = 0
                            false_negatives = 0

                            attempts_path = os.path.join(path, "attempt")
                            for attempt_name in os.listdir(attempts_path):
                                attempt_file = os.path.join(attempts_path, attempt_name)
                                attempt = extract_attempt(attempt_file)
                                result = login(profile, attempt, timing_tolerance,
acceptance_threshold)

                                if result:
                                    true_positives += 1
                                    system_results[method][str(acceptance_threshold) + ","
+ str(timing_tolerance)]["true_positives"] += 1
                                else:
                                    false_negatives += 1
                                    system_results[method][str(acceptance_threshold) + ","
+ str(timing_tolerance)]["false_negatives"] += 1

                            imposter_path = os.path.join(root, "imposter")
                            for attempt_name in [x for x in os.listdir(imposter_path) if
name in x]:
                                attempt_file = os.path.join(imposter_path, attempt_name)
                                imposter_attempt = extract_attempt(attempt_file)
                                result = login(profile, imposter_attempt, timing_tolerance,
acceptance_threshold)

                                if result:
                                    false_positives += 1
                                    system_results[method][str(acceptance_threshold) + ","
+ str(timing_tolerance)]["false_positives"] += 1
                                else:
                                    true_negatives += 1
```

```
                                    system_results[method][str(acceptance_threshold) + ","
+ str(timing_tolerance)]["true_negatives"] += 1


                            output += (f'{acceptance_threshold},{timing_tolerance},
{true_positives},{false_positives},{true_negatives},{false_negatives}\n')


                    output_file_name = name + "_" + method + "_results.csv"
                    output_file = open(os.path.join(path, output_file_name), 'w')
                    output_file.write(output)
                    output_file.close()


    #unpack system results into csv
    for method in averaging_methods:
        output =
"acceptance_threshold,timing_tolerance,true_positives,false_positives,true_negatives,false_
negatives\n"


        for acceptance_threshold in acceptance_thresholds:
            for timing_tolerance in timing_tolerances:
                key = str(acceptance_threshold) + "," + str(timing_tolerance)


                true_positives = system_results[method][key]["true_positives"]
                false_positives = system_results[method][key]["false_positives"]
                true_negatives = system_results[method][key]["true_negatives"]
                false_negatives = system_results[method][key]["false_negatives"]


                output += (f'{acceptance_threshold},{timing_tolerance},{true_positives},
{false_positives},{true_negatives},{false_negatives}\n')


        output_file_name = method + "_results.csv"
        output_file = open(os.path.join(root, output_file_name), 'w')
        output_file.write(output)
        output_file.close()


if __name__ == "__main__":
    main(ROOT+"results/", TIMING_TOLERANCES, ACCEPTANCE_THRESHOLDS, AVERAGING_METHODS)
    main(ROOT+"password_only/", TIMING_TOLERANCES, ACCEPTANCE_THRESHOLDS,
AVERAGING_METHODS)
```

3d_plot.py

This script is used to generate the 3D plots in Section 6.1. It takes a CSV containing the (acceptance threshold, timing tolerance) vs (FAR, FRR) data as input. A shortened version of file containing the values for the mean averaging method is shown below. A second file was used for the median averaging method.

```
acceptance_threshold,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7….
timing_tolerance,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,0.1….
FAR,0,0,0,0.033333333333333,0.2,0.4,0.4,0.4,0.4,0.4,0,0,0,0,0.1,0.2,0.3,0.366666666666667….
FRR,0.5,0.366666666666667,0.166666666666667,0,0,0,0,0,0,0,0.5,0.5,0.3,0.166666666666667….
```

Running the script produces a 3D bar chart which is shown on the screen and can then be saved or screenshot for inclusion in this document. Changing between FAR and FRR is done by modifying the final variable in the zip() part of the for loop on line 23. The title and labels should then be changed accordingly.

