Can distributed two-factor authentication provide security benefits over traditional methods?

Supervisor: <redacted>

Abstract

This report aims to assess the feasibility and security of a distributed approach to multi-factor authentication. The main issue facing current multi-factor authentication solutions is that they can be inconvenient to use as they require manual entry of a code by the user, or for the user to sign in to an email account to click an authentication link. One additional concern that this project aims to address, is the issue of storing sensitive user authentication data in a single database. Storing this data in a single database turns this database into a 'holy-grail' for attackers looking to exfiltrate data for nefarious purposes. This project, and the accompanying program, look at the potential in distributing security data across MySQL databases and a USB storage device. This approach will be assessed for its security and ease of use for the end user, as well as a comparison with other methods of multi-factor authentication. The accompanying software was built in Python 3.7 and is available on GitHub through the link found in Appendices 3.

Table of Contents

Introduction

This report aims to assess the feasibility of a distributed approach to multi-factor authentication security data storage. Multi-factor authentication adds an additional layer of security to the traditional password and username combination, or single-factor authentication. A basic example of multi-factor authentication would be a four-digit code that is sent to the user attempting to sign-in via SMS, which they then must enter after correctly entering their password. This would be considered 2FA and can be found on websites and applications that have web-based services, for example the discord app (Discord Inc., 2019). The issue with these systems is that the security codes can be bypassed a man-in-the-browser attack with tools such as evilnginx2 (kgretzky, 2018) by exploiting a vulnerability in the 'trusted device' feature which was originally intended to increase usability. Secondly, there have been examples of security data exfiltration by malicious actors against large scale organisations such as the adobe hack of 2013 (Hern, 2013). This data leak, and others like it, were partly due to the data being stored in an insecure manner in a single database, turning the user account security database into a 'holy grail' for attackers to exfiltrate data from.

This project aims to explore these two main issues with user account security, to try and find resolutions these issues whilst maintaining usability for non-technical users and keeping security as a priority as well. This report will look at several multi-factor authentication solutions and compare their performance with the distributed login concept to assess its feasibility and integrity as a security solution. The concept will attempt to solve the 'single database' security problem by splitting the multi-factor authentication data across multiple MySQL databases and a USB device that the user takes with them. This device also adds the benefit of increasing the factors of authentication, in that the device itself is required to perform the login sequence as it contains the critical part of the authentication key that in turn can be increasingly complex as the user does not need to enter the data manually. The project also looks at an implementation of groups as an authentication factor; this simply asks the user which 'group' they last signed out from. This adds an additional factor of authentication, as the user will be the only person who knows which group they last used, and therefore the system will be able to refer to the database that corresponds with that group to obtain the correct security data for authentication. As both the USB device and the group are a required step in the login process, and the project treats every device as 'untrusted' when a sign in request is made, this should mitigate the exploitation of the 'trusted device' feature that evilnginx2 (kgretzky, 2018) exploits.

Literature Review

Single factor authentication using passwords has inherent security flaws form multiple perspectives. Breaches such as the adobe password database hack in 2013 as covered by Ducklin in their article (DUCKLIN, 2013), have taught the security community that centralising security credentials can have disastrous consequences, even if the credentials are encrypted. Colnago et al. elaborate; "*Password breaches, either due to the large number of password database leaks or to increasingly sophisticated (and possibly targeted) phishing attacks, seriously increase the risk of authentication credential compromise. Worse, these risks are further compounded by poor user password hygiene, such as creating easily discoverable passwords or reusing them across multiple accounts*" (COLNAGO, et al., 2018). The need for a second method of authentication has, since breaches such as the aforementioned adobe hack, been in increasing demand by large organisations in order to protect their users. Due to this demand, development of multi-factor authentication tools has become increasingly pertinent for companies such as Google (Google LLC, n.d.) and Microsoft (Microsoft Corporation, 2018) as a way of authenticating users with something more than just their password. Some companies, such as yahoo have even replaced passwords with a push notification on the

user's mobile device when they log in. (Yahoo, n.d.) However, adoption of multi-factor authentication technologies is still alarmingly low, a study performed by Ackerman et al. reveals that adoption of two-factor authentication (2FA) by end users is low.

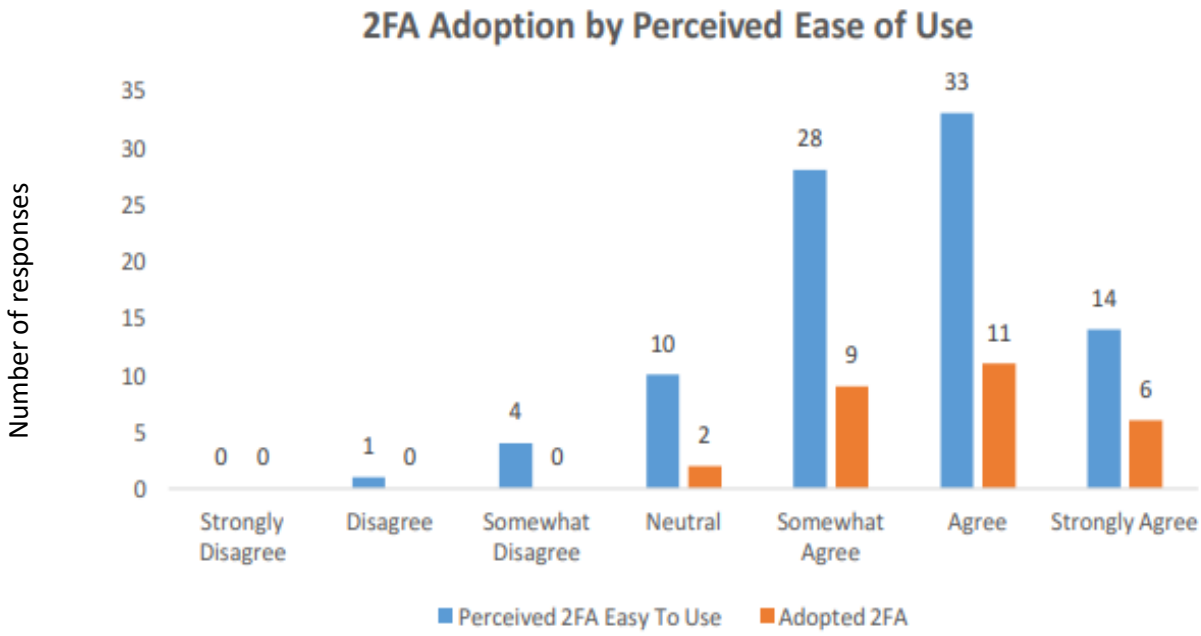

Figure 1: Comparison of Responses to "2FA service (such as SMS/text, email, Google Authenticator), are easy to use" with Adoption of 2FA Services (ACKERMAN & HUMBERTO, 2014)

A figure of particular interest in Figure 1 is the number of users who adopted 2FA a week after taking the study; only 13% of users that were indifferent or opposed to 2FA adopted it. This may be due to users stating that 2FA systems are not easy to use, but still, there is a vast majority of users in the survey who did not adopt it, those who were in favour of 2FA had a much higher adoption rate. This suggests that traditional key entry 2FA systems are not adopted due to the extra processes needed to login, whether that is entering a generated code on an app or text message or clicking a link in an email.

With these notions in mind, this project aims to reduce the burden of entering authentication information to the user by automating the key validation process via a USB key. This project presents a novel idea of combining the simplicity of physical hardware tokens, with the security of complex hash fixed length keys. The benefit of this is that it is easier, as the user simply needs to plug a USB device into their machine and the authentication process is automated from there**.** A study conducted by Reynolds et al. looked into the deployment of YubiKey, a hardware USB 2FA device designed for authenticating login to online account such as Google or Microsoft Windows. One participant in the study was quoted as saying *"I prefer having the YubiKey because I don't have to worry about typing in a number, I can just press it really quick."* (Reynolds, et al., 2018). This project aims to appeal to users who may also enjoy this preference, whilst also combatting the tedious set up for end users that products such as YubiKey have, Reynolds et al. touch on this in their study; *"Four participants (4; 12%) also emphasized that while the initial setup was difficult, it was easy to use after that"* (Reynolds, et al., 2018). More participants thought that the YubiKey was useful when it was set up for them in the study performed by Reynolds et al. which this project aims to provide. Although this is a very small sample size, this does give encouragement to the fact that end users would benefit from a hardware authentication device from this project.

A common exploit against 2FA systems is a Man in the Browser (MitB) attack. A MItB attack will tamper with user data as it is submitted to the system they are logging in to in order to steal the trusted token that would be stored in the users browser on the machine they are using, Krishnan et al. elaborate; "*[A] User enters OTP and leaves the trusted device option unchecked (to emulate an untrusted computer) […] Trojan intercepts the submit OTP request and adds a new POST parameter to programmatically enable the option "Don't ask for codes again on this computer". This is a typical MitB attack. User will never know that the device has been added as a trusted device*" (KRISHNAN & JUMAR, 2014) this allows malware such as evilnginx2 (kgretzky, 2018) to log in to the users account as a trusted device unbeknownst to the user unless they are vigilant and check their trusted devices, which a majority of users do not. This project aims to combat this by using the hardware USB stick as the trusted device, and if a session is stolen by an attack similar to MitB the key is changed when the user logs out, to avoid extended abuse of their session token after they log out. A secondary benefit of the use of a USB device is that, like YubiKey, it removes the requirement of the user to enter a passcode or remember a number. This removes the risk that the user may enter the incorrect code and lock themselves out, whilst also making the security more usable for less security savvy users.

Another feature that this project looks at is distributed storage of credential data. Shirvanian et al. discuss the distribution of passwords when exploring further features of their SPHINX password manager; *"SPHINX can also benefit from a distributed service in which the [security] keys for web accounts is distributed among several servers. This prevents learning specific keys upon the compromise of one server or a subset of servers."* (Shirvanian, et al., 2017) This benefit of distribution is something that this project is going to utilise, however unlike its proposed implementation in Shirvanian et al.'s password manager, this project will be distributing the 2FA data across two SQL databases, and a USB storage device of any size that the user carries with them.

One other interesting point the Shirvanian et al. bring up is the *"notion of outputting decoy passwords to an attacker who compromises the manager and attempts to decrypt the passwords with a wrong master password. Since the attacker is not aware of the correct password, any attempt to login with the decoy passwords can be prevented and raise an alert."* (Shirvanian, et al., 2017) This strategy could also be applied to the authentication keys that are generated in this project, as this would allow the organisation to further learn about where an attacker is trying to reach, or what actions they are trying to perform against the organisation. One alternative product to SPHINX that Shirvanian et al. reference is PwdHash, a proposed password manager that specifically discusses the verification of the web page that is being used, so that phishing sites are blocked from stealing user password data. This project could implement this current site domain verification to legitimise the login portal to the USB device. As Ross et al. explain, *"A natural choice is to use the domain of the page (or frame) where the password field is located. This a password field at a phishing site will be hashed with the phishing domain name, while a password field on a legitimate site will be hashed appropriately."* (Ross, et al., 2005) The stored key can be salted using the legitimate login portals site domain, as implemented in PwdHash. When the USB device receives a request from a site to provide the key chunk, it first uses the current site domain to decrypt the encrypted key chunk and check that it creates the same hash signature as previously recorded from the previous session logout. Both aforementioned features could be implemented as part of further research but are firmly out of scope for this project due to time constraints.

