Final Report

Ryan Olivieri, Chuong Trinh
Texas A&M University
{rco16, ctrinh}@tamu.edu

I. Introduction

In today's world the Internet has become ubiquitous and ever growing into every part of one's daily life. This integration of the Internet into our daily lives has brought great benefits and convenience, but has also came with difficult security problems. Users often have tens if not hundreds of different online accounts for various websites ranging from entertainment, social media, hobbies, work, school, to financial accounts [5]. With each of these sites being protected by a username and a password, a user needs to keep track of these passwords. With a username and password often the only requirements to authenticate as a user for online accounts a compromised username and password combination allows an attacker complete control of the account. This can lead to defamation in which posts are made as the user that can ruin the users reputation. Also a compromised account can have have massive financial losses as an attacker can electronically transfer funds to another bank account out of country and it can be hard if not impossible for a victim to recover the funds.

II. Background

Over the years tools have been created to help a user keep track of their various usernames and passwords for each of the websites they have an account with.

One option is a small notebook that all of the information for each website is written down in and the notebook is stored in a secure place such as a safe or locked drawer when not in use. This has the benefit of allowing a user to use complex and different passwords for each website and not have to use password reset since they forgot the complicated password. It however does have the disadvantage being that if an attacker has physical access to the notebook they can comprise all of the user's accounts and the notebook is not very portable if the user wants to keep it secure.

Another option is to use a password manager which is a piece of software that stores usernames and passwords for each website in a digital vault that is secured with a master password. Different styles of password managers exist such as desktop or mobile app versions, but also cloud password managers exist that support access to the password manager from any computer [7]. One example is LastPass which integrates into a web browser through browser extensions [6]. The benefits are that the passwords are secure from attackers accessing them assuming that the master password to secure the password manager is strong enough. The disadvantages to a password manager is that it is only as useful as the user makes it to be. If a user doesn't use the password manager then no benefits are gained. The main reason a user doesn't use a password manager is that it is inconvenient to use since extra steps are required to log into an account.

III. Problem and Motivation

Password managers are a great solution to providing strong and unique passwords for each account a user has, but they have yet to gain wide spread use due to the difficulties of use and the inconvenience to the user.

One of existing solutions is to use the password manager that helps users store and organize their passwords. However, the limitation of this method is that not many users actually use it. In addition, the users have to rely their trusts on the tool, and since the system requires a user to enter their password for opening their entire passwords, this still exposes vulnerabilities to be attacked by key logging or acoustic cryptanalysis [3].

To overcome the problem, alternative solution existed on the Android market today is Enpass Password Manager. The main purpose of this application is to store all kinds of important credentials. Unlike us, the system utilizes a master password that is stored in user's device and it can be synced through their cloud services [9]. The problem with having a master password key is that there is no way to recover the master password if the user lost it. Hence, the user has to either remember the master password or store it somewhere and this expose security risks as we discussed earlier. Even worst, if an attacker can discover the master password, he has access to all other stored password.

Another popular password manager is LastPass [6] which doing very similar thing like Enpass but different authentication method. The system monitors and captures whenever the user is about to type into a password field and scans to see if it has information associated with the account. Hence, it can fill the password with a

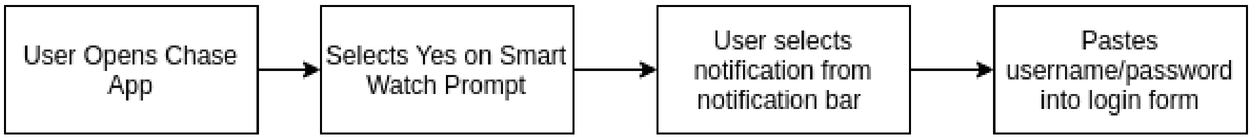

User Opens Chase App
Selects Yes on Smart Watch Prompt
User selects notification from notification bar
Pastes username/password into login form

Fig. 1. Example of User interaction with current password manger applications

single button. The downside of this is that it needs to use fingerprint as the master key. However using fingerprint for password has explored a serious vulnerability [1][2]. A higher secured level is Apply Pay which is activated by user typing one-single time generated 4-digits password on the watch, and it generates one-time encrypted number to match with the issuing bank's database to complete the payment without transmitting credit card information to merchants [4]. However, this is only dealt with payment transaction but not auto filling passwords as our objectives.