```
#! /bin/python3

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.cm as cm


INPUT = "/media/jbm/JBM-UNI/CompSciMSc/Security/Assignments/Assignment 2/Part1/data/csv/median_param_FRR_FAR.csv"


f = open(INPUT, 'r')

acceptance = np.array([float(x) for x in f.readline().split(",")[1:]])

timing = np.array([float(x) for x in f.readline().split(",")[1:]])

far = np.array([float(x) for x in f.readline().split(",")[1:]])

frr = np.array([float(x) for x in f.readline().split(",")[1:]])

f.close()


fig = plt.figure()

ax = plt.axes(projection = '3d')


colors = cm.tab20(np.linspace(0, 1, len(far)))

width = depth = 0.1


#ax.bar3d(a-width/2., b-depth/2., 0, width, depth, a_score, shade=False, color = colors[i], edgecolor = 'black')


for i, a, t, f in zip(range(len(far)), acceptance, timing, frr):
```

```
    ax.bar3d(a-width/2.0, t-depth/2.0, 0, width, depth, f, shade=False, color=colors[i],
edgecolor='black')


ax.set_title("Mean FRR")

ax.set_xlabel("Acceptance Threshold")

ax.set_ylabel("Timing Tolerance")

ax.set_zlabel("FRR")

plt.show()
```

APPENDIX C: DATA

Table 5: Reference profiles using median averaging method.

	Subject 1		Subject 2		Subject 3		Subject 4		Subject 5	
	Dwell	**Flight**	**Dwell**	**Flight**	**Dwell**	**Flight**	**Dwell**	**Flight**	**Dwell**	**Flight**
Median Reference Profile Data	87.319762	296.301486	128.127093	32.116125	130.066911	117.272092	115.931804	63.939369	93.482401	32.888933
	120.452321	432.444327	88.241046	176.160337	102.260184	50.105983	103.142829	26.418256	103.793386	16.10361
	103.163868	230.267471	135.780702	100.623727	84.343917	129.091262	166.102788	61.332785	70.865437	516.133139
	105.868016	256.495245	87.574209	264.626286	77.711491	217.488863	175.619435	23.682952	80.71732	240.629117
	135.211538	376.602666	63.012835	544.769763	127.339027	397.609857	79.942441	95.588856	241.65522	56.048214
	111.354677	313.062072	73.021376	191.402444	84.020955	256.947331	87.893097	56.277664	87.501376	16.565829
	102.685828	313.218182	60.122143	155.455404	88.015429	153.377589	87.645943	136.771059	57.134271	76.196594
	77.085829	410.713411	56.009126	128.023107	62.826204	100.179842	71.88884	148.48364	63.203052	76.600043
	127.326495	312.531788	87.987776	136.640787	87.729164	184.654168	103.901106	39.543767	65.988807	167.677451
	127.453029	2556.817264	87.1531	88.098544	52.429236	1152.214633	80.036176	160.103027	87.531623	239.556451