For storage of the authentication credentials, a series of distributed MySQL (Oracle Corporation, 2019) databases distributed as closely or as widely as the implementor wishes to deploy them. Being a relational database system, it beat out document-based solutions such as MongoDB due to it being

better suited to data where integrity is of paramount importance, which in this project is a priority. Deari et al. allude to this strength in their comparison between MySQL (Oracle Corporation, 2019) and MongoDB; *"If data integrity is a major concern, than MySQL would be the choice as it implements ACID transactions. Although it is announced that the latest versions of MongoDB will implement a sort of ACID, in general to achieve better scalability and performance, they have weaker concurrency model implemented known as BASE."* (DEARI, et al., 2018). The key feature that strengthens MySQL's data integrity is its compliance with the ACID standard. The four pillars of this standard are Atomicity, Consistency, Isolation and Durability, with this project specifically benefitting from Consistency that ensures that only valid data is written to the database, Durability that ensures that all transactions to the database are not lost, and Isolation which protects each transaction from others to stop transactional collisions.

Project features

This project uses SHA-256, a member of the SHA-1 family of hashing algorithms, to generate authentication keys and also for verification of the correct key being entered. This was selected due to SHA-1 and variants having a good reputation for security applications. Wang et al. performed a comparison of SHA-1 versus MD5, a strong alternative to SHA-1, and their findings were favourable to SHA-1; *"the SHA-1's CR value is lower than MD5, which means that SHA-1 has higher security then MD5."* (Wang & Cao, 2013). One consideration when selecting a hashing algorithm is their collision rate, whereby if the output of hashing algorithm from two different input strings is the same a hash collision occurs. An additional consideration when looking at different hashing algorithms is the potential for it to be cracked, Wang et al. address this when comparing MD5 and SHA-1; *"The MD5 algorithm has cracking method, while the SHA-1 is only theoretical cracked, so the MD5 is more vulnerable"* (Wang & Cao, 2013). This purely theoretical crack that Wang et al. mention, gives further merit to SHA-256 as the hashing algorithm of choice for this project as it means that if a malicious actor were to compromise one of the databases that holds one of the key chunks, they would still have to crack the other 75% using only a theoretical crack.

Additionally, this project will utilise hash salting further increase the uniqueness of the generated keys from the SHA-256 algorithm. The salt's used will be in the pattern '<username&&password&&datestamp>', where password is the users regular password, '&&' means that the three pieces of data are appended to each other, and datestamp is the combined date and time whenever a sign in/out action is performed. However, even with salting, SHA-256 still pertains vulnerability to a length extension weakness, Gauravaram et al. explain; *"Hash functions that are designed using Merkle-Damgard framework are vulnerable to "length extension weakness" where it is possible to compute a new hash value from a known hash value and the length of the message from which this previous hash value was computed without knowing the message"* (GAURAVARAM, 2012). Given this projects implementation of SHA-256, which is designed using the Merkle-Damgard framework, there is a concern for the strength of the values it produces. On that same note however, Gauravaram also suggests a solution for this weakness, *"Proposing a password security policy which sets limits on both the minimum and maximum lengths of the user passwords such that after padding the password, the length of the padded password is equivalent to the size of one block of the compression function. This countermeasure would be effective for the computer systems and networks that use SHA-1, SHA-256 and SHA-512 hash functions for salt||password hashing."* (GAURAVARAM, 2012). This enforcement will be implemented in the code of this project to make sure that this weakness is mitigated.

For version control, this project will use Git and the hosting site GitHub (GitHub, Inc., 2018) the link for this project can be seen in Appendices 3. This will ensure that versions are kept consistent across

multiple development machines, by making use of the pull and push commands of Git and the integration of Git into PyCharm (JetBrains s.r.o., 2018) also increases the benefit of using Git and GitHub. Git is widely regarded as one of the best version control systems available for software development, Henry Van Styn touches on Git's reputation in the opening line of his article; *"[Git] is extremely flexible and guarantees data integrity while being powerful, fast and efficient."* (Van Styn, 2011) Due to this popularity amongst professionals in the development community, this project will utilise Git to maintain and manage the code. Other version control systems were considered, such as Subversion (Apache Software Foundation, 2018), as it has some useful features such as autoversioning which for a project like this with a single developer working on it, is useful as "*For one, a new version is created every time a file is saved. That way, you have a complete save history of your files without worrying about whether you've done a commit recently."* (Nagel, 2006) However, this project development requirements required a more manual approach to version control, so Git was the clear choice in this context.

When testing the application being developed in this project, there are several different testing methods that can be used for a Python program, Orso et al. discuss software testing in their paper through identifying four key areas: "*(1) automated test input generation, (2) testing strategies, (3) regression testing, and (4) support for empirical studies*" (Orso & Rothermel, 2014). Whilst Orso et al. do not claim to have done a comprehensive survey of the entire software development industry, and all of the testing methodologies used throughout, their sample was from their colleagues involved in automated test integration. During their investigation, Orso et al. identify JUnit for Java program testing as a popular tool used by their colleagues, and also acknowledge "*similar frameworks for other languages referred to collectively as xUnit*" (Orso & Rothermel, 2014). There are a number of xUnit frameworks available for many of the popular scripting/object-oriented languages, including one for Python, the language that this project uses. The xUnit framework for Python is available on the official Python Wiki as "*PyUnit*" (Purcell, 2017), and due to the relatively high regard that Orso et al. and their colleagues regard the xUnit series of testing frameworks this project will be testing using a similar unit testing framework that is built in to the PyCharm IDE (JetBrains s.r.o., 2018) that the project is being developed in. There are, of course, limitations to any and all testing, as Orso et al. state; "*In general, the characteristics of modern systems can render them extremely problematic for existing testing approaches. Heterogeneity, rich environments, and high configurability make it difficult to model a system in its entirety*" (Orso & Rothermel, 2014). This concern applies in particular to this project as it is designed to be an open-ended implementation, and therefore the variability seen in the surrounding applications and languages that the project would be integrated with must be considered when assessing the validity of testing with unit testing.

Functionality

The functional structure of this project is illustrated in Diagrams 1 and 2. The main feature of this project is the distribution of the generated key across multiple MySQL (Oracle Corporation, 2019) databases and a USB storage device. Diagram 1 illustrates how the sign in procedure is carried out. The user inserts the USB storage device that contains the chunk of the key that we last generated when they signed out of the previous group they were using. They enter their username and password as normal, and additionally state which group they last signed out from. With this information, the system will then grab the other three pieces of the fragmented key from the respective databases and compile them in the order they were broken up form, using the chunk stored on the users USB storage device as the final piece of the key. The system then performs the same hashing function that was used to generate the fingerprint originally and compares the result

with the stored fingerprint from the database of the group the user signed out from. If the fingerprints match, then the user is signed in, otherwise the system refuses access.

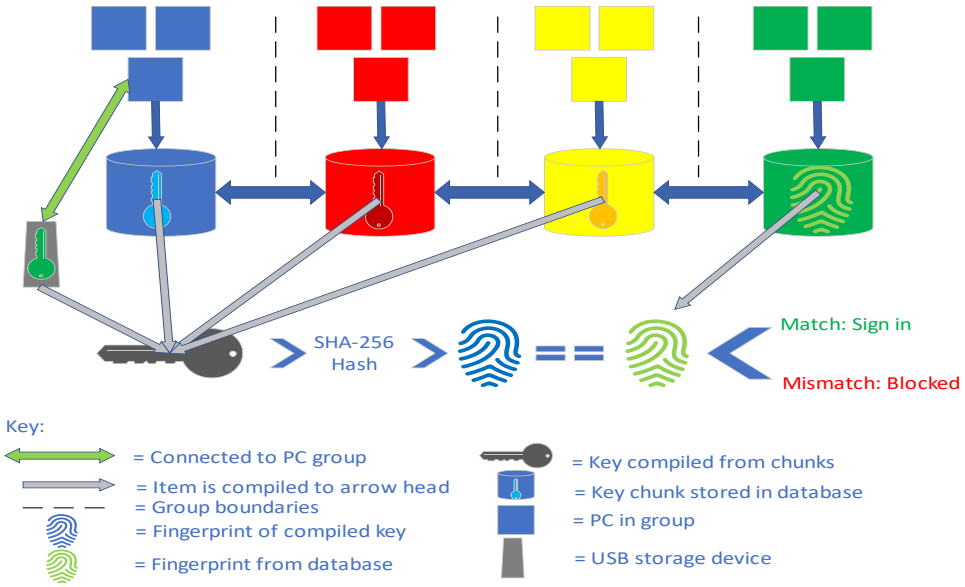

Diagram 1: Illustration of sign in procedure

In order for the sign-out procedure to work, the USB device and user credentials are configured locally on the *'dist_login'* main server, which places the first USB chunk onto the device for sign in. Diagram 2 illustrates the sign-out procedure for a user signing out of the 'green' group of machines. Once they start the sign out procedure, the key that was generated at signup/sign in is broken in to equal length pieces based off the number of distribution databases that are configured and stored in databases attached to the other three groups, and a USB storage device stores the chunk from the current group into the USB storage device that the user takes with them to sign in wherever they wish to next. They then simply need to follow the sign in steps as illustrated in Diagram 1, and they will be able to sign in using two-factor authentication. The tables within the databases will have 3 columns as illustrated in Figure 1. These three columns cover the storage of the key chunk and username it relates to, as well as the timestamp used in the salting of the key during generation. This allows the sign in procedure to perform the exact same fingerprint generation, using the exact same parameters. This precision means that the only possible combination is the correct key chunks, having even one character different in any chunk, will result in a mismatch.