Motivated by current limitations and a high demand of a truly convenient and secured password application, we propose to investigate our application called SmartPass to provide the seamlessly experience of managing passwords without remembering the master key every time the user want to use it or using high risk fingerprint authentication.

IV. Proposal

Through the combination of using a smart watch and a smart phone a user can be provided a seamless account login sequence that provides a strong level of security. This can be achieved by creating a smart watch app and accompanying smart phone app that stores the account information for each account the user has. The password information will be split between the two devices, so that if one device is stolen or lost the the one device is not enough to compromise the accounts.

An high level workflow can be seen in figure 1. This workflow shows how SmartPass is able to know that the user opened an app that they have an associated account for and then to automatically display a prompt for the user on the smart watch to log in or not. When the user selects to login the smart watch will securely transmit the half of the password to the smartphone which then creates a notification for the user to click on for the username/password to be added to the clip tray. Thus the user was able to login to their account with a complicated and strong password without being burdened to type the password in character by character or have to open a separate password manager app on their smart phone and enter the master password to login.

A. *Contribution*

A streamlined convenient password management system that utilizes strong (long, random, special characters) passwords to log a user into their accounts without the user ever having to type a password in. Instead the possession of paired unlocked smart watch and the unlocked smart phone are used as authentication of the user.

B. *Security*

Smart watches that run Android Wear OS have the capability of locking when they are removed from the users wrist which allows this protection to be turned on and not interfere with the user since they only need to unlock it each morning when they put the watch on. If the watch was stolen off the users wrist it would be locked preventing an attacker from accessing it.

The lock for an Android phone would be required to be used when using this password manager and the lock setup for an Android phone is trusted by Google since it allows payments to be made via Android Pay by having the phone unlocked.

Google states that the communication between the smartwatch app and the accompanying android app is secure. Also an android app has the option to have private storage that only the app can access. Also full disk encryption has become common with newer android phones which adds an extra level of security.

V. Workflow

The workflow of the SmartPass app can be seen in figures 2 to 11. The workflow shows how the user adds accounts and how they are able to login to their accounts.

The first task for a new user is to add their accounts to the password manager. The entry for this can be seen in figure 3. The user is able to select the application they want to save the login for and enter the username and password as well. When the user clicks add account half of the password is sent to the watch and stored in the database on the watch as shown in Figure 4. The user can add as many accounts as they wish to the password manager.

Next the workflow for how the user uses the app will be shown. The userthe database entry on the watch running android Wear. The watch has a full database running. opens their app of choice which can be seen in figure 5. When this happens our app is able to detect that an app that has been registered by the user with login information has been opened. This can be seen in figure 6 that shows the foreground applications. After a registered app has been detected a notification is sent to the watch which can be seen in figure 7. The user then can swipe up on the notification on the watch to show the full notification as seen in figure 8. After this the user can swipe to the left to show the yes button to login as shown in figure 9 or they can swipe left twice to select the no button and not login. After this the mobile device reassembles the mobile and wear password into the original password and creates notifications on the mobile to copy and paste the username and password as seen in figure 10.

A diagram showing the flow through the app use can also be seen in figure 12. This shows the logic used to make sure that the user has the smart watch paired and in the vicinity of the mobile phone. It then shows how the user uses the smart watch to choose whether or not to login to the account the smart watch displays.

VI. Implementation

Android was chosen as the development environment for this prototype due to its availability and wide array of development tools and libraries for rapid development. Also by using Android and leveraging a smart watch that uses the Android Wear OS the two apps will be able to interact seamlessly for a better user experience. The Android Wear OS on the smart watch is very similar to the Android OS running on the mobile phone and supports many of the same operations. Also other watch specific functions are supported such as a heart rate monitor or other gesture related actions. Overall, this environment is great to design an application in and allows a broad user base to be reached due to the popularity.