	107.505158	1120.121417	63.562018	1816.203297	72.109743	129.285149	77.201076	179.526623	264.019696	176.017975
	1332.704428	743.658696	69.145342	903.118681	104.996458	143.964705	79.063601	772.531121	136.48999	70.26302
	103.709186	768.42769	917.798935	282.254325	71.16879	152.939814	87.720639	257.093798	89.045796	128.968614
	88.051638	240.417625	68.564856	123.690719	87.923546	96.475921	79.639067	680.416244	0	0
	124.971657	424.909625	63.601773	210.948598	102.722219	520.724258	792.742008	199.535656	0	0
	103.265294	560.544829	48.142607	437.555408	87.867479	136.207426	71.922984	337.19082	0	0
	111.159016	208.272749	71.514858	240.223534	94.908559	161.111739	73.138564	304.110468	0	0
	76.804398	387.328049	79.302494	168.705303	94.794349	153.303151	158.640021	47.95566	0	0
	127.566959	824.204232	68.426092	184.093426	111.055085	160.920844	65.795532	71.931754	0	0
	103.929867	749.139501	71.556512	1832.511807	71.385188	1552.514112	87.271967	234.013086	0	0
	111.664665	181.549374	96.488573	63.417893	0	0	94.922674	417.202351	0	0
	104.072044	377.313894	54.766791	121.219005	0	0	0	0	0	0
	111.402852	4469.359964	86.798768	137.219228	0	0	0	0	0	0
	136.601245	441.383328	62.651089	47.849927	0	0	0	0	0	0
	488.647665	336.365132	63.382756	304.365987	0	0	0	0	0	0
	119.273093	215.252019	110.856017	22.897231	0	0	0	0	0	0
	102.901964	248.8454	80.274157	824.855008	0	0	0	0	0	0
	88.326394	366.485754	0	0	0	0	0	0	0	0
	121.728831	134.298732	0	0	0	0	0	0	0	0
	73.144012	328.232848	0	0	0	0	0	0	0	0
	70.180688	137.315618	0	0	0	0	0	0	0	0
	65.853062	214.647776	0	0	0	0	0	0	0	0
	143.269489	1752.023028	0	0	0	0	0	0	0	0
Overall Time:	1752.023028		824.855008		1552.514112		417.202351		128.968614	

Table 6: Parameter tuning data for mean averaging method.

acceptance_threshold	timing_tolerance	true_positives	false_positives	true_negatives	false_negatives	FAR	FRR
0.6	0.1	0	0	15	15	0.00%	50.00%
0.6	0.2	2	0	15	13	0.00%	43.33%
0.6	0.3	11	0	15	4	0.00%	13.33%
0.6	0.4	12	3	12	3	10.00%	10.00%
0.6	0.5	14	8	7	1	26.67%	3.33%
0.6	0.6	15	12	3	0	40.00%	0.00%
0.6	0.7	15	12	3	0	40.00%	0.00%
0.6	0.8	15	12	3	0	40.00%	0.00%
0.6	0.9	15	12	3	0	40.00%	0.00%
0.6	1	15	12	3	0	40.00%	0.00%
0.7	0.1	0	0	15	15	0.00%	50.00%
0.7	0.2	0	0	15	15	0.00%	50.00%
0.7	0.3	5	0	15	10	0.00%	33.33%
0.7	0.4	10	1	14	5	3.33%	16.67%
0.7	0.5	13	5	10	2	16.67%	6.67%
0.7	0.6	15	7	8	0	23.33%	0.00%