Username	Key chunk	Timestamp
John	412CB322137D81A5….	17:20 09/02/2019

Figure 2: An example entry of data in one of the distributed databases

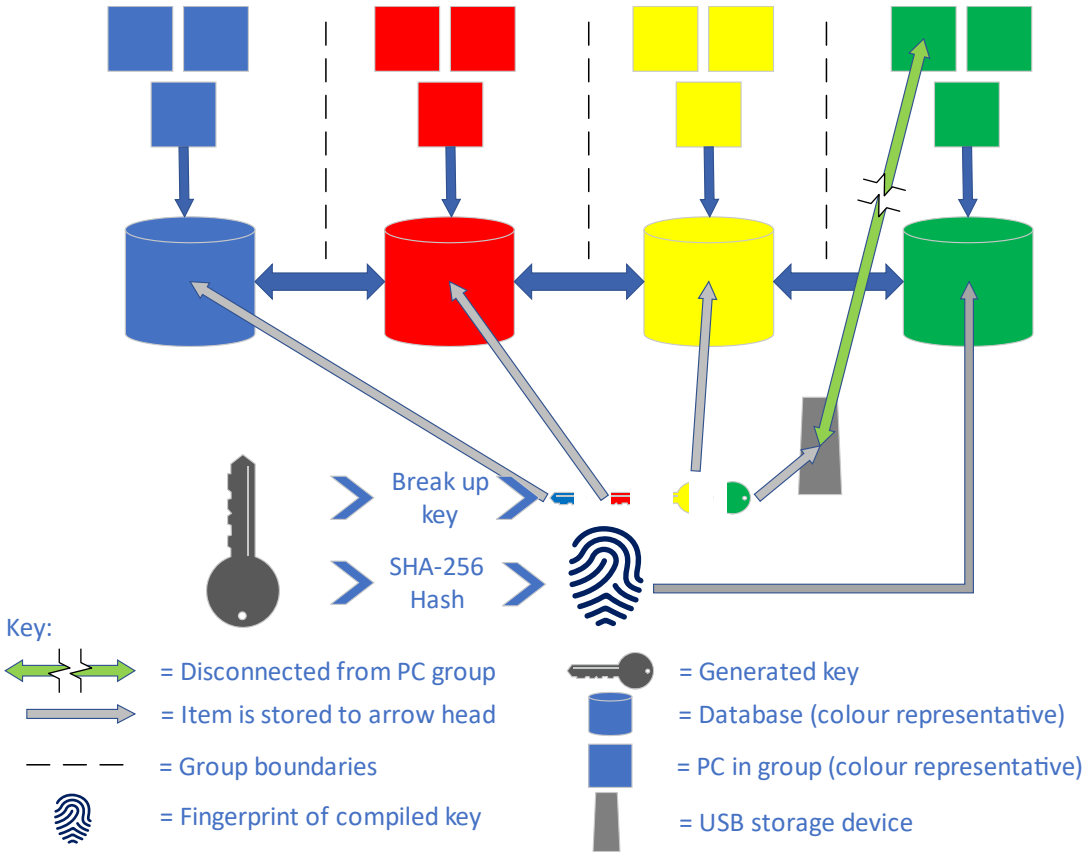

Diagram 2: Illustrated sign-out procedure

One key feature that Diagrams 1 and 2 show is the fact that the authentication key is stored across multiple databases present in different groups across the network. This makes it possible to use the databases as backups for each other, storing 2 parts of the key in a rotational fashion, so that they still rely on each other to compile the complete key, but they can recover from one database being taken down through the storage of two key chunks in a redundant measure. This feature will not be addressed in this paper but is a speculative feature for a future update.

With regards to communication between the databases and the users' machine, encryption will be implemented when data is sent across the network. This will mitigate the concern of malicious actors 'sniffing' the contents of the SQL data in an attempt to compile the key themselves through application layer data extraction. Additional security will be implemented between these databases to establish which hosts can interact with them. This could manifest as firewall rules to only allow the databases to request data from each other, or for a reactive measure for other databases to protect themselves with in case one of them is detected as being compromised in some way.

Development

Coding approach

Initially, the project was being developed using GNU Nano (Allegretta, 2018) an opensource text editor, however as the project became increasingly complicated to develop, it was moved to PyCharm (JetBrains s.r.o., 2018) due to its ability to maintain compliance with the PEP-8 (van Rossum, et al., 2001) python programming standard. Zhang et al. explain the importance of using the PEP standards in creating python code; *"Good software engineering practices guide the effort, including the use of type hinting, a Model-View-Control pattern, PEP 8 code and PEP 257 documentation styles, unit testing, and continuous integration."* (Zhang, et al., 2018) PyCharm also has the benefit of full Git integration with support for the GitHub platform. This granted to project version control within the Integrated Development Environment (IDE), meaning that the code was easier to manage. Other IDE's for Python that could have worked for this project include PyDev with Eclipse, a plugin for the hugely popular Eclipse IDE software package (Eclipse Foundation, 2018). It offers a wide range of tools and code checking and software testing tools similar to PyCharm (JetBrains s.r.o., 2018), however didn't offer Git integration like PyCharm, and required learning the Eclipse IDE itself too. Another alternative to PyCharm is Thonny, Annamaa, the author of Thonny, highlights the key goal of their software; *"With Thonny, our goal has been to offer single environment which integrates all the tools necessary for first programming course. Each of these has been designed with beginners in mind, unlike similar features in professional IDEs, which aim for greatest user performance and require much steeper learning curve."* (Aivat, 2015) The push towards PyCharm from Thonny for this project, was Thonny's orientation towards newer programmers, as opposed to a more experienced developer such as the one working on this project. Therefore, PyCharm emerged the IDE of choice for development.

Code structure

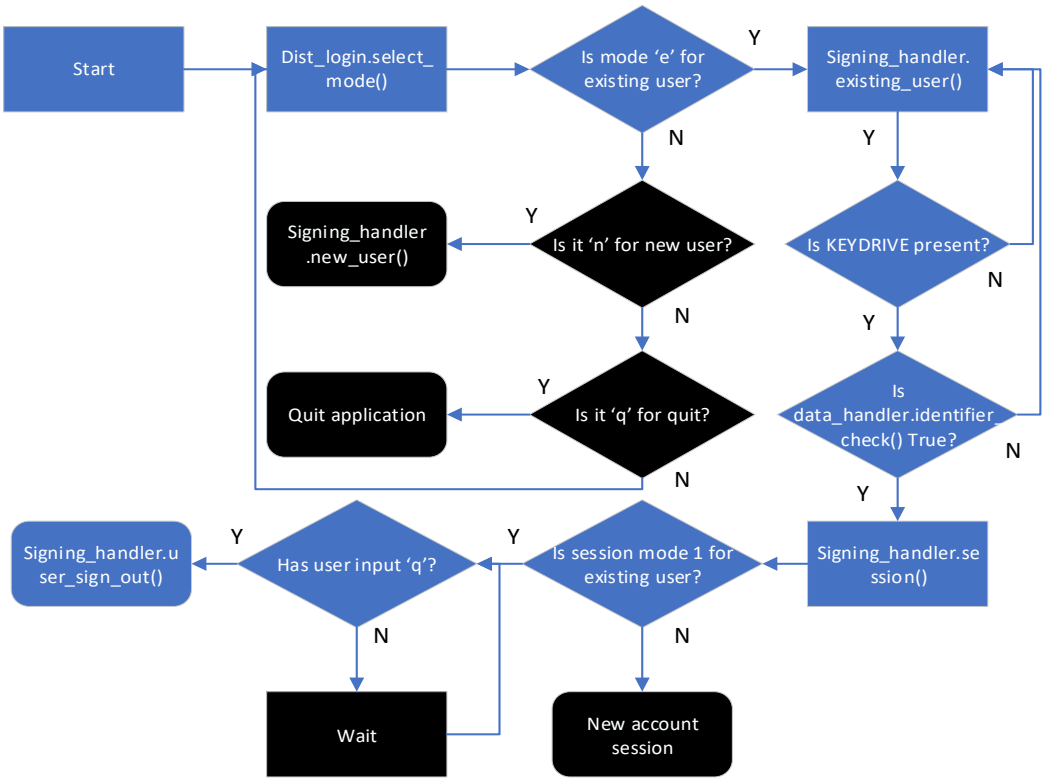

Figure 3: Flow chart showing a typical login procedure through the structure of the program

```
def new_user():
    print('Create new account')
    new_username = input_username(1)
    new_password = input_password(1)
    hashed_password = hash_password(new_password)
    output_results(new_username, new_password, hashed_password)
```

Figure 4: The 2 frequently used functions of the program are a set of function calls (not final version)

```
        if mode is 1:
            arg = get_password('New Password: ')
        elif mode is 2:
            arg = get_password('Password: ')
```

Figure 5: Use of 'is' and 'is not' in python code to make it more human readable (not final version)

The structure of the project began with abstracting the core functionality of the project into several defined functions in python. This allows the program to simply start with a single function call and the rest of the program will call the other functions that have been defined when needed. The exact flow of each of these calls during a typical login process can be seen in Figure 3. One particular prominent example of this is the '*new_user()'* and *'existing_user()'* functions as seen in Figure 4 with the *'exisiting_user()'* program path being detailed in Figure 3. There was also a move to make the notation more consistent by using verbose language instead of programming symbols and terminology. This decision was made to utilise pythons 'pseudo-code' appearance in a way that makes the code appear more logical to read for a human, as seen in Figure 5.

```
from hashlib import sha256 as sha256_hash_data
from hashlib import sha1 as sha1_hash_data
```

Figure 6: Import statements at the top of the 'crypto_handler' module of the program

Import statements were also made with aliases, this gave the imported libraries a clearer definition of what they were doing. This decision also had the benefit of only importing what is needed from a library, instead of importing everything with a blanket *'import all from library'* statement. For example, Figure 6 shows the import headers for the encryption module of the program, these import statements import only the functions that are needed from the selected libraries, while also renaming the functions to what they will be doing in the program, which in this case is hashing data that is passed to it using either sha256 or sha1 respectively

Initially, the generated authentication key was created by combining the username, password and a date/timestamp as salts in to the sha-256 data hash function. Once the project was able to create keys successfully, the keys were written to two separate text files, *'Chunk0.txt'* and *Chunk1.txt'* within the working directory of the main program. Once the system was able to read from text files to successfully verify the compiled keys fingerprint, it will be a simple conversion of the functions that interact with these text files into SQL actions to manipulate the same data. The text documents only allow for a single user to sign in and sign out of the system but does prove that the concept of fingerprinting key data once it was compiled from two different sources works upon restarting the program. Once this proved the projects functionality was capable with one user, it was then implemented with MySQL (Oracle Corporation, 2019) databases later in development.

The program originally began as a single file that had all functions and processes within it. This structure, while useful in initial stages of development, became far too long to be easily readable. To

remedy this, the project was split in several modules or *'handlers'* that were designed to break up the functionality of the program into easily understandable files, each dedicated to a specific feature, or set of features of the main program. The main file that starts the program *'Dist_login.py'* became a single function module that is the initial mode selection part of the program. It then calls the relevant modules from there, and they in turn call the modules that they require. This re-structure lead to significant decrease in the number of lines of code in the main *'Dist_login.py'* program, as seen in Figure 8 the total number of lines is just over 15 times less than the single file structure, 412 lines down to 27.