A. Detecting foreground applications

One of challenges in the main flow of our application is the ability of detecting foreground running applications. This functionality has been widely used in Android development for user apps in 4.0 and older. By simply calling the command *"getRunningAppProcesse()"* in **ActivityManager** class, developers can see all the applications are running including background apps. However, since android 5.0 and above, Google prohibited this feature except for certified system applications for security purposes. We adopted several ways of word-around approaches and only one approach, which is developed by Jared Rummler [8], works as expected for both 5.0 and 6.0 version. The author also mentioned that the technique may not work on newly developed 7.0 android version. It obtains the list of running application by parsing the output from shell command "toolbox ps" which does not require root permission and has been existed since android 2.0.

B. Communication

Communicating between the mobile phone and the smart watch involved a variety of different intricacies. A Google Messaging API exists that handles creating and closing the Bluetooth connection between the mobile and watch. By using this API and intent-filters that let the wear and mobile to watch for messages with specific paths to arrive the mobile and watch are able to communicate commands successfully. The path is simply a string value that gives each message sent a unique identifier to be sorted upon once it arrives at the OS on the watch or mobile. Then the OS can can notify the app that was waiting for the communication and the app can process the data.

The process of notify the watch to display the login notification involves using a Data Item and changing the value of the Data Item on the mobile app and also including which app the user wants to login to. When the watch receives this value it notifies the background service that is watching for data changes with a specific path value. This background service can then launch the custom notification on the watch. The watch also can communicate back to the mobile device through the Google Messaging API. This allows the half of the password on the wear and other data such as confirmations to be sent to the mobile device.

C. Storage

A SQLite database was created to store the user's account information in a table. This allows persistent storage as well as allowing the data to be easily queried. Storing the data in a serial JSON format would require loading the whole storage file each time and it residing in memory. This is not necessary and could cause performance issues by the increased workload to login to one account. The database option is also available on the watch which allows for a similar implementation for both the watch and mobile device.

D. Account syncing

Keeping the accounts on both the mobile and watch in sync is important for the watch to function correctly. To facilitate this the mobile device is able to send a command to the watch to delete an entry from its records when the user selects to delete an account.

VII. Improvement

A. Encrypting password

To further increase the security protection for our application, we add an extra security layer to ensure the communication between the smart watch and smart phone by encrypting the password sent over to the watch. We adopt the extra library for key encryption

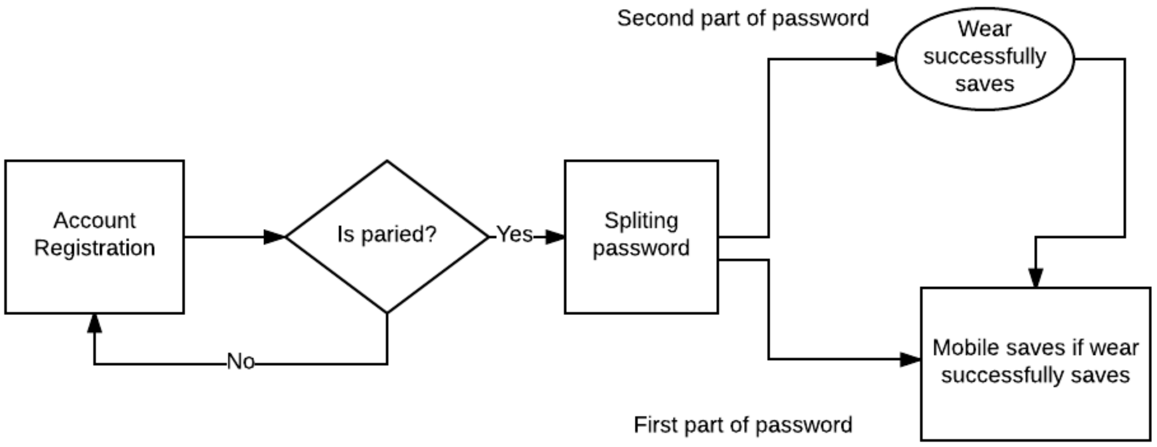
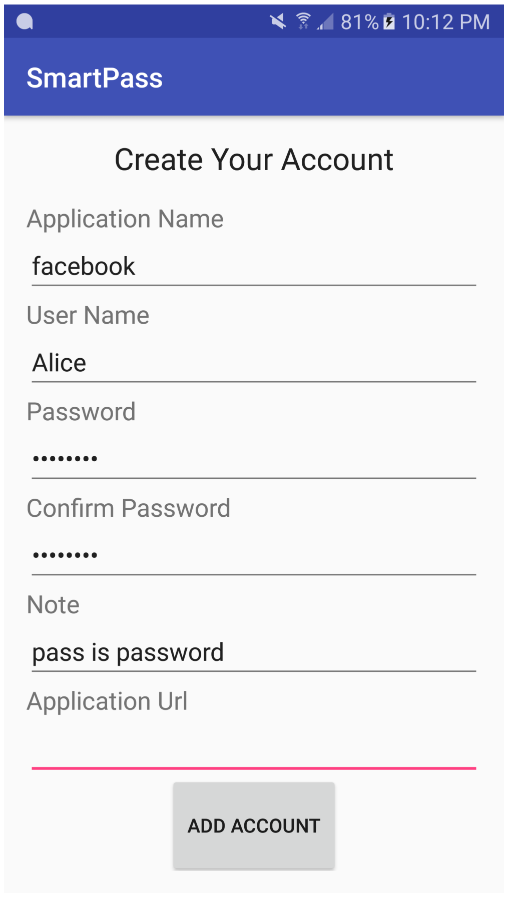


