Behavioral Biometric Authentication in Android Unlock Patterns through Machine Learning

José Torres[1], Sergio de los Santos[1], Efthimios Alepis[2] and Constantinos Patsakis[2]
[1]*Telefónica Digital España, Ronda de la Comunicación S/N, Madrid, Spain*
[2]*Department of Informatics, University of Piraeus, Karaoli & Dimitriou 80, Piraeus, Greece*

Keywords: Android, Authentication, Biometrics.

Abstract: The penetration of ICT in our everyday lives has introduced numerous automations, and the continuous need for communication has made mobile devices indispensable. As a result, there is an ever-increasing deployment of services for which users need to authenticate. While the use of plain passwords is the default, many applications require higher standards of security, such as drawn patterns and fingerprints, used mostly to authenticate users and unlock their smart devices. In this work we propose a biometrics-based machine learning approach that supports user authentication in Android to augment native user authentication mechanisms, making the process more seamless and secure. Our evaluation results show very high rates of success, both for authenticating the legitimate user and also for rejecting the false ones. Finally, we showcase how the proposed solution can be deployed in non-rooted devices.

1 INTRODUCTION

Smartphones have radically changed the way of modern living providing communication to users at any time and place, as well as access to an unlimited number of services and online applications. As a result, these smart devices store and process vast amounts of sensitive information, ranging from personal, to financial and professional data. Despite the introduction, by a significant number of software companies and also from the mobile OSes, of several biometric-based authentication mechanisms such as fingerprints and face recognition, only a small fragment of users actually uses them. The reasons behind this lack of adoption can be attributed to several factors including skepticism regarding who may process this "unique" and personal data, the impact of this data been leaked, as well as their usability and efficiency. As a result, the vast majority of smartphone users are still using the well-known PIN/Pattern authentication method, with the "pattern" method being the most used user-authentication method for smartphone unlocking to date (Malkin et al., 2016).

An important aspect that has to be considered in this direction is that due to their use, this authentication method has several drawbacks. The most obvious one is that since users have to continuously authenticate to the device, quite often the PIN/Pattern that users tend to have is relatively "easy". Moreover, due to the way that users have to perform the authentication, shoulder surfing cannot be avoided. Even by excluding shoulder surfing attacks, other methods could be utilized in order for an attacker to gain information about the victim's pattern, such as malicious apps with "drawing over other apps" permissions and analyzing finger traces on smartphone screens attacks.

Towards this direction, smartphone locking and consequently unlocking is an issue that has drawn much attention from all over the world, since it involves private user data and consequently affects user rights. Indeed, back in 2016, in the "Apple - FBI encryption dispute", (Wikipedia, 2018), the FBI asked Apple to create and electronically sign new software that would enable the FBI to unlock a work-issued iPhone 5C, which was recovered from one of the shooters of the San Bernardino terrorist attack in December 2015, in California. Nevertheless, it has to be outlined that modern smartphones have increased their security and may now wipe the device once many failed authentication attempts are made.

Main Contributions. This paper focuses on providing a robust solution that utilizes user behavioural biometrics through Machine Learning to improve smartphone locking solutions. This approach can be adapted to a variety of other use cases, such as smartphone app and resource locking and two-factor authentication. The presented evaluation study not only provides proof about the significance of our proposed approach but also gives evidence about the protection levels of specific lock patterns that are frequently used

Torres, J., Santos, S., Alepis, E. and Patsakis, C.
Behavioral Biometric Authentication in Android Unlock Patterns through Machine Learning.
DOI: 10.5220/0007394201460154
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 146-154
ISBN: 978-989-758-359-9

by users. Moreover, contrary to related work we detail how the proposed mechanism can be deployed in stock Android devices without the need to root the device.

Organisation of This Work. The rest of this work is structured as follows. In the next section, we present the related work regarding locking pattern in smartphones. Section 3 details the problem setting and the use case that this paper addresses, while Section 4 illustrates our proposed solution. Then, in Section 5 we detail the Machine Learning core of the developed system. Section 6 presents the evaluation experiment that was conducted, along with its results. Finally, Section 7 summarises our contributions and proposes future work.