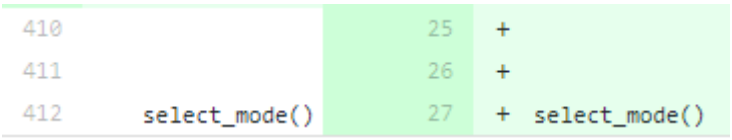

Figure 8: A comparison of the number of lines of code in last github commit before the re-structure (left), and the lines in first commit using the new structure (right)

The creation of modules also meant that there was a reduced number of import statements in each module header, this makes it clearer as to which module is using which libraries, as well as showing the relationship between modules. Figure 9 illustrates this, it shows the header of the *'data_handler.py'* module, which deals with checking, compiling and splitting the security data of the program. As the three final import statements show, the *'data_handler.py'* module requires the functions of the MySQL (Oracle Corporation, 2019) and USB handler modules, as it does not store the data directly. Instead this module prepares data before it is written, and formats data once it reads in from the data sources that the imported modules interface with directly. Every module must be present in the same working directory when the main *'Dist_login.py'* module is called.

```
import re as regular_expression
import usb_handler
import group_a_mysql
import group_b_mysql
```

Figure 9: The import part of the *'data_handler.py'* module

With regards to interfacing Python with MySQL (Oracle Corporation, 2019), the modules that interact with the databases for password and authentication key data storage have a single connect statement in their headers. This proved useful in increasing the speed of queries and actions in the databases. Before this implementation, each SQL function in python set up a connection to the database, executed its query/action and then closed the connection. This created a significant and noticeable delay in signing a user in and out of the distributed login system, therefore the decision to a have a constant connection created by the SQL modules upon start-up was made. This improvement gave a 20, to 25 second decrease in sign in time, which was a clear performance improvement for the overall project.

As seen in figure 10, the structure of each SQL handler header sets up the connection and the cursor for executing commands for each of the databases that the program interacts with. Each SQL module also has a *'sql_close_connection()'* function that allows for a graceful shutdown of the program's SQL connections, as shown in Figure 11. The constant connection created by this structure, meant that the program had a constant connection to the database while active. This prevents other machines from connecting using the same username and password that the program is given to perform its SQL queries and actions. This also had the additional benefit of reducing the

number of lines of code in the SQL modules significantly, as there was no need to repeat the connection declaration as seen in Figure 9 for each function within the module.

```
sql_connection = mysql.connector.connect(
        user='node',
        password='n0d3',
        host='192.168.152.131',
        database='dist_login')


sql_cursor = sql_connection.cursor()
```

Figure 10: The header of one of the SQL modules, initiating the login to the MySQL (Oracle Corporation, 2019) server, and creating the cursor later used to execute queries and actions to the MySQL (Oracle Corporation, 2019) database. (Username, password and IP address are set by the administrator implementing the program)

```
def sql_close_connection():
    # print("Group A close connection") # Uncomment this for debugging sql
    sql_connection.close()
```

Figure 11: The function that closes the SQL connection of the module, used during the graceful shutdown of the program.

Developed features

To combat SQL injection attacks, a checking function was added to the project. This function takes the users entered data and compares the string to a text file containing a list of common SQL injection strings, as well as some additional strings that are specifically relevant to the queries and commands executed by the program. This ensures that when the username is committed to the database, it is sanitised for these injections strings. These attacks are performed by encapsulating SQL syntax into user input, and that formatted input then returning to the user database information, such as the number of columns and their names. Due to the need for secrecy in this projects design, this potential vulnerability needs to be mitigated using the common SQL injection strings file that the system will check all user input destined for databases against. While this is much more effective than no user input cleansing, it is not entirely comprehensive in mitigating this vulnerability, as new injection strings could be crafted in the future. This feature is illustrated in Figure 12.

```
def sql_inject_check(arg):
    if arg in open('../files/sql_inject_strings.txt').read():
        return True

    else:
        return False
```

Figure 12: The function that checks the users submitted string against a text file contain forbidden SQL strings

The system requires a USB device named ‘KEYDRIVE’ to be present for any sign-in or sign-out sequence. To meet this requirement, the system has a *‘detect_drive()’* function that will check that the drive is present before each USB read or write function. This ensures that the user is prompted

for their key storage drive, this means that the security data is written or read successfully without interruption. This also ensures that the correct USB device is connected to the system, avoiding the key being written to the wrong device, or malicious strings being read from an imposter USB device. Should a USB device be disconnected during the process the key will be lost, however the account can be recovered by re-configuring the USB device as it was when the account was set up. This function has also been designed to work on windows and Linux kernels, as seen in Figure 13 this is implemented by reading the os.name type imported from the host system that the program is running on, where 'nt' is for the windows nt kernel, and 'posix' is for most Linux kernels. This feature is quite lax when it comes to device verification as it could be improved with further fingerprinting of the USB device. This could be simply recording the devices total capacity, its serial number or other manufacturer data that can be interpreted from the hardware. Another possibility is to replace the USB storage device with a one-time string entry device such as YubiKey, this possibility is discussed in further detail in the conclusion of this report.

```
def detect_keydrive():
        if os.name == 'nt':
            try:
                subprocess.check_output("wmic logicaldisk list brief | findstr KEYDRIVE", shell=True)
            except subprocess.CalledProcessError:
                return False
            return True
        elif os.name == 'posix':
            output = str(subprocess.check_output("lsblk -o MOUNTPOINT | grep KEYDRIVE", shell=True))
            if not output.find("KEYDRIVE"):
                return False
            else:
                return True
```

Figure 13: The 'detect_drive()' function that checks for the presence of the KEYDRIVE

One other prominent feature is the implementation of groups as part of the sign in procedure. This feature requires a user to declare the group that they last signed out from as part of their process of proving identity to the server. This allows for a memorable piece of information such as the group's name or a code pertaining to the name, to be a useful element in the assessment of a user to ascertain if they are who they claim to be. This is enforced using the MySQL (Oracle Corporation, 2019) databases that are situated in each group. So for example a user may sign out of a finance department, with a department code of 'FINDEPT' and they then attempt to authenticate themselves when signing in to the a different group, the distributed login system will then first require them to declare which group they last used, in this example 'FINDEPT' and then the server is aware that the piece of the security key that the user has on their USB device is the key chunk originally destined for the department under the code 'FINDEPT'. If the server were to sign them out of a different group, say 'SALESDEPT' then the key chunk for 'FINDEPT' would be stored in the related database, thus keeping the distributed nature of the system and using it as a method of authentication. Figure 3 illustrates the decision process in the program for the group authentication feature.

```
def select_group():
    group = input('Group: ')
    if len(group) > 1:
        print("Group not recognised, please use a single letter \'a\' or \'b\'.")
        select_group()
    elif not regular_expression.match('[a]|[b]', group):
        print("Group not recognised, please try again.")
        select_group()
    else:
        print("Group " + group + " selected")
        return group
```

Figure 14: The main function for selecting groups as part of the authentication process

As seen from Figure 14, in the prototype the project simply uses the group names ‘a’ and ‘b’ for illustrative purposes, but these names can be any compatible string. Once the user enters a valid group name, the system them checks to see if the key chunk that is submitted matches as part of the rest of the key, and if the fingerprint matches to what is recorded in the databases, then the user passes that stage of the authentication process. If the user gives the incorrect answer, such as a group that they did not last sign out from, then when the fingerprint check of their key chunk as part of the larger key is checked against the recorded fingerprint it will not match, and the authentication is no successful.

Testing

To properly test the program, there are two angles that were considered; security and performance. Both these metrics are important to this project as it is primarily designed to manage 2FA across several clients organised into groups under the different login servers. Due to its main functionality being security focussed, an assessment of the confidentiality, integrity and availability limitations of the software is paramount. In addition to this project being a security solution, it is also a Python program, which ideally should be given at the very least, boundary software testing to ascertain if it functions as intended with a selection of valid inputs, as well as handling errors in input and functionality gracefully to help administrators troubleshoot if anything goes awry. The plans for these testing methods are outlined in this section to provide context and replicability for further testing by others.

As this project is written in Python 3.7, a performance test was also carried out to assess the project speed and reliability with incorrect inputs using the unit testing framework built in to PyCharm (JetBrains s.r.o., 2018). This is of particular importance to this project as it is designed to be a layer of security that protects multiple user accounts and will have several requests made of it in a deployment within a large organisation. The testing metrics are detailed below, but in a broad sense they are designed to test the boundaries of user input, for example the character limit of 99 applied to the username variable. These tests are designed to ascertain the programs ability to properly process valid inputs, such as special characters in the password and username variables. Further testing is designed to check that the program can properly handle inputs that exceed the defined variable boundaries, such as an input that exceeds 99 characters for the username variable, as well as its ability to gracefully handle system failures, such as the back-end MySQL (Oracle Corporation, 2019) databases going offline to inform the administrators where the issue in the process is.

To test network traffic confidentiality, three other multi-factor authentication methods were deployed and tested alongside this project. The two elements that are being compared in the graph

below are the number of factors in the authentication process, and the total time taken from the user initiating the connection, to a connection being established and the remote ssh shell being shown. This was timed using a stopwatch on the desk next to the test machine, and therefore a room for error must be accommodated in these results. In terms what the testing defines as a factor in the authentication process, this is one or more of *"something you have (e.g., a hardware token), something you are (e.g., a fingerprint), and something you know (e.g., a password)."* (BRAINARD, et al., 2006). In the case of this project there are 4 observable authentication factors: The KEYDRIVE and the chunk held within it as 2 things you have and your password and last used group ID as two things you know. A full breakdown of these observations is also provided below the accompanying table to these tests.

The final test performed on this project was to test usability and user experience. The test was comprised of the distributed login project implemented alongside other multi-factor authentication solutions that could also be deployed to protect an ssh server, which was the service of choice for testing this project against others. Each test was performed five times, twice by two users that were totally new to all solutions that were tested in the context of securing ssh, and once by the main developer of the project. The solutions were tested by using the 'time' linux command from a pre-configured ssh client connecting to a multi-factor authenticated ssh server pre-configured for each solution in each test. The users were simply asked to sign in, and then type 'exit' to sign out and get the official time of their session from their initial client machine. An average of their times taken to sign in and out using each solution was taken and are presented below as a graph.

Software testing with unit testing

The testing performed with the built-in unit testing feature of the PyCharm IDE to ascertain the handling of a range of possible inputs was performed on all functions in libraries that return a value to ascertain that each function would return the expected value for incorrect and correct inputs. Libraries and functions such as those that handle the SQL side of the project had a limited testing range, as they mostly commit or read data from MySQL (Oracle Corporation, 2019) databases, and therefore the testing would need to encompass MySQL (Oracle Corporation, 2019) also, which is outside the scope of the testing done in this instance. The libraries that take in user input were tested manually, as unit testing is best suited to testing non-interactive functions.