Fig. 2. Registration workflow



Application Name	facebook
User Name	Alice
Password	••••••••
Confirm Password	••••••••
Note	pass is password
Application Url	

Fig. 3. Adding new accounts

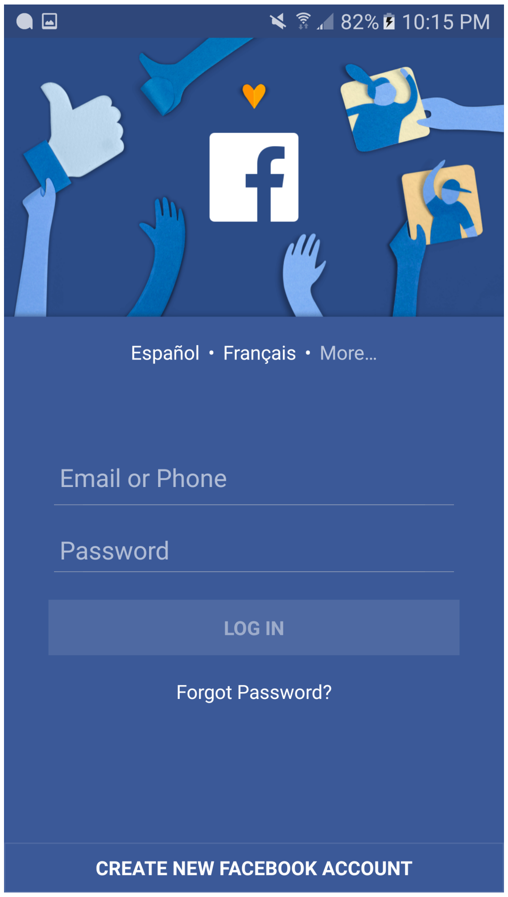
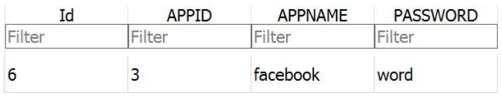
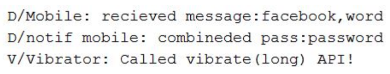


Fig. 5. Open Facebook app



Id	APPID	APPNAME	PASSWORD
Filter	Filter	Filter	Filter
6	3	facebook	word

Fig. 4. Wear database



```
D/Mobile: recieved message:facebook,word
D/notif mobile: combineded pass:password
V/Vibrator: Called vibrate(long) API!
```