2 RELATED WORK

In what follows we provide a brief overview regarding research on smartphone protection through its locking mechanisms and more specifically using the locking pattern. The authors have included works that reveal this topic's scientific significance, in terms of user statistics, studies on the theoretic security level of the mechanism in question, as well as attacks focusing on bypassing secure pattern lock screens.

As stated in (Malkin et al., 2016), to prevent unauthorized access to smartphones, their users can enable a "lock screen" which may require entering a PIN or password, drawing a pattern, or providing a biometric, such as users' fingerprints. In the survey conducted by (Malkin et al., 2016), involving more than 8.000 users from eight different countries, the prevailing method for locking a smartphone is considered the smartphone pattern, used in almost half of all users of the survey (48% of all locking mechanisms).

Nevertheless, in (Aviv et al., 2010), the authors examine the feasibility of "smudge" attacks on touch screens for smartphones and focus their analysis on the Android lockscreen pattern. Alarmingly, their study concludes that in the vast majority of settings, partial or complete patterns are easily retrieved. Indeed, the authors of (Aviv et al., 2010) managed to partially identify 92% of Android patterns and fully in 68% of their attempts, using camera setups.

As stated in (Aviv et al., 2017), Android unlock patterns are considered as the most prevalent graphical password system to date. The same researchers argue that human-chosen authentication stimuli, such as text passwords and PINs, are easy to guess and therefore investigate whether an increase in the unlock pattern grid size positively affects the security level of the mechanism in question.

In (Meng, 2016) two user studies were conducted with a total of 45 participants to investigate the impact of multi-touch behaviours on creating Android unlock patterns. While focusing mainly on the issue of usability, the author proposes increasing the number of touchable points and improving the rules of unlock pattern creation.

As shown in (Kessler, 2013), a mathematical formula for the exact number of patterns is not known yet, even for the simplest case of the unlock patterns, namely, the 3x3 grid. The authors of (Lee et al., 2017) and (Lee et al., 2016) respectively calculated in their works the lower and upper bounds of Android unlock patterns, thus providing a theoretical estimation of the unlock patterns' corresponding security level.

In (Canfora et al., 2016) readers will find a very interesting approach where the authors propose a continuous and silent monitoring process based on a set of user specific features, namely device orientation, touch and cell tower. Other kinds of protection techniques regarding user authentication include the works of (Nicholson et al., 2006) where the authors propose authentication token-based mechanisms to identify legal users, (Dunphy et al., 2010) where graphical password systems are utilized and (Luca et al., 2012) where the authors propose a novel application, where the user draws a stroke on the touch screen as a input password utilizing touch pressure, touch finger size and speed. While this work is the most similar to our work, our experiments have much higher percentages of accuracy, while our approach is based on finger movements on the screen. Arguably, only a very limited number of smartphone devices to date support finger pressure data calculations.

After a thorough investigation of the related scientific literature, we have come up with the conclusion that even though there is a growing interest towards the direction of securely locking, and consequently unlock attacks to, smartphones, we did not find significant scientific attempts, other that (Luca et al., 2012), in the direction of improving the already adopted unlock mechanism of the smartphone pattern utilizing machine learning and behavioral biometric user data, as described in this paper.

3 PROBLEM SETTING

As already discussed, smartphone locking is very common to the majority of smartphone users to date. In this section, we further analyze the problem of securing sensitive personal data that involve smartphones, going a step further than the actual access to

a physical device. We argue that modern smartphone users not only want to secure their mobile device but in many cases, there are also specific applications and data that need to be strongly secured. In order to understand this argument, a use case is described.

Let us assume that Alice has an Android device which has several applications, some of which require strong authentication, e.g. banking applications. Moreover, Alice may often share her device with daughter Carol to let her play or browse the Internet. Android may support more than one users. However, this feature is not often used due to usability issues and because in the case of Carol, shoulder surfing cannot be avoided.