For example, the 'dist_login.py' library requires one of three characters to be entered for it to call the correct libraries and functions to add a new user, sign in an existing user or quit the program entirely. An example of where unit testing was used, was the 'identifier_check' function found in the 'data_handler.py' library of the project, which takes in two values, and returns a True or False Boolean value based on whether the two values match or not. Other functions were more easily tested, as they produce a very specific result from various inputs. For example, the 'generate_key' and 'fingerprint_data' functions in the 'crypto_handler.py' library, will produce a unique hash from the input that is injected into them as parameters. As this input is already validated in the 'input_handler.py' library, there is no need to test for handling of incorrectly formatted inputs, but instead there is a simple need to check that the expected unique hash is created every time the same input is passed to the functions. As seen in the unit testing script in appendices 1, these strings were both a normal generated key and a simple 'test' string for the 'fingerprint_data' function, and a two simple 'test' strings, formatted with both single and double quotes to account for potential variation on different systems.

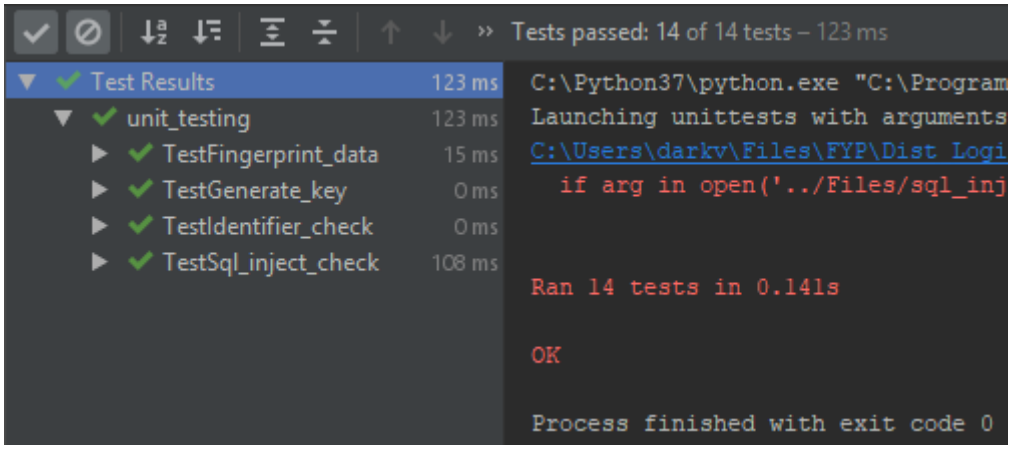

Figure 15: The output from the unit testing module in PyCharm, full testing script in appendices 1

As seen from the output of the testing above in Figure 15, the unit testing was 100% successful for four functions from two libraries; 'data_handler.py' and 'crypto_handler.py'. These tests were designed to check the output of the functions matches the expected output based off pre-determined inputs. An example of these test inputs is putting a pair of matching strings into the 'identifier_check' function and expecting a true Boolean output, and then inputting two different strings as parameters and expecting a false Boolean output. There were some functions that required extensive testing, such as the 'sql_inject_check' function, which needs to produce a true Boolean output if it detects any of the phrases in the 'sql_inject_strings.txt' file in the project directory when assessing the input parameter. To simply this test, the test case would place all strings from the 'sql_inject_strings.txt' and another file containing names that should produce a false Boolean result, into an array and running each element of those arrays through the function to test its output.

Network confidentiality testing with Secure Shell (Ssh):

An implementation with Secure Shell on Ubuntu 18.04 was performed as a means of testing the security of this project across a network. This implementation can be seen in the 'Ssh_Implementation' branch of the GitHub project. Utilising the automation in bash, this implementation launches the 'dist_login.py' script upon a user signing in via a preconfigured ssh connection. In addition to the ssh connection, there are also two python sockets running with transport layer security which are used to transfer key chunks between the server and client. The unique approach to Transport Layer Security (TLS) that this implementation uses, is the way the certificates are transferred between the machines. Traditionally, the administrator would copy the keys in a similar way to ssh when implementing 'password-less' ssh, by copying the certificates over the network. This does present the issue of interception however, as a malicious actor on the same network could 'sniff' the key exchange and ascertain the keys and certificates as they are exchanged initially. With this project's implementation via python sockets, the keys for both the client and the server are transported via a USB storage media, the same media used to store the key chunk during normal operation. This is designed to avoid the initial key exchange being 'sniffed' by a malicious actor who has compromised the network, whilst also maintaining the encrypted communication benefits of transport layer security. This need to physically connect the authentication media to the login server allows for greater security controls when it comes to new users being added to the system, as physical security can be implemented to protect the server's hardware from being accessed in an un-authorised manner.

```
user@mysql1:~/Dist_Login/Libraries$ sudo python3.7 dist_login.py
Select 'e' for Existing User, 'n' for New User or 'q' to quit: n
Create new account
KEYDRIVE not detected, insert KEYDRIVE into local server port then press enter to try again. (New ac
counts cannot be created over ssh)_
```

Figure 16: The implementation with ssh during configuration of a new account, note that configuration over ssh is deliberately absent from the program.

In terms of functionality, a few additions were made to the master copy of the project to accommodate ssh. The first being the addition of sending and receiving scripts for the transfer of the key chunk destined for the KEYDRIVE. Integrated with TLS, these sockets were mainly an arbitrary creation to illustrate the potential functionality of securely transferring the key chunk. These scripts were 75% automated, with only the clients sending script needing to be triggered manually by the user to submit the key chunk from the USB device. Additionally, due to the python sockets being used only as demonstration and not as a full feature of the program, the received key chunk from the client is cached in a temporary text file locally on the server, which is wiped every time a user successfully logs in and out. Although this can be considered a vulnerability to data confidentiality, the design of the implementation with ssh was developed for testing the confidentiality of network packet data only, and not the security of the file systems of the server or client, the rest of the communication with the server works over the regular encrypted ssh protocol.

Upon examining the contents of the traffic flowing between the server and the client with this implementation, it can be observed that all data is indeed encrypted in the payloads of the python socket traffic. Figure 17 shows a screenshot of a packet inspection of the python socket traffic in Wireshark (Combs, n.d.), and it clearly shows that the payload contents are ‘Encrypted Application Data’ this encryption was created securely using TLS. This confirms that network monitoring by a malicious actor will not yield any sensitive information related to the login system without prior acquisition of the certificates and keys that are required to read the conversation between the two endpoints. As for the encryption and protection of the user’s session, the implementation relies on the already established privacy built in to ssh, this can be seen from figure 18 which is a Wireshark screenshot of an ssh conversation between a server and a client machine. Note that the data is encrypted in Wireshark, meaning that much like the python sockets that were implemented as part of this project, a malicious actor listening on the network would be unable to interpret the plain text data without prior access to the certificates and keys used to encrypt this client/server conversation.

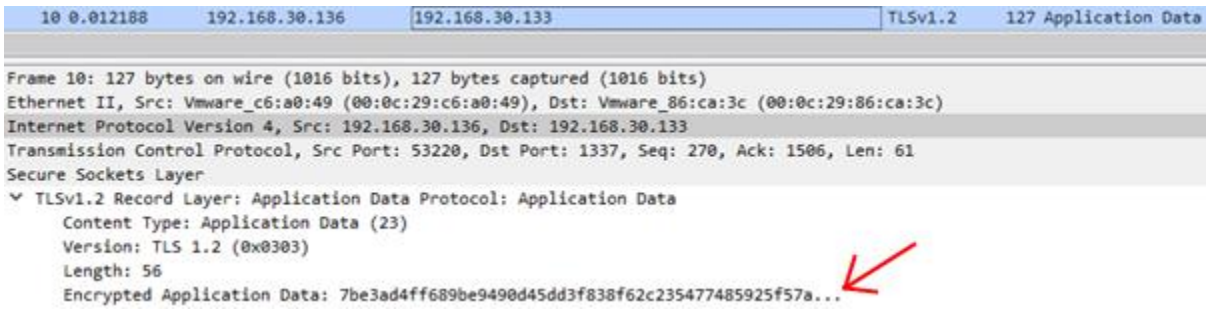

Figure 17: A Wirehsark capture of the encrypted conversation between the two python socket endpoints, highlighted is the encrypted application data.

```
    62 29.571444     192.168.152.135      192.168.152.134                SSHv2     1006 Server: Encrypted packet

Frame 62: 1006 bytes on wire (8048 bits), 1006 bytes captured (8048 bits)
Ethernet II, Src: Vmware_5d:c1:bf (00:0c:29:5d:c1:bf), Dst: Vmware_5d:00:62 (00:0c:29:5d:00:62)
Internet Protocol Version 4, Src: 192.168.152.135, Dst: 192.168.152.134
Transmission Control Protocol, Src Port: 22, Dst Port: 56368, Seq: 1506, Ack: 1829, Len: 940
SSH Protocol
 SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
      Packet Length (encrypted): e218e638
      Encrypted Packet: 8881bdb86408af68825fc037918986a29ce6bc1036ba61a2...
      MAC: d16ca9b5528c09622ab36de4c56796f5
```

Figure 18: A Wireshark capture of the encrypted conversation between an ssh server and a client

Performance testing with user experience timing:

With regards to performance testing, this project was examined from a usability perspective. These were the speed of the system measured as the number of seconds taken from initiating the login procedure to signing out of the system again, to compare the speed of authentication by each solution. As seen from Figure 19, Distributed Login has a comparable speed with others, as its average time of 36.306 seconds, which is faster than SMS by 29.494 and only 7.07 seconds behind Email authentication making it comparable to email and SMS as an authentication solution. With regards to speed, Figure 19 shows the speed at which a test user sign in and out using each authentication solution. The test was performed five times on each solution, with two new users and one experienced user to acquire a more representative average for each solution. The test was performed by initiating the 'time' command from the Linux client that the user was signing in from when they ran the initial command to connect to the ssh server that each solution was set up to secure. Once the user was signed in, they signed out as soon as they could by typing 'exit' into the remote terminal, at which point the time command reports back how long it took the user to initiate and close their ssh session with each solution.