Fig. 6. Detecting foreground app

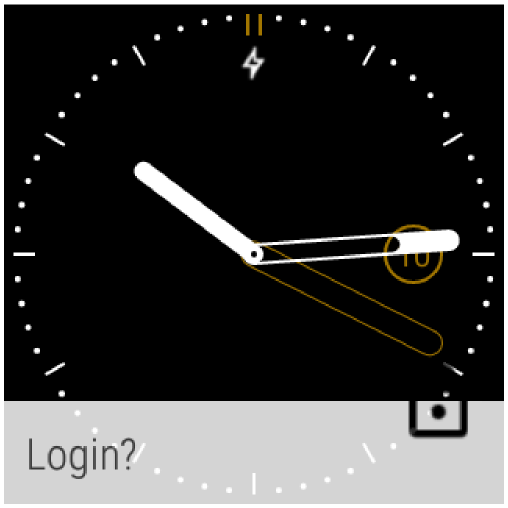


Fig. 7. Notification Popup on the watch

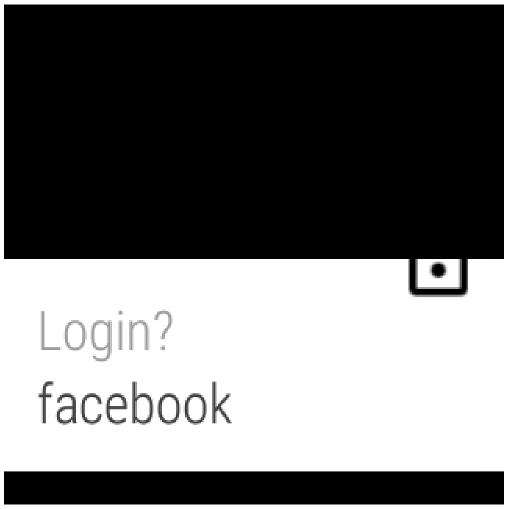


Fig. 8. Notification Popup on the watch

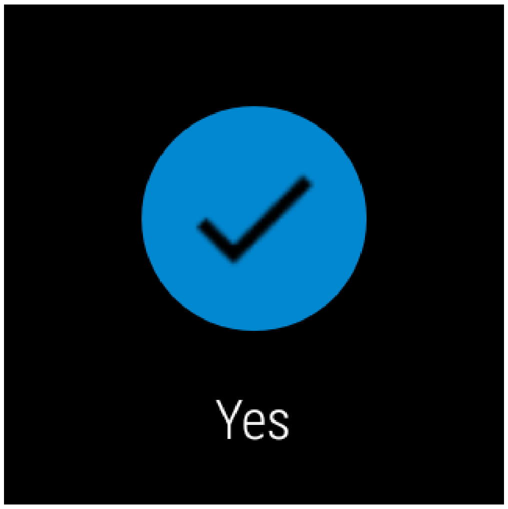
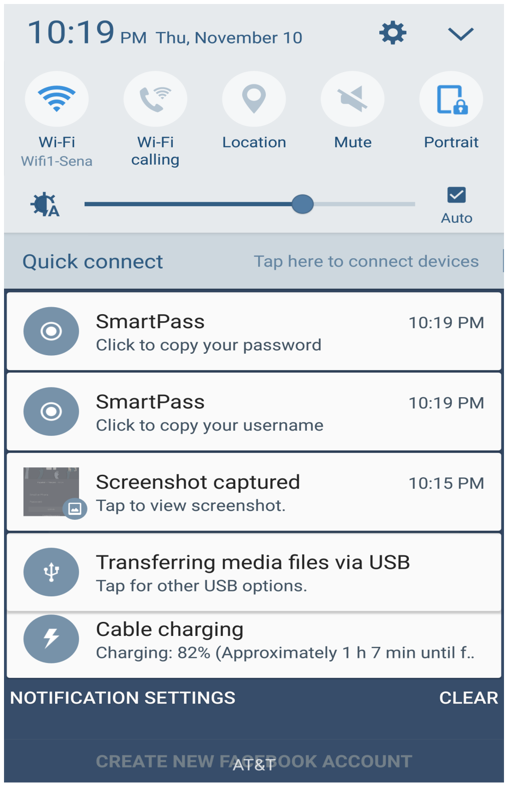