0.7	0.7	15	10	5	0	33.33%	0.00%
0.7	0.8	15	12	3	0	40.00%	0.00%
0.7	0.9	15	12	3	0	40.00%	0.00%
0.7	1	15	12	3	0	40.00%	0.00%
0.8	0.1	0	0	15	15	0.00%	50.00%
0.8	0.2	0	0	15	15	0.00%	50.00%
0.8	0.3	1	0	15	14	0.00%	46.67%
0.8	0.4	4	0	15	11	0.00%	36.67%
0.8	0.5	8	0	15	7	0.00%	23.33%
0.8	0.6	12	2	13	3	6.67%	10.00%
0.8	0.7	13	6	9	2	20.00%	6.67%
0.8	0.8	15	7	8	0	23.33%	0.00%
0.8	0.9	15	10	5	0	33.33%	0.00%
0.8	1	15	10	5	0	33.33%	0.00%
0.9	0.1	0	0	15	15	0.00%	50.00%
0.9	0.2	0	0	15	15	0.00%	50.00%
0.9	0.3	0	0	15	15	0.00%	50.00%
0.9	0.4	0	0	15	15	0.00%	50.00%
0.9	0.5	3	0	15	12	0.00%	40.00%
0.9	0.6	3	0	15	12	0.00%	40.00%
0.9	0.7	5	0	15	10	0.00%	33.33%
0.9	0.8	7	2	13	8	6.67%	26.67%
0.9	0.9	10	7	8	5	23.33%	16.67%
0.9	1	12	7	8	3	23.33%	10.00%
1	0.1	0	0	15	15	0.00%	50.00%
1	0.2	0	0	15	15	0.00%	50.00%
1	0.3	0	0	15	15	0.00%	50.00%
1	0.4	0	0	15	15	0.00%	50.00%
1	0.5	0	0	15	15	0.00%	50.00%
1	0.6	0	0	15	15	0.00%	50.00%
1	0.7	1	0	15	14	0.00%	46.67%
1	0.8	1	0	15	14	0.00%	46.67%
1	0.9	2	0	15	13	0.00%	43.33%
1	1	3	0	15	12	0.00%	40.00%

Table 7 Parameter tuning data for median averaging method. Optimal parameters highlighted.

acceptance_threshold	timing_tolerance	true_positives	false_positives	true_negatives	false_negatives	FAR	FRR
0.6	0.1	0	0	15	15	0.00%	50.00%
0.6	0.2	4	0	15	11	0.00%	36.67%
0.6	0.3	10	0	15	5	0.00%	16.67%
0.6	0.4	15	1	14	0	3.33%	0.00%
0.6	0.5	15	6	9	0	20.00%	0.00%
0.6	0.6	15	12	3	0	40.00%	0.00%
0.6	0.7	15	12	3	0	40.00%	0.00%
0.6	0.8	15	12	3	0	40.00%	0.00%
0.6	0.9	15	12	3	0	40.00%	0.00%
0.6	1	15	12	3	0	40.00%	0.00%
0.7	0.1	0	0	15	15	0.00%	50.00%
0.7	0.2	0	0	15	15	0.00%	50.00%
0.7	0.3	6	0	15	9	0.00%	30.00%
0.7	0.4	10	0	15	5	0.00%	16.67%
0.7	0.5	14	3	12	1	10.00%	3.33%
0.7	0.6	15	6	9	0	20.00%	0.00%
0.7	0.7	15	9	6	0	30.00%	0.00%
0.7	0.8	15	11	4	0	36.67%	0.00%
0.7	0.9	15	12	3	0	40.00%	0.00%
0.7	1	15	12	3	0	40.00%	0.00%
0.8	0.1	0	0	15	15	0.00%	50.00%
0.8	0.2	0	0	15	15	0.00%	50.00%
0.8	0.3	1	0	15	14	0.00%	46.67%
0.8	0.4	4	0	15	11	0.00%	36.67%
0.8	0.5	8	0	15	7	0.00%	23.33%
0.8	0.6	12	1	14	3	3.33%	10.00%
0.8	0.7	13	3	12	2	10.00%	6.67%
0.8	0.8	14	5	10	1	16.67%	3.33%
0.8	0.9	15	10	5	0	33.33%	0.00%
0.8	1	15	10	5	0	33.33%	0.00%
0.9	0.1	0	0	15	15	0.00%	50.00%
0.9	0.2	0	0	15	15	0.00%	50.00%
0.9	0.3	0	0	15	15	0.00%	50.00%
0.9	0.4	0	0	15	15	0.00%	50.00%
0.9	0.5	3	0	15	12	0.00%	40.00%
0.9	0.6	3	0	15	12	0.00%	40.00%
0.9	0.7	5	0	15	10	0.00%	33.33%