Due to her job, Alice comes in contact with many people on a daily basis, and quite often she has to unlock her device in front of them. Therefore, Alice fears that her unlock PIN/pattern (and possibly other credentials) may have been disclosed through shoulder surfing. Moreover, Alice would like to have control of some web pages and apps to avoid possible issues, from posting something inappropriate on a social network to messing with her bank account.

Based on the above, Alice wants to be able to authenticate on a device easily, avoiding shoulder surfing attacks. Additionally, Alice wants to be able to share her device with Carol, yet lock some apps and possibly web pages so that her daughter cannot access them. In terms of implementation, we opt for a light solution so that the device must not be rooted but use existing and native mechanisms.

We argue that the above could be solved by providing some *context awareness* to an app that controls access to the device. In this case, context awareness refers to the ability of an app to infer:

- Which user has authenticated.
- Which are the running apps.
- Which web page the user wants to browse.

From the questions above, only the first one can be answered. Regarding running applications, Google has removed the `getRunningTasks` method of `ActivityManager` as of API level 21 to avoid apps surveying users and more importantly to prevent malicious apps from timely overlaying other apps (e.g. banking) and harvesting credentials. While in the literature there are several ways to determine the foreground app (Chen et al., 2014; Bianchi et al., 2015; Alepis and Patsakis, 2017), yet all of them have been deprecated in AOSP. Nevertheless, Google as of Android Lollipop, allows developers to use two methods to get usage statistics or detect the foreground app. More precisely, they may either use the `UsageStatsManager` API which requires the `PACKAGE_USAGE_STATS` system permission and allows an app to collect statistics about the usage of the installed apps, or use the `AccessibilityService` API which requires the `BIND_ACCESSIBILITY_SERVICE` system permission. Notably, both these permissions are "more than dangerous" permissions. In the first case though, Android does not allow apps to derive anything else apart from aggregated statistics about the usage of the installed apps. In this regard, an app that has been granted this permission may collect aggregated usage data for up to 7 days for daily intervals, up to 4 weeks for weekly intervals, up to 6 months for monthly intervals, and finally up to 2 years for yearly intervals, always depending on the chosen interval. In the second case, using the `AccessibilityService`, which includes handling the `onAccessibilityEvent()` callback and checking whether the `TYPE_WINDOW_STATE_CHANGED` event type is present, one may determine when the current window changes. It is important to note that Google has warned developers about this permission, that she will remove apps from the Play Store if they use accessibility services for "non-accessibility purposes" (Android Police, 2017).

Finally, regarding web pages, it is worthwhile to notice that since Nougat apps may not access any content of `/proc/` beyond `/proc/[pid]/` where `[pid]` is their own pid. Therefore, access to `/proc/net/tcp6` is not possible which would allow an app to infer the domain that another app tries to access. Hence, the only available ways to intercept the network usage seem to be by re-routing the network traffic through a local proxy or a VPN. Either of these cases introduces its own security and trust constraints. For instance, the use of a local proxy implies that self-signed certificate must be installed with the latter triggering a security notification in Android and implying further trust issues. In both cases (local proxy/VPN) one has access to all the user's unencrypted traffic.

4 PROPOSED SOLUTION

In this section, we describe our proposed solution regarding the resulting application. It should be noted that the backend of our app implements a specific use case of our online service, also called "SmartPattern", which works as an authorization/authentication mechanism of a service that allows protecting any external resource through "Smart Patterns" (using an API, Oauth2 or JWT). At present, the app focuses on securely locking specific apps inside the Android ecosystem. The OS itself could adopt the described

underlying approach and use it for locking the Android devices more "securely". Indeed, we have developed an Android app that once installed and initialized, can be used as a "locking" mechanism, allowing access to smartphone owners in specific resources inside their smartphone and correspondingly denying access to fake ones.

All resources inside the Android OS are handled through corresponding OS apps or third-party apps. In this sense, private resources can also be present within specific apps, such as file-managers, chat applications and photo gallery applications. Our approach utilizes the provided by the OS `UsageStatsManager` API and handles it to recognize the foreground app successfully. Then, after the app's initialization, its users may choose from a list of installed and pre-installed apps, in their device, that they need to be securely locked.