Fig. 9. User can either choose Yes or No



10:19 PM Thu, November 10

SmartPass 10:19 PM
Click to copy your password

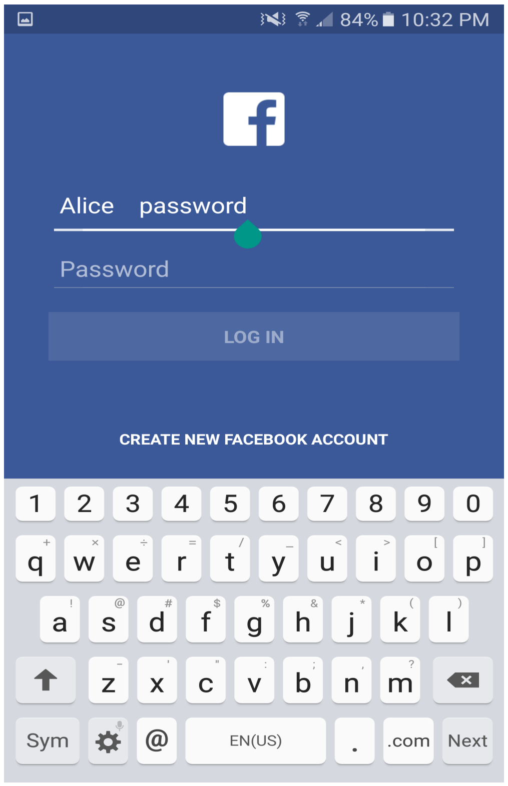
SmartPass 10:19 PM
Click to copy your username

Screenshot captured 10:15 PM
Tap to view screenshot.

Transferring media files via USB
Tap for other USB options.

Cable charging
Charging: 82% (Approximately 1 h 7 min until f..

Fig. 10. Password and user name are copied to ClipBoard



84% 10:32 PM

Alice password

Password

Fig. 11. User can paste the user name and password into the fields

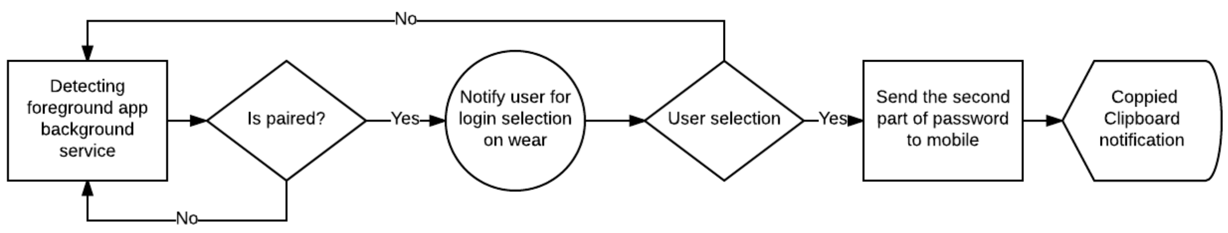
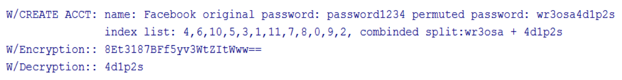

No
Detecting foreground app background service
Is paired?
Yes
No
Notify user for login selection on wear
User selection
Yes
Send the second part of password to mobile
Coppied Clipboard notification

Fig. 12. Running application workflow

using AES-128 in CBC mode. The secret key is randomly generated only once whenever the application is installed. In this case, the attackers will not be able to decode the source code to find the key. The key is stored in the private local data which has been claimed to be ”secured” by Google.

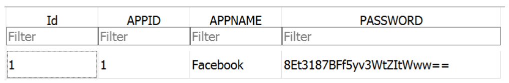
B. Permuting password

In order to make it harder to guess by the attackers, we incorporate password permutation to increase the difficulty in brute-force passwords. The whole process can be seen as following: we first randomly shuffle all the characters in the password for every single application registration; the list including original indexes is stored safely in private local data; we split the permuted password and apply encryption on the part stored in the wear database. An example of storing password ”password1234” is demonstrated in Fig. 13 and Fig. 14



```
W/CREATE ACCT: name: Facebook original password: password1234 permuted password: wr3osa4d1p2s
               index list: 4,6,10,5,3,1,11,7,8,0,9,2, combinded split:wr3osa + 4d1p2s
W/Encryption:: 8Et3187BFf5yv3WtZItWww==
W/Decryption:: 4d1p2s
```