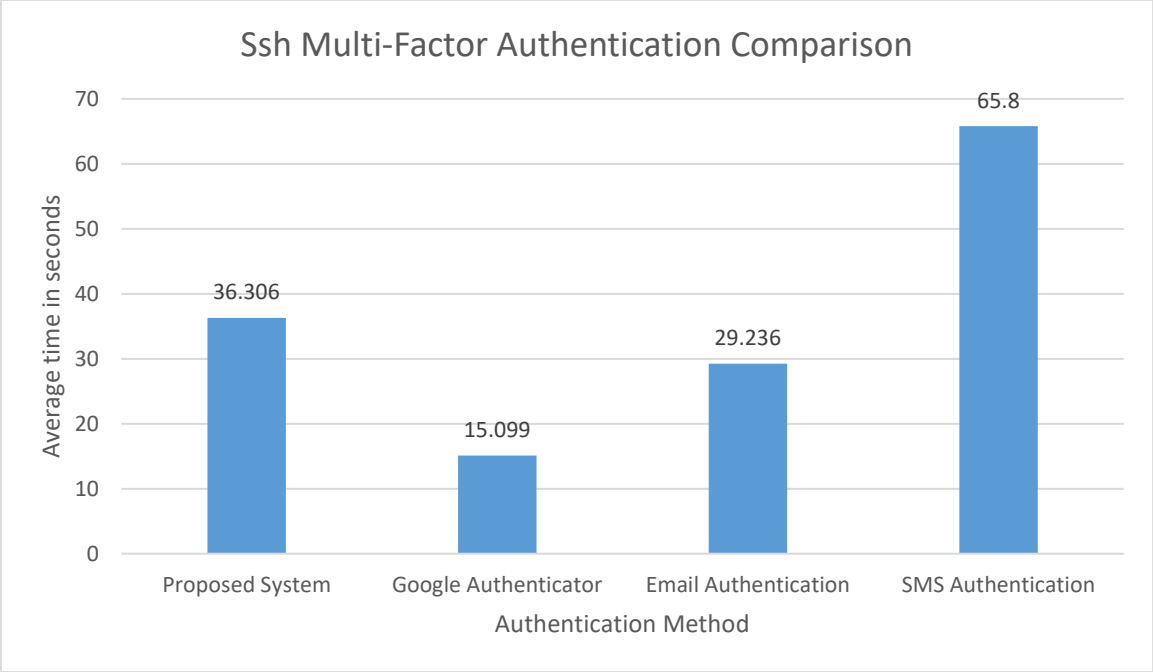

Figure 19: Comparison of Distributed Login versus other multi-factor authentication methods in terms of average time taken to authenticate a user and for them to sign in, and then out again. Full result data can be seen in the appendices 2

As can be seen from Figure 19, the average time taken by different users to sign in using this project is in line with the performance of similar multi-factor authentication solutions. Whilst this projects performance is not the fastest, being beaten out by both email and Google Authenticator (Google LLC, n.d.), it is worth bearing in mind that part of the test had to be performed manually, due to the solution not being fully automated to send and receive key data of the encrypted python sockets. The users had to manually initiate the send of their key data to the server to complete sign in. There are also other points where time is consumed with extra steps that can be automated in a full implementation, such as the system being contextually aware of which group that the sign in system is authenticating to, making the need for the user to enter which group they are signing in to redundant. Lastly, the system also has an additional step to the quit procedure when compared to the other solutions, as the user needs to press ‘q’ to quit the project itself, and then type ‘exit’ to close the ssh connection and submit the time result back to the client terminal. These improvements in a full implementation will very likely decrease complexity for the end user, whilst increasing the speed of a typical sign in procedure using this project.

Conclusion

Product evaluation

One of the main strengths of Distributed login when compared to other popular multi-factor authentication systems, is the number of authentication factors that the ssh implementation possess. As seen from Table 3, the Distributed Login 'Ssh_Implementation' branch of the project has 4 factors of authentication, these are; The USB device as 'something you have', the password as 'something you know', the last group you signed out from as another 'something you know' and the certificates and keys for both the Python sockets and the ssh connections as another 'something you have'. The closest in terms of factors that comes close is email authentication, which requires a second password for the email account to acquire the authentication code sent by the login system. There is a caveat however, in that some users may have their mailbox configured to display as a notification on their mobile device lock screen, which can bypass the need for a second passcode or password to access the authentication code. This indicates that as an authentication system, Distributed Login has strength in multi-layered validation of a user being who they say they are, due to its inclusion of a physical device that is in the users' possession for them to use as a method of authentication of who they are, as this device is linked to their account. As this USB device contains a unique piece of the authentication puzzle metaphorically speaking, this increases the strength of the security that Distributed Login provides over the other solutions it was compared against.

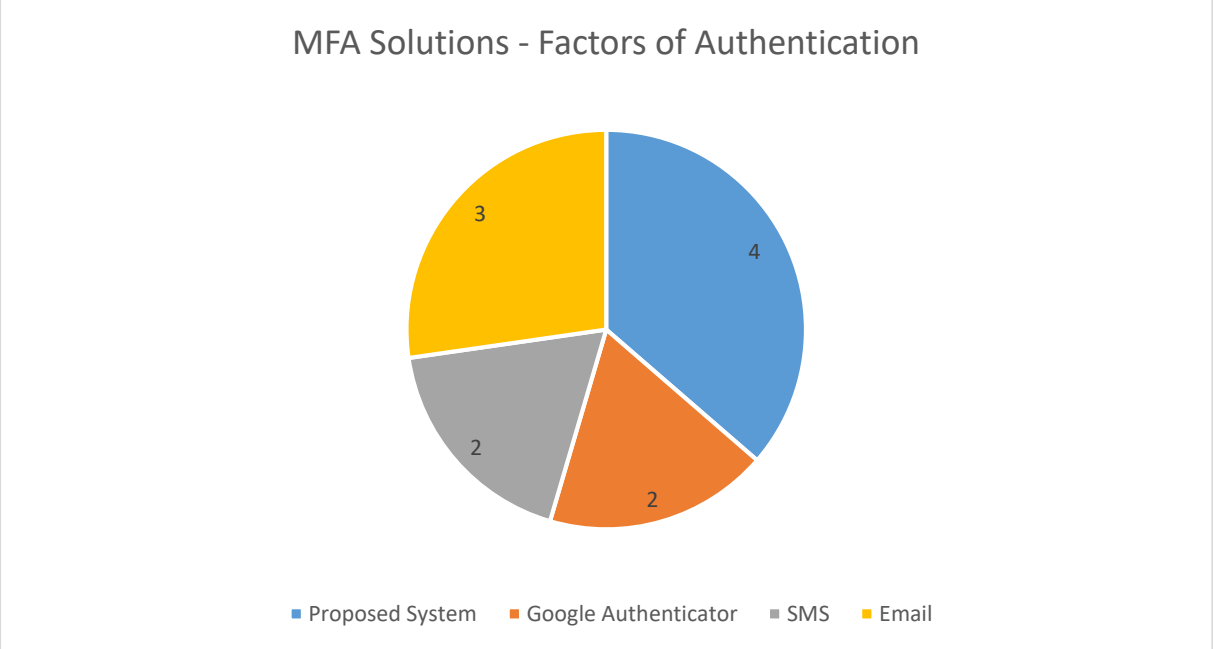

Figure 20: A representation of the number of authentication factors found in each solution compared to Distributed Login.

Additionally, one other clear strength of Distributed Login is its method of security data distribution. As the authentication key for the user is generated using an ever-changing salt though the use of a datestamp in the plain text input alongside the users' name and password, this makes security keys highly unpredictable and as they are not a readable string this gives the security key a layer of obscurity. Complementing the unique keys, is the distribution of the security key across multiple areas and devices. This increases the confidentiality of the security key data as the data is not only stored in a database, but also on a USB device that once removed from the users' client machine is completely disconnected from the network and is not vulnerable to any wireless surveillance unlike

mobile devices. This provides mitigation against data leaks such as the adobe hack of 2013 as covered by Hern in the Guardian. In the article, Hern elaborates as to why the hack was so detrimental for Adobe systems Inc. "*The method, called ECB mode, means that every identical password also looks identical when encrypted.*" (Hern, 2013). The three main issues with Adobe's method of security data encryption was that 1) They did not add any salts to their encryption inputs meaning all passwords that were the same in plain text, were encrypted to the same cipher text, allowing for easy reverse engineering of the algorithm that Adobe used, making decryption of all other passwords in the leak easier. 2) The accounts to which the passwords related were not protected with multi-factor authentication of any kind, which would have protected users accounts as anyone using a stolen account would not have been able to get to the authentication data that a multi-factor authentication system would provide. Finally, 3) the entirety of their users' account data was stored in one, central database. This meant that once that single database was compromised, their entire set of security data was accessible. Distributed Login mitigates this problem by storing at maximum 50% of security data in one of, at minimum two possible databases, and not both at once. Whilst the other half of the security data is stored on a separate USB device which is inaccessible most of the time and in the users' possession.

Additional Features

There are several features and improvements that could be considered to improve the functionality and viability of this project. These improvements and features were realised as part of the development process, but are not implemented in this iteration, due to this mainly being a first stage prototype to demonstrate the potential of the distributed login as a concept. Features mentioned here are all possible theoretically but have not been tested in a practical manner, therefore they may not be technically possible as features. The ideas that these features present however are something to be drawn from these ideas, as there may be another technique that achieves the same desired functionality in a different manner, or the functionality may become irrelevant if the landscape of multi-factor authentication changes drastically.

To provide protection to the contents of the USB device, one possible addition to this project would be a method of encrypting the data on the device, so that it is not readable to unauthorised devices or users. This would obfuscate the data to those reading it directly, giving an additional layer of security to the data, whilst still maintaining the core functionality of the distributed login system as a whole. As an example, a Vigenere cipher could be implemented by using the fingerprint generated from the generation of the full key as the key to the cipher. This allows the program to re-user existing security data as keys, instead of generating another set of unique keys for each user, keeping this less complex. When the user's client submits the encrypted security key, the login server would be able to reverse the rotation of each character in the clients submitted chunk by reversing the effect of each character in the stored fingerprint to get the decrypted data. This means that the user's client does not have any part in the obfuscation process, eliminating the client machine as an attack vector on the data itself. As a further extension of this feature, the rotation could be uniquely generated each time in a similar way that the security keys themselves are generated by the distributed login server program, thus ensuring a varied encryption method each time a key is sent back to the user's client machine at sign out.

Another feature that could be added to this project could be an account recovery service, or a redundancy to user accounts. Currently, if the user losses or destroys the USB device, the entire system cannot function as it requires the USB device to work. With this in mind, an account recovery service could be implemented, whereby the user can reconnect a fresh USB device to the server and initialise it as their new security key with a new key chunk and fingerprint for their account. Due to

the intentional need for physical access to the login server to initialise new accounts, the account recovery service could leverage a lot of the functionality the project already has, and simply use it to generate new keys and fingerprints for an existing user in the SQL databases, and update these databases with the new credential data. The deliberate requirement of physical server access means that physical security measures, such as access controlled doors, can be implemented around the server. This bars malicious actors from registering their own device as the trusted USB device for a new or existing account. The main limitation with this implementation is that if a user needs access to their account quickly, the time taken to recover it has to account for the time taken to gain physical access to the server, or for a system administrator to re-initialise a new USB device for the user.