More precisely, through the `UsageStatsManager` API, we have managed to successfully recognize the foreground app, contrary to Android documentation that states that this API is used only for "*Usage data that is aggregated into time intervals: days, weeks, months, and years*", (Developers Android, 2018). In our approach, we check for newly created lists of `UsageStats` in regular time intervals, taking only the latest `INTERVAL_DAILY` records into consideration. Then, looping through the retrieved list, the most recent record in terms of the timestamp is kept which corresponds to the foreground app's package name. Using this approach, we manage to "segregate" the initially "aggregated" results and detect the user's foreground application.

Then, our app starts working in the background, as a service, silently monitoring each launched app. Whenever the service recognizes a launched app that is selected by the user to be securely locked, our app presents a full-screen Android activity, "hiding" it, and thus protecting the app in question. This "protecting" screen can only be bypassed if the user authenticates himself, through our novel smart pattern mechanism. As it will be further analyzed in the following section, the pattern mechanism not only checks whether the correct pattern is being drawn, but also whether the user who is drawing the pattern can be authenticated from his behaviour while drawing the pattern, through the machine learning core underlying module.

The secure screen provided by our app cannot be bypassed, since even when it is minimized or closed, the running service will continuously keep on relaunching it, having detected a "protected" app in the foreground. A possible attack on our app could be an attempt to uninstall it. This can also be easily protected, by additionally monitoring the "settings" of the device through our application, denying access to the app uninstallation panel.

Our proposed secure locking/unlocking mechanism has been designed to be as much "lightweight" and "robust" in using it as possible. Both for supporting user experience (UX) and also for providing the end user with a mechanism that is very close to the one that he is used to. Indeed, after some running several experiments with real hardware devices (not emulators) our results have shown that the entire authentication process, after the user has drawn a pattern in the app, is completed on an average of 100-180 milliseconds, clearly not "slowing down" the authentication process.

In each case, the work presented in this paper does not focus only to the app being the final "product", but to the underlying approach, which has been made to further secure smartphones and sensitive resources from malicious users. As a result, the proposed solution could be used in an increasing number of even more use cases, such as two-factor authentication with increased levels of security. Thus, its evaluation results, presented in the following sections are also considered as of great importance in terms of its scientific contribution.

5 MACHINE LEARNING CORE

To work with drawn patterns, we need to initially establish an encoding of the associated data. Besides, this encoding must include information about how the user enters the pattern; not only the drawing path, that would enable the Machine Learning (ML) system to learn about a set of pattern inputs and allow for successful predictions.

5.1 Pattern Design and Encoding

To encode the information provided by the unlock pattern, we have established an enriched pattern path (pattern + times) as an array of vectors (one by each existing point in the pattern). According to that, the representation of the resulting, "enriched" pattern will meet the following structure:

$$[(X_1, Y_1, t_1), (X_2, Y_2, t_2), ..., (X_n, Y_n, t_n)]$$

where X_i represents the row number of the point regarding the virtual matrix where the pattern is entered, Y_i represents the column number and t_i the time elapsed since the last point was reached (it will be 0 for the first point). Clearly, $(X_i, Y_i) \neq (X_{i+1}, Y_{i+1}) \forall i \in$

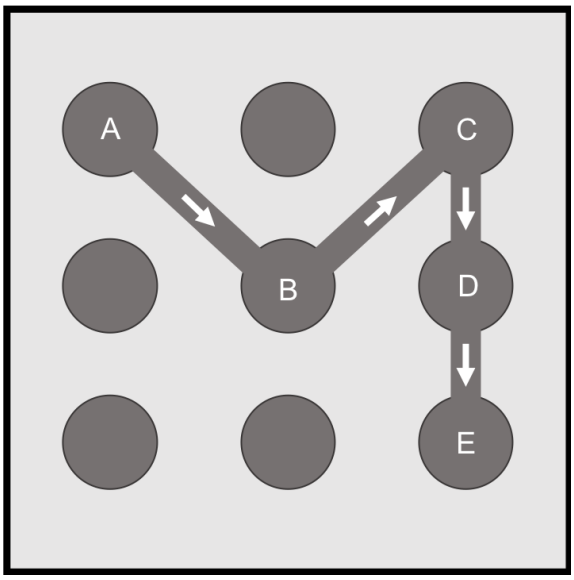


Figure 1: Example of a pattern input.