Fig. 13. Permuted part of password sent to the wear is encrypted and it can be decrypted when it is sent back to the mobile



Id	APPID	APPNAME	PASSWORD
Filter	Filter	Filter	Filter
1	1	Facebook	8Et3187BFf5yv3WtZItWww==

Fig. 14. Encrypted-permuted wear-password in wear’s database

VIII. Evaluation

A. Environment

In SmartPass’s current implementation the following requirements are needed for proper operation. An Android phone running at least Android 5.1, an Android Wear OS smart watch, the mobile and watch paired to each other, a lock screen enabled for mobile and the watch, and the SmartPass Android app installed.

B. Security

Since the workflow depends heavily on the authentication pair activity between the mobile and the wear, our application ensure the core value of the app by enforcing the presents of both unlocked devices. We also integrated encrypting password and permuting password mechanisms to ensure the security protection of passwords but still maintain the convenience aspect of our application.

C. Mobile

For the mobile version, our application is succeed to handle basic user registration procedures, detect foreground application, notify the wear for user selection, and allow the user to copy their credentials through a click on the notification bar. Due to repeatedly checking foreground app in our background services, our application still run and consume battery even if the application is closed. However, this does not cause overhead or huge battery consumption as we have tested, and it is obviously seen that only running shell command call and parsing outputs will not consume too much battery as compared to other activities from other apps such as network communication.

D. Watch

For the watch version, our application is succeed to listen to requests from the mobile when an registered app is detected, handle user selection and communicate with the mobile accordingly to the user selection. The app will not consume battery on the wear since its functionality is to handle package from the mobile and send back the request account data. From the memory standpoint, this is not the issue because the database is relatively small and the app won’t store anything else except for records for one database.

IX. Future Work

A. Password entry

The current methodology of using the system clip tray to store the username and password of the account to login when the user clicks on the notification for the respective attributes does work, but is less than ideal. There are at least two problems with this approach. The first is the lack of smoothness this process requires of the user. By having to open the notification tray, select the username, click and hold on the username text box, and then selecting paste all to enter the username for their account is quite a long process. Then this process needs to be repeated by the user for the password. A streamlined experience for the user is highly preferred and a requirement for the users to continue to use the password manager.

The second problem with this approach is the security issues of saving the values to the system clip tray. In an ideal world this would be safe because only the only person to see the values in the clip tray would be the user which is fine since it is their credentials. Unfortunately, malicious actors exist and are actively attacking and probing android systems for vulnerabilities. An attacker could entice a user to download an app that watched for clip tray updates and then saved the values that are in the clip tray. An attacker doing this could sniff the username and password in plain text from the user. On top of having the username and password the attacker could also know which app the user is logging onto by using a similar approach that the SmartPass password manager uses to find the application currently active. This would allow an attacker to compromise a user's account.

A potential solution to the security problem would be to clear the contents of the clip tray a set amount of time after they were added to the clip tray. This would prevent the credentials from staying in the clip tray history for a long period of time where they could be viewed by other apps or even other people who use the user's phone. This however would not prevent another app that is actively snooping the credentials on the clip tray. This is due to the clip tray having its contents in plain text. It may be possible to ask Google to modify the clip tray to allow for content that is user readable only. If this was possible then the user would we able to paste the contents of the clip tray, but another app wouldn't be allowed to read the value of the clip tray. With this approach clearing the clip tray 30 seconds after the user adds the username or password to the clip tray would also prevent another user from later viewing what the credentials are by viewing the clip tray.