One of the main features of this project, is the implementation of groups within the database architecture. As each user signs in or out of a group, their account information is updated on the SQL servers to reflect their activity, including a timestamp. Upon signing out, the system also deliberately leaves a blank entry in the 'credentials' table of the 'dist_login' database relating to the group the user signs out of. This feature not only ensures that the user is who they say they are, as they are able to recall which group they signed out from last if they were implemented as department names, for example. There is also potential for this feature to be implemented as a session protection feature, for example when a user is signed in successfully their entire security credential and fingerprint data could be erased from all databases while they are signed in to their account. If a malicious actor then attempted to login in to the same account whilst it is signed in, the system can recognise that there is currently no security data related to that user and can deny access to the account until it signs out from whichever group it is currently signed in to. The use of groups also applies to security when the user is signed out, as a malicious actor attempting to sign in using a USB device stolen from the user, would also need to know which group the user last signed out from in a string format without being prompted. These both categorise as the 'something you know' class of authentication factors.

One alternative to the use of USB storage devices, is the deployment of Yubikey devices as the key chunk storage devices. Yubikey devices have a public python library that can be easily integrated with the existing projects python libraries to implement Yubikey devices as one-time password entry devices. In the example of the 'Ssh_Implementation' branch of the project, the users receiving python socket script, would write data to the Yubikey using the relevant libraries in the exact same way that it currently does with USB storage devices. The clients sending script can then be adjusted to send the data that the Yubikey injects to it as a parameter, to then be sent over the encrypted python socket established in the existing code. This would mitigate the risk associated with USB sticks for security data storage, as they are read by the operating system by default and any malicious program that is optimised for grabbing USB media data would be able to steal the security key before the user attempts a sign in. A Yubikey needs to initiate the data transfer to the host it is connected to, and this would therefore allow for more control over the flow of the security data from the user's physical authentication device and the sending script situated on their machine.

Diagram 1 and Diagram 2 showcase a system with four MySQL (Oracle Corporation, 2019) databases as the implementation of choice. However, the developed program only accommodates two databases, and therefore only two groups currently. The full implementation of this program is designed to exponentially grow depending on how many groups the organisation implementing it requires. For example, if a company has departments such as Sales, Administration, Marketing and Technical Support, then each department could be segregated by department, with their own dedicated MySQL (Oracle Corporation, 2019) database for security data storage, as part of the wider

distributed login system. A scale up to 4 databases would be an excellent evolution of this project, as it would demonstrate the ability of the program to handle many connections continuously, increasing its viability as an authentication solution for large scale organisations.

Due to the potential growth in the number of machines that this project could use, consideration was given to deploying the project in a cloud-based service to lighten the resource requirements of the project. Solutions from both Microsoft and Amazon were considered. The idea of situating the distributed SQL servers permanently in the cloud, as to be available to any machine with an internet connection, would have been an excellent demonstration of the projects ability to distribute authentication credentials across the internet. This would provide a layer of obscurity to the network traffic and data generated by the project during normal operation, increasing confidentiality whilst keeping availability strong through the availability of the internet. Ultimately however, there was no service that could provide the level of availability required without requiring some method of monetary subscription, and the project was therefore restricted to local virtual machines on the main development laptop used to both write the program and perform the testing of the project.

With regards to the connection between the server and the MySQL (Oracle Corporation, 2019) databases, there are a few basic network security tools that ideally would be implemented as part of the deployment of this project. The first would be a set of firewall rules that dictate that the server can only send and receive SQL data from the trusted, statically defined IP's of each MySQL (Oracle Corporation, 2019) server, with similar rules on each MySQL (Oracle Corporation, 2019) server referencing the login server. This would ensure that relay attacks could be largely mitigated due to the denial of other IP addresses attempting to receive or send data to the servers, as the connection between the server and databases would ideally be a constant one, with a potential security alert being generated if the TCP connection between the server and any databases is interrupted or disconnected. A second network security feature to implement is the encryption of each connection between the server and the databases. Similarly, just like the python sockets in the 'Ssh_Implementation' branch of the project use transport layer security to encrypt messages, a simple addition of this encryption method for MySQL (Oracle Corporation, 2019) could be implemented to provide confidentiality to the security data that is being transferred between these servers frequently.

One final feature that would increase the usability of this project would be a graphical user interface. This simply increases the appeal to non-technical users, whilst maintaining the functionality of the project. There a few graphical user interface libraries available for Python 3, such as TKinter, integration of this library with the project, whilst maintaining the projects multi-platform compatibility would be an ideal move in the right direction for the project. Similarly, a web interface would also be an excellent alternative to a desktop client application, as this would eliminate the need for libraries to be installed on client machines, and even muse the possibility of using the distributed login system as a means of web access to the organisation through a wide area network for example. It is recognised, however, that a web implementation would possibly involve re-writing some of the projects libraries in a web scripting language such as Node.JS, which would need to be investigated as well as a look into web design for the CSS and HTML that would also be required to make the system usable to non-technical users.

Points of concern for Cyber Security

The main point of attack that can be identified with regards to the use of a USB device for storing the security data is that it is a physical device that can be lost or stolen by an attacker. As there is not personally identifiable information that links a user to the device with the current implementation, this effectively means that if the attacker is able to obtain the password, guess the last group a user signed in to and, (in the case of the ssh implementation branch) obtain the ssh and python socket certificate and key files, then they will be able to sign in to a server with physical access to any of the configured clients using the distributed login service. There is also a question of user safety with regards to being tasked with protecting their authentication device, any attacker that is prepared to assault any of the users for their device will pose a significant risk to the safety of users of distributed login. Two possible mitigations for this risk is to have a policy that prioritises user safety over the protection of security data, allowing for users to give up or lose their device for it without consequence, with the system supporting the re-configuration of user accounts to accommodate loss of devices through authorisation of a new device, and the blacklisting of the older device.

One other concern is the relatively heavy dependency on MySQL (Oracle Corporation, 2019) databases as part of the core functionality of the project. These are not under full control of the project, as the application is designed and maintained by a third party. This means that the main security of the MySQL (Oracle Corporation, 2019) application is down to the work and testing of a third party, and thus there is potential for zero-day vulnerabilities and other security holes to be uncovered as time goes on. Due to their central position in this project, this concern can lead to a potential compromise of the entire system that surrounds those databases, and therefore could have disastrous consequences should the security data held within them were to be leaked or deleted, causing major disruption, particularly in larger organisations that this project is aimed at. The only real mitigation for this concern is to set up strict connectivity rules alongside the already 'bare-minimum' privileges granted to the 'node' users in the MySQL (Oracle Corporation, 2019) setup scripts included with the project. These connectivity rules would declare that only the login server running the main program of the project, and the MySQL (Oracle Corporation, 2019) databases can connect to each other with statically defined IP addresses, and only the required incoming and outgoing ports being open for connectivity. Additionally, transport layer encryption would be desirable to increase the confidentiality of the data flowing between the server and the databases.

Research techniques

With regards to the research in this area, the sources used and referenced in the literature review are solid and establish several of the generally accepted trends and innovations in the multi-factor authentication area of development and research. Things such as the 'something you know/have/are' method that allows for dynamic assessment and increased complexity of a login request. The literature around usable security are of particular importance to this project, as increased usability can lead to increased security, as compliance with protocol is made easier with usable security. Users that find the system easier to use, are more likely to adopt and use it properly especially if it can beat out the traditional username and password combination method that basic single factor authentication utilises. The discoveries made during this research have indicated that this project has this potential, particularly as the 'identifier' module that was developed to showcase this system can work alongside password systems is optional and is not an integral part of the authentication process.

The distribution of security data is another aspect that is particularly innovative, as it approaches the idea of security through obscurity, it does this through splitting the data across multiple databases. This distribution can be taken further by implementing these databases in different locations, form simply occupying different subnets in a network, to placing them in physically different geographical locations across the internet or wide area networks. There is even merit in utilising different database systems for the storage of the split security data, such as using MongoDB for one set of security data and MySQL (Oracle Corporation, 2019) for another set. This would allow for a newly discovered zero-day vulnerability as the other database systems that are used to store the rest of the data would not have the same issue and would therefore still protect the rest of the security data, thus ensuring that a total theft of user authentication data is significantly more difficult to pull off. These features and discoveries combine to conclusively suggest that yes, distributed multi-factor authentication can provide security benefits over traditional methods. This is all whilst maintaining a similar level of usability and performance to match those traditional methods.

Evaluation of approach

For project management a Gantt chart (redacted) was made and followed as closely as possible. This allowed for the creation of a timetable to dedicate the appropriate time scale to each activity and task that needed to be completed. This also helped in achieving targets on time and having deliverables ready to showcase to the tutor for the project. The main development method when it came to software features was run in an entirely agile way. This was mainly due to the uncharted territory that was the structure and functionality of the project, as nothing similar had been written in Python before. Therefore, the features and functionality required evolved with the project as the objectives from the Gantt chart were pursued. Overall, this hybrid approach led to a very successful and quick development cycle for a single developer working independently. There was an element of self-management because of this total independence from others, which was an eye-opening experience to the realities of the software development life-cycle.

The main lessons learned from this project were the time constraints of software development, as the feature that you are trying to develop often takes up more time and resources than anticipated. This led to moments where certain features had to be re-prioritised over others that ended up requiring less resources committed to them. The technique of reviewing functions proved extremely useful in this regard, as functions which replicated functionality elsewhere could be removed or adapted for other purposes, thus reducing the total number of lines of code and libraries in the entire project. This also helped when reviewing the path of the process when the project performed its various functions during operation. This also allowed for the identification of functions that could be unit tested as part of the testing phase once the development was complete. The use of Git as the main version control tool for this project also proved to be useful in two main ways, one being the backing up of the project itself to an easily accessible source to be downloaded and tested anywhere. The easily accessible nature of Git was particularly useful when setting up different virtual machines for the network security and usability tests, as the project could be clones from GitHub with one command, and then adjusted to fit the exact configuration of the machine it was installed to. Gits other useful feature for this project was its version control mechanisms, that not only allowed for publicly available documentation of the development process, but also for creating a branch for the testing phase for integration into Ssh.

If this project were attempted again from the beginning, a stronger focus on the actual requirements and specifications would be made, as this proved to be an area of unexpected obstacles to overcome in order to achieve something that was initially though trivial. This included features such as the integration with MySQL (Oracle Corporation, 2019), and the interfacing of USB devices across python

sockets, which was unfortunately not completed in time. If the focus on grasping the core requirements and exploring the work that needed to be done for those requirements was explored earlier, there would have been more time for some of the more desired features, such as a graphical user interface or web server integration. There were a few areas of development where the project could have gone to if they had been approached as well, for example multi-threaded execution of the MySQL (Oracle Corporation, 2019) connection libraries would have increased performance, as well as increasing the speed of the key generation and checking functions due mainly to their mathematically heavy features. In terms of project management, a more agile system of project management would be more suitable, particularly as this project was made with a single developer who was self-managed and motivated. The unnecessary processes involved in the Gantt chart approach was more time consuming and didn't have the benefit of informing others in a team as to the progress of the work, as there were no other team members to generate tickets or scrum meet with.