$(1, n-1)$. According to this encoding, the pattern presented by Figure 1 will be encoded as follows:

$$[(1,1,0),(2,2,t_{AB}),(1,3,t_{BC}),(2,3,t_{CD}),(3,3,t_{DE})]$$

5.2 ML Algorithms Selection and Specifications

Once the enriched pattern representation is defined, it should be possible to generate a model for each user to predict whether each one of the future patterns introduced in the application actually belongs to the "genuine" user. Therefore, and after checking the shape of the pattern, the next step focuses mainly on exact "timing" extraction of the drawn pattern, generating a feature vector composed by each point's timestamp representation in the drawn pattern. Continuing with the same above example, we could have different valid inputs for the pattern shown by Figure 1 such as:

$$[(1,1,0),(2,2,0.22),(1,3,0.15),(2,3,0.43),(3,3,0.50)]$$
$$[(1,1,0),(2,2,0.17),(1,3,0.15),(2,3,0.41),(3,3,0.57)]$$
$$[(1,1,0),(2,2,0.20),(1,3,0.12),(2,3,0.40),(3,3,0.54)]$$
$$[(1,1,0),(2,2,0.21),(1,3,0.16),(2,3,0.40),(3,3,0.53)]$$
$$[(1,1,0),(2,2,0.23),(1,3,0.14),(2,3,0.42),(3,3,0.53)]$$

After extracting only the time values, the corresponding result will be the following:

$$[0,0.22,0.15,0.43,0.50]$$
$$[0,0.17,0.15,0.41,0.57]$$
$$[0,0.20,0.12,0.40,0.54]$$
$$[0,0.21,0.16,0.40,0.53]$$
$$[0,0.23,0.14,0.42,0.53]$$

Using this known and valid pattern features vectors, it is then possible to generate a dataset to build a ML model, able to predict whether an entered pattern belongs to the real user or not. In this case, it seems not straightforward to implement a supervised strategy, mainly because it is not possible to learn incorrect pattern inputs. However, it is possible to use some One Class Supervised algorithms (such as One Class Support Vector Machine, SVM) or another unsupervised ML approaches like clustering. In our case and for the purposes of this study, we have used both, namely a One Class SVM and also the Clustering approach through an implementation of the K-means algorithm.

5.2.1 One Class SVM Implementation

Using the One Class SVM algorithm, we have found it possible to learn how to distinguish the valid enriched patterns from other, invalid ones, using a training dataset only composed by valid samples (only one class). As output, the algorithm will return a binary result (usually 1 and -1), although the result may vary depending on the implementation, language, etc., depending on the sample that has been classified as valid (similar to valid samples used to train the model) or invalid.

5.2.2 K-means Implementation

In this case, after a dimensionality reduction using PCA, we have used different configurations for the K-means algorithm. Nevertheless, after analyzing the results, we have come up with the findings that the best results have been obtained using 3 and 5 clusters in number respectively. The method used for identifying the most representative pattern for the user consists of clustering the training set samples and then selecting the main cluster, that is the one that contains more samples. During the algorithm's final step we have been able to predict whether an enriched pattern belongs to the user by checking whether it has been positioned into the main cluster of the trained model.

Figure 2 illustrates both the training and the testing phase, where the main cluster is "C1".