Another solution to the problem would be following what the Keepass2Android app currently does. The Keepass2Android is a traditional password manager that is on a phone. A user unlocks the password database with a master password and then selects an account to login too. When it does it creates a notification. Then the user navigates to the chosen app to login to and the user then taps the Keepass notification. Then the user is prompted to switch to a custom Keepass keyboard. This keyboard which can be seen in figure 15 allows the user to move the text cursor to the username field and simply tap the username button on the keyboard. Then the user can move to the password field and click

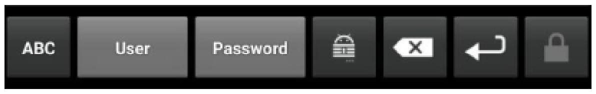


Fig. 15. Example of password keyboard from Keepass2Android

the password button on the keyboard. The app also has the functionality of auto entering the username and password once the custom keyboard is selected if the app the user is logging into supports the functionality.

This design solves both the issues that the clip tray option has. The user experience is greatly simplified to allow a convenient work flow. The security is also much higher since the username and password are not placed in a system wide clip tray that another app could sniff data from. Instead by using the keyboard to input the username and password the credentials are safe. This relies on the security of the android OS to prevent other apps from sniffing the data that is sent from the keyboard to the application which is a reasonable assumption considering all secure applications depend on the OS to prevent other apps from sniffing user input with the respective applications.

B. Mobile Browsing Support

While many online accounts have a mobile specific application that a user can use there is also a great number of accounts that are only accessible through a web browser. In a case where a native app does not exist the user would would have a difficult time using our password manager in its current state. One option would be to allow the user to use the mobile SmartPass application that is used to create and manage accounts to launch the notification to the watch for a specific account the user chose. This would require an awkward work flow for the user to access their login credentials for an account accessed through a mobile account.

The reason that the mobile web browser is different than a native app is due to how android presents what app is currently running on the phone. Android allows the foreground app to be read by a background service, but does not allow the user to know more specific information about an application. This means that the current action the user is doing on the app is not known, but purely what app is currently running. This leads to a problem when trying to automatically display a login notification for an account when the associated login page is visited. An app other than the web browser does not know what web page the user is currently visiting and this is considered a benefit since it protects the users privacy from malicious applications. Without knowing which web page is currently being viewed by the user our app would be unable to present the user with the login notification on their watch when the visit the mobile login website for their account.

A possible solution would be an application that runs on the watch could allow a user to access their accounts easily and quickly. The workflow for logging into a mobile browser based account would involve the user visiting the website on the mobile phone. Then the user would open the SmartPass app on their watch and scroll to the respective account select it. This would then send the half password stored on the watch to the phone and the phone would create the notification for the user to copy the username and password to the mobile web page.

C. *Exporting accounts*

Backing up a user's accounts is important if the SmartPass app is to be relied on a day to day basis as it is intended. This has yet to be implemented, but would allow the user to export the accounts from the mobile and watch to a secure format such as the Keepass database format. This would also enable a user to use this on a desktop or laptop as a conventional password manager setup.

X. Conclusion

Our Smartpass application has been demonstrated to be a simple, secure, and convenient password manager application by providing abilities to authenticate with unlocked paired smartwatch and phone, store encrypted part of the password on the smartwatch and half on the phone in which both are originally permuted, and finally, automatically display notification on smartwatch when a registered app is opened on the phone.

Smart watch and a smart phone linked together to login into accounts is more streamlined than other solutions currently available. This mainly hinges on not having to type in a secure password on a mobile phone, since a secure password needs to be long or have a variety of special characters which are hard to type on a mobile phone. Instead the possession of unlocked paired watch and mobile phone allow the user to authenticate themselves and access store accounts. Further, the login procedure is simple and easy for the user, since when the app to login on is opened on the mobile phone a notification asking if the user would like to login is on the user's watch automatically. This allows the user to easily use SmartPass which allows strong and secure passwords to be used without the hassle of the user manually remembering and typing the passwords in.

References

[1] Think your fingerprint sensor is impervious to criminals? think again.

[2] Why you should not use the new smartphone fingerprint readers.

[3] A feasible and cost effective two-factor authentication for online transactions. 2010.

[4] Michael deAgonia.

[5] Shirley Gaw and Edward W Felten. Password management strategies for online accounts. In *Proceedings of the second symposium on Usable privacy and security*, pages 44–55. ACM, 2006.

[6] LastPass, oct 2016.

[7] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 465–479, 2014.

[8] Jared Rummler, 2015.

[9] Sinew Software Systems, 2015.