0.9	0.8	8	0	15	7	0.00%	23.33%
0.9	0.9	8	1	14	7	3.33%	23.33%
0.9	1	10	5	10	5	16.67%	16.67%
1	0.1	0	0	15	15	0.00%	50.00%
1	0.2	0	0	15	15	0.00%	50.00%
1	0.3	0	0	15	15	0.00%	50.00%
1	0.4	0	0	15	15	0.00%	50.00%
1	0.5	0	0	15	15	0.00%	50.00%
1	0.6	0	0	15	15	0.00%	50.00%
1	0.7	1	0	15	14	0.00%	46.67%
1	0.8	1	0	15	14	0.00%	46.67%
1	0.9	1	0	15	14	0.00%	46.67%
1	1	1	0	15	14	0.00%	46.67%

Table 8: Parameter tuning data for median averaging method on password-only dataset

acceptance_tolerance	timing_tolerance	true_positives	false_positives	true_negatives	false_negatives	FAR	FRR
0.6	0.1	0	0	15	15	0.00%	50.00%
0.6	0.2	7	0	15	8	0.00%	26.67%
0.6	0.3	9	1	14	6	3.33%	20.00%
0.6	0.4	10	4	11	5	13.33%	16.67%
0.6	0.5	14	7	8	1	23.33%	3.33%
0.6	0.6	15	9	6	0	30.00%	0.00%
0.6	0.7	15	9	6	0	30.00%	0.00%
0.6	0.8	15	11	4	0	36.67%	0.00%
0.6	0.9	15	12	3	0	40.00%	0.00%
0.6	1	15	12	3	0	40.00%	0.00%
0.7	0.1	0	0	15	15	0.00%	50.00%
0.7	0.2	2	0	15	13	0.00%	43.33%
0.7	0.3	6	0	15	9	0.00%	30.00%
0.7	0.4	8	0	15	7	0.00%	23.33%
0.7	0.5	8	2	13	7	6.67%	23.33%
0.7	0.6	14	3	12	1	10.00%	3.33%
0.7	0.7	15	7	8	0	23.33%	0.00%
0.7	0.8	15	9	6	0	30.00%	0.00%
0.7	0.9	15	9	6	0	30.00%	0.00%
0.7	1	15	9	6	0	30.00%	0.00%
0.8	0.1	0	0	15	15	0.00%	50.00%
0.8	0.2	0	0	15	15	0.00%	50.00%
0.8	0.3	3	0	15	12	0.00%	40.00%
0.8	0.4	6	0	15	9	0.00%	30.00%
0.8	0.5	7	0	15	8	0.00%	26.67%
0.8	0.6	7	1	14	8	3.33%	26.67%
0.8	0.7	10	2	13	5	6.67%	16.67%
0.8	0.8	11	4	11	4	13.33%	13.33%
0.8	0.9	12	8	7	3	26.67%	10.00%
0.8	1	12	9	6	3	30.00%	10.00%
0.9	0.1	0	0	15	15	0.00%	50.00%
0.9	0.2	0	0	15	15	0.00%	50.00%
0.9	0.3	0	0	15	15	0.00%	50.00%
0.9	0.4	1	0	15	14	0.00%	46.67%
0.9	0.5	2	0	15	13	0.00%	43.33%
0.9	0.6	4	0	15	11	0.00%	36.67%
0.9	0.7	5	0	15	10	0.00%	33.33%
0.9	0.8	9	1	14	6	3.33%	20.00%
0.9	0.9	10	3	12	5	10.00%	16.67%
0.9	1	12	4	11	3	13.33%	10.00%
1	0.1	0	0	15	15	0.00%	50.00%
1	0.2	0	0	15	15	0.00%	50.00%
1	0.3	0	0	15	15	0.00%	50.00%
1	0.4	0	0	15	15	0.00%	50.00%
1	0.5	0	0	15	15	0.00%	50.00%
1	0.6	1	0	15	14	0.00%	46.67%
1	0.7	1	0	15	14	0.00%	46.67%
1	0.8	2	0	15	13	0.00%	43.33%
1	0.9	2	0	15	13	0.00%	43.33%
1	1	3	0	15	12	0.00%	40.00%

Table 9: Username length data sorted according to username length in ascending order.

Subject #	Username Length	true_positives	false_positives	true_negatives	false_negatives	FAR	FRR
5	3	3	1	2	0	0.1666666667	0
3	10	3	0	3	0	0	0
4	12	3	0	3	0	0	0
2	19	3	0	3	0	0	0
1	22	3	0	3	0	0	0