6 EVALUATION EXPERIMENT

In this section, we describe the settings of the experiment that has been conducted to evaluate the effectiveness of the resulting app. The experiment involved

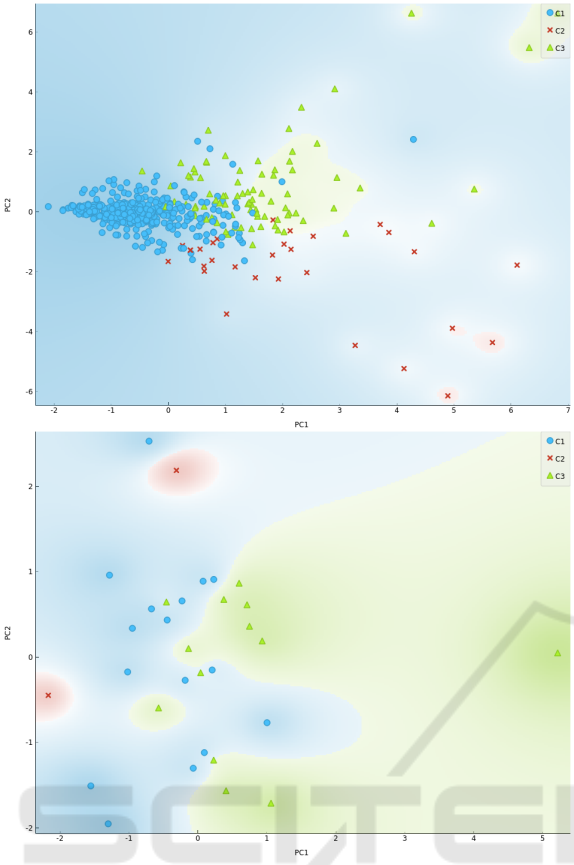


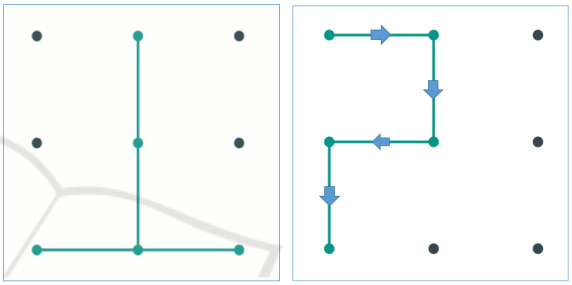
Figure 2: Clustering samples at test (top) and training (bottom) phase.

64 users, as well as 6 supervisors in a total of 70 participants, including 2 phases of evaluation. The first phase involved 54 users and the 6 supervisors, while the second phase involved 10 users and the same supervisors of phase 1.

Specifically, 54 of the evaluation users tested the app's effectiveness of the smart-pattern approach, not being able to be "broken" by other users. Respectively, 10 of the evaluation users tested the app's user-friendliness and focused in its efficiency by considering interaction complexity and minimum false positives.

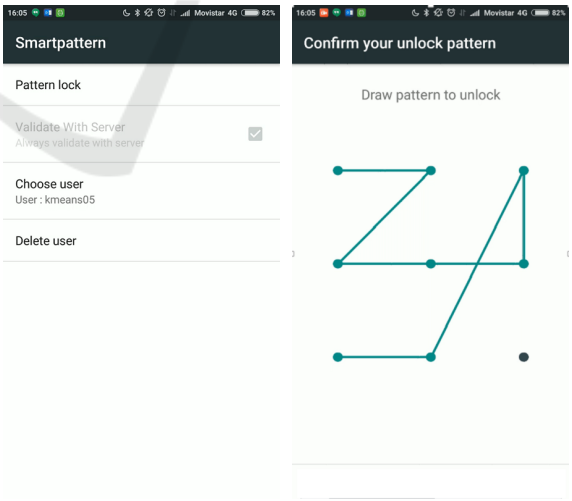
At this point we should make a short discussion about the different types of user patterns both in terms of complexity that translates into the number of points the users' use and also in terms of complexity that translates in differentiations while drawing the actual pattern, namely quick or slow finger movements and also intended "strategic" pauses of the user in specific parts of the pattern. Our experiment revealed that these criteria are quite significant since both the number of points of the patterns and also the timings involved affect both the effectiveness of the underlying algorithm to reject the false users, but also the system's effectiveness in minimizing false positives of real users. Indeed, as expected, a smart pattern becomes more effective in terms of security as the number of points increases, while users' atomic timings while drawing their patterns is of equal or even greater importance in terms of app safeguarding.

The following figures illustrate some drawn patterns that have been used for the evaluation of the smart pattern app. More specifically, Figures 3a and 3b represent the initial, simple patterns of the experiment. Of course, the actual patterns that take the timings of the drawn patterns into consideration cannot be visualized in any figure, since for this purpose, video files should be utilized. Figures 4a and 4b illustrate screenshots while using the resulting app.



(a) Pattern without direction data. (b) Pattern with direction data.

Figure 3: Simple patterns used in the first two steps of the evaluation study.



(a) App settings (b) Drawing Pattern

Figure 4: Screenshots from the app.

6.1 First Phase of the Experiment (54 Users)

Settings: The experiment included a tablet and a smartphone for each evaluator user. The tablet was displaying the instructions and corresponding videos for the users. The smartphone had the app installed to be tested. A supervisor was also monitoring the whole process, both for helping and most importantly noting the evaluation results of the experiment. The supervisor was noting for each phase and user attempt, whether the users' attempts were successful (allowing access to the user) or not (denying access to the user). All the supervisors' results were afterwards transferred to a database for further processing and visualization.

Steps: This experiment phase involved 5 steps, each one corresponding to the 4 levels of smart-pattern complexity within the evaluation experiment, while there was a fifth step where the users had more "help" to pass the smart pattern test successfully. In each one of the 5 steps, each user made 10 subsequent attempts to pass the corresponding smart pattern challenge. More specifically, step 1 involved an experiment with the user trying to pass the pattern screen successfully by only seeing an image of the correct pattern, having no motion directions. Step 2 involved the user trying to pass the smart pattern screen successfully by only seeing an image of the correct pattern, accompanied with small arrows in the image, indicating the correct direction. However, the users had no indication of the "way" the real users' fingers moved, trying to pass the pattern screen. Step 3 involved the user trying to pass the pattern screen successfully by watching a video of the pattern being drawn by the actual real users of the smartphone, using a common, yet of medium complexity, pattern. Step 4 involved the user trying to successfully pass the pattern screen by watching a video of the pattern being drawn by the actual real users of the smartphone, using a difficult (of higher complexity) pattern. Finally, step 5 was almost identical to the fourth step of the evaluation, while there was additional help to the users by giving them the opportunity to watch the video of unlocking the screen one more time to memorize the correct pattern even better.

6.2 Second Phase of the Experiment (10 Users)

Settings: This phase involved 10 users using their own mobile, android powered smartphone, where they downloaded, installed and consequently used the app. The app was downloaded from a web link. A supervisor was also monitoring the whole process, both for helping purposes and also to note down the results of the evaluation experiment. All the supervisors' results were transferred to a database for further processing and result visualization.

The users followed instructions about how to create their personal pattern and subsequently "train" the model. After successfully creating their patterns the users made 10 subsequent attempts to unlock the app. The supervisor noted the number of successful unlocks by each user.

6.3 Results of the Evaluation Experiment

The collected results were merged and analyzed to produce the core of our evaluation experiment. As described above, each user in the first phase made in total 50 attempts to successfully pass each one of the five smart pattern challenges. As for the evaluation results, the 54 evaluation users of the first phase made 2700 attempts in total. After the analysis of the results, from the total of 2700 user attempts, 2530 of them were unsuccessful, meaning that our proposed system had 93.7% success in denying access to unauthorized users. Respectively, in the second phase of the experiment, 10 users made 10 attempts to unlock the app using their own personal trained model of the smart pattern app. In total, they made 100 attempts, where 86 of them have been successful. As a result, our proposed approach reached a percentage of 86% in successfully "recognizing" the app's genuine user and consequently accepting his/her access to the app.

For each basic pattern attempt for the attacker, the SmartPattern app blocked the attempts illustrated in percentages in Figure 5. As a next step of analyzing the results of the evaluation study, each one of the five steps is also discussed. We may easily note that the percentage of the SmartPattern app's security increases, the more "complex" the drawn pattern is. This result might seem expected, however, in our study the complexity level could be translated not only in terms of "more points" but also in terms of "timing pauses", meaning that if the real user had a more "unique" way of drawing the pattern, then this would increase its security level. Another very interesting observation deriving from our study is that malicious users having knowledge only for the "final drawing" and not of the way, in terms of consequent points, that it was drawn did not have much success in "guessing" the correct way of unlocking the pattern. Finally, another significant result of the study is that in the cases of the more complex drawn patterns and consequently their more complex involved tim-

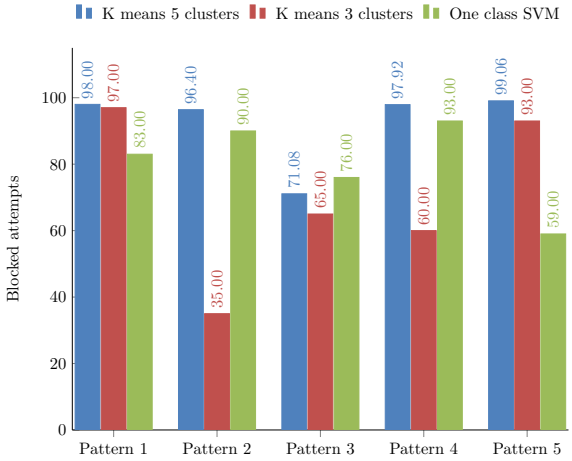


Figure 5: Attempts blocked by SmartPattern when other users try to unlock the mobile phone.

ing biometrics, even when the "attackers" had more "help" by watching the pattern being drawn more times in videos, their unlock attempts where still unsuccessful in their vast majority. These results have been quite encouraging, indicating that our research pointed in a good direction, actually providing improvements in the smartphones' unlocking mechanisms.

7 CONCLUSIONS

In this paper, a novel locking mechanism for Android smartphones that utilizes Machine Learning and user biometric data has been presented. Our approach can be incorporated in many application domains, ranging from providing more security to mobile devices when locking them, to specifically securing targeted "sensitive" resources and also improving 2FA without the need to root the device. The evaluation results have shown a 93.7% success rate in denying access to unauthorized users and a 86% success rate in allowing access to the real users of the app. This little "lower" success percentage of the resulting app can be justified since it was the authors' primary objective, when calibrating the ML backend, to maximize user protection, even if this resulted in lower "user friendliness" level.

Our evaluation experiments have provided us with strong evidence that our approach is very successful in "distinguishing" genuine and malicious users through the way they draw lock screen patterns. The paper's results also provide the scientific literature with valuable evidence about the efficiency of common patterns in securely protecting the smartphones.

In the future we plan to develop apps corresponding to other use cases, such as the implementation of an app for 2FA purposes using our ML biometric backend, to further improve the protection level of the smartphone users.

ACKNOWLEDGMENTS

This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the *OPERANDO* project (Grant Agreement no. 653704) and the University of Piraeus Research Center.

REFERENCES

Alepis, E. and Patsakis, C. (2017). Trapped by the ui: The android case. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 334–354. Springer.

Android Police (2017). https://www.androidpolice.com/2017/11/12/google-will-remove-play-store-apps-use-accessibility-services-anything-except-helping-disabled-users/.

Aviv, A. J., Gibson, K. L., Mossop, E., Blaze, M., and Smith, J. M. (2010). Smudge attacks on smartphone touch screens. In Miller, C. and Shacham, H., editors, *4th USENIX Workshop on Offensive Technologies, WOOT '10, Washington, D.C., USA, August 9, 2010*. USENIX Association.

Aviv, A. J., Kuber, R., and Budzitowski, D. (2017). Is bigger better when it comes to android graphical pattern unlock? *IEEE Internet Computing*, 21(6):46–51.

Bianchi, A., Corbetta, J., Invernizzi, L., Fratantonio, Y., Kruegel, C., and Vigna, G. (2015). What the app is that? deception and countermeasures in the android user interface. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pages 931–948. IEEE Computer Society.

Canfora