Evaluation of tools used

Python as a language for writing this project proved perfectly suitable, as it has the right level of readability for non-technical users, combined with a very flexible and modular structure. The way that it can import other python files as modules down to individual functions came in very useful when writing functions in other parts of the project. The language also has a plethora of modules pre-made that can be imported to perform functions that otherwise have had to of been written in other languages, for example the MySQL (Oracle Corporation, 2019) module. The PyCharm IDE (JetBrains s.r.o., 2018) proved an excellent environment to develop python in, as it had useful features such as continuous PEP-8 (van Rossum, et al., 2001) structure checking, GitHub integration for version control and a built-in unit testing module for the testing phase. With regards to research sources, both the ACM Digital Library and IEEE Xplore proved excellent sources for up-to-date materials on all topics that this projects research element touched on, and even provided ideas to help improve the concept of this project further and to help explain the objectives and achievements of the project when compared to the references works.

Final thoughts

This exploration of the current state of multi-factor authentication, and the subsequent improvements that were developed, are put forward as an observation of where current solutions perform well and how this project can enrich this selection of tools further. The ideas that were put forward appear to be truly ground-breaking, with some concepts that were never put together in the context of multi-factor authentication it would seem. The idea of distributing security data is something that could lead to a significant decrease in data leaks should it be implemented properly as discussed in this report, as well as the increased security mechanisms that distribution allows for such as the inclusion of physical security data storage devices for further separation and isolation of sensitive data to strengthen integrity and confidentiality of authentication keys.

Word count: 12,702

Bibliography

ACKERMAN, P. & HUMBERTO, M., 2014. *Impediments to Adoption of Two-factor Authentication by Home End-Users.* [Online]
Available at: https://www.sans.org/reading-room/whitepapers/authentication/impediments-adoption-two-factor-authentication-home-end-users-37607
[Accessed 30 10 2018].

Aivat, A., 2015. Introducing Thonny, a Python IDE for learning programming. *Koli Calling '15 Proceedings of the 15th Koli Calling Conference on Computing Education Research,* Volume 2015, pp. 117-121.

Allegretta, C., 2018. *Nano's ANOther editor, an enhanced free Pico clone.* 2.9.3 ed. s.l.:Free Software Foundation.

Apache Software Foundation, 2018. *Apache Subversion.* [Online]
Available at: https://subversion.apache.org/
[Accessed 14 11 2018].

BRAINARD, J. et al., 2006. *Fourth-factor Authentication: Somebody you know.* Alexandria, Virginia, United States of America, ACM New York, pp. 168-178.

COLNAGO, J. et al., 2018. "It's not actually that horrible"; Exploring Adoption of Two-Factor Authentication at a University. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems,* April, Issue 456, pp. 1-12.

Combs, G., n.d. *Wireshark - Go Deep,* s.l.: s.n.

DEARI, R. et al., 2018. *Analysis And Comparison of Document-Based.* Varna, Bulgaria, IEEE.

Discord Inc., 2019. *Setting up Two-Factor Authentication - Discord.* [Online]
Available at: https://support.discordapp.com/hc/en-us/articles/219576828-Setting-up-Two-Factor-Authentication
[Accessed 01 04 2019].

DUCKLIN, P., 2013. *Anatomy of a password disaster – Adobe's giant-sized cryptographic blunder - Naked Security.* [Online]
Available at: https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/
[Accessed 20 10 2018].

Eclipse Foundation, 2018. *Eclipse IDE for Eclipse Committers | Eclipse Packages.* [Online]
Available at: https://www.eclipse.org/downloads/packages/release/2018-09/r/eclipse-ide-eclipse-committers
[Accessed 14 11 2018].

GAURAVARAM, P., 2012. *Security Analysis of salt||passwrd Hashes.* Kuala Lumpar, Malaysia, IEEE.

GitHub, Inc., 2018. *GitHub.* [Online]
Available at: https://github.com/
[Accessed 12 11 2018].

Google LLC, n.d. *Google 2-Step Verification.* [Online]
Available at: https://www.google.com/landing/2step/#tab=how-it-protects
[Accessed 27 10 2018].

Hern, A., 2013. *Did your Adobe password leak? Now you and 150m others can check.* [Online]
Available at: https://www.theguardian.com/technology/2013/nov/07/adobe-password-leak-can-check
[Accessed 10 March 2019].

JetBrains s.r.o., 2018. *PyCarm 2018.2.4 (Professional Edition).* Build #PY-182.4505.26. ed. Prague: JetBrains s.r.o..

kgretzky, 2018. *GitHub - kgretzky/evilnginx2: Standalone man-in-the-middle attack framework used for phishing login credentials along with session cookies, allowing for the bypass of 2-factor authentication.* [Online]
Available at: https://github.com/kgretzky/evilginx2
[Accessed 30 10 2018].

KRISHNAN, R. & JUMAR, R., 2014. Securing User Input as a Defense Against MitB. *Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing,* p. Article 50.

Levin, D. V., 2018. *A Linux-PAM Page.* [Online]
Available at: http://www.linux-pam.org/
[Accessed 11 02 2019].

Microsoft Corporation, 2018. *How to use the Microsoft Authenticator app.* [Online]
Available at: https://support.microsoft.com/en-us/help/4026727/microsoft-account-how-to-use-the-microsoft-authenticator-app
[Accessed 27 10 2018].

Nagel, W., 2006. *Subversion: Not Just for Code Anymore.* [Online]
Available at: http://delivery.acm.org/10.1145/1120000/1119455/8596.html?ip=185.206.227.167&id=1119455&acc=ACTIVE%20SERVICE&key=BF07A2EE685417C5%2EA26EB5666899191E%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1542239761_966fb22e50ccbcd105658ce634dc234b
[Accessed 14 11 2018].

Oracle Corporation, 2019. *MySQL,* s.l.: Oracle Corporation.

Orso, A. & Rothermel, G., 2014. Software testing: a research travelogue (2000–2014). *FOSE 2014 Proceedings of the on Future of Software Engineering,* 31 May.pp. 117-132.

Purcell, S., 2017. *PyUnit - Python Wiki.* [Online]
Available at: https://wiki.python.org/moin/PyUnit
[Accessed 10 03 2019].

Reynolds, J. et al., 2018. *A Tale of Two Studies: The Best and Worst of YubiKey Usability.* San Francisco, California, United States of America, IEEE.

Ross, B. et al., 2005. *14th USENIX Security Sympsoium - Technical Paper.* [Online]
Available at: https://www.usenix.org/legacy/event/sec05/tech/full_papers/ross/ross_html/
[Accessed 16 11 2018].

Shirvanian, M., Jareciky, S., Krawcykz, H. & Saxena, N., 2017. *SPHINX: A Password Store that Perfectly Hides Passwords from Itself.* s.l., s.n.

van Rossum, G., Warsaw, B. & Coghlan, N., 2001. *PEP 8 - the Style Guide for Python Code..* [Online]
Available at: https://www.python.org/dev/peps/pep-0008/
[Accessed 05 11 2018].

Van Styn, H., 2011. *Git.* [Online]
Available at:
http://delivery.acm.org/10.1145/2030000/2020790/10971.html?ip=185.206.227.167&id=2020790&acc=ACTIVE%20SERVICE&key=BF07A2EE685417C5%2EA26EB5666899191E%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1542234369_6427234d32c5a9348761c1230e5181f6#URLTOKEN#
[Accessed 14 11 2018].

Wang, Z. & Cao, L., 2013. *Implementation and Comparison of Two Hash Algorithms.* Shiyang, China, IEEE.

Yahoo, n.d. *Set up, use, and manage Yahoo Account Key to sign in without a password | Yahoo Help - SLN25781.* [Online]
Available at: https://help.yahoo.com/kb/SLN25781.html?guccounter=1
[Accessed 27 10 2018].

Zhang, C. et al., 2018. Lexs 2017: bulding reliable software in python. *Journal of Comuting Sciences in Colleges,* 33(6), pp. 124-134.

Appendices

Appendices 1 – Unit testing script used in the software unit tests

unit_testing.py

```
from unittest import TestCase
from data_handler import sql_inject_check
from data_handler import identifier_check
from crypto_handler import generate_key
from crypto_handler import fingerprint_data

class TestSql_inject_check(TestCase):

    def test1(self):
        array = []
        with open('../Files/sql_inject_strings.txt', 'r') as f:
            for line in f:
                array.append(line)
            f.close()

        c = int(0)
        for element in array:
            c = c + 1

        c = c - 1
        while c is not -1:
            i = array[c]
            assert sql_inject_check(i) is True
            c = c - 1
        self.assertRaises(Exception)

    def test2(self):
        array = []
        with open('../Files/common_names.txt', 'r') as f:
            for line in f:
                array.append(line)
            f.close()

        c = int(0)
        for element in array:
            c = c + 1

        c = c - 1
        while c is not -1:
            i = array[c]
            assert sql_inject_check(i) is False
            c = c - 1
        self.assertRaises(Exception)


class TestIdentifier_check(TestCase):
    def test0(self):
        assert identifier_check("a", "a") is True
        self.assertRaises(Exception)

    def test1(self):
        assert identifier_check('a', 'a') is True
        self.assertRaises(Exception)

    def test2(self):
```

```
        assert identifier_check(str(1), str(1)) is True
        self.assertRaises(Exception)

    def test3(self):
        assert identifier_check(str(0.12345), str(0.12345)) is True
        self.assertRaises(Exception)

    def test4(self):
        assert identifier_check("a", "b") is False
        self.assertRaises(Exception)

    def test5(self):
        assert identifier_check('a', 'b') is False
        self.assertRaises(Exception)

    def test6(self):
        assert identifier_check(str(1), 1) is False
        self.assertRaises(Exception)

    def test7(self):
        assert identifier_check(str(0.12345), 0.12345) is False
        self.assertRaises(Exception)

class TestGenerate_key(TestCase):
    def test0(self):
        assert generate_key('test') ==
'9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'
        self.assertRaises(Exception)

    def test1(self):
        assert generate_key("test") ==
'9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'
        self.assertRaises(Exception)


class TestFingerprint_data(TestCase):
    def test0(self):
        assert
fingerprint_data("9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b
0f00a08") == 'e9e8fa9602f37a97a0e3f588c8e49e1bbb756706'
        self.assertRaises(Exception)

    def test1(self):
        assert fingerprint_data('test') ==
'a94a8fe5ccb19ba61c4c0873d391e987982fbbd3'
        self.assertRaises(Exception)


if __name__ == '__main__':
    unittest.main()
```

Appendices 2 – All results from user experience timing tests

Distributed Login

0m44.061s

0m28.750s

0m15.671s

1m0.133s

0m32.913s

Email

0m17.648s

0m32.296s

0m39.586s

0m20.380s

0m36.272s

Google Authenticator

0m13.721s

0m15.863s

0m24.867s

0m12.690s

0m8.354s

SMS

4m8.287s

0m22.522s

0m17.138s

0m23.001s

0m18.304s

Appendices 3 – Source Code Link

https://github.com/JEdwards27/Dist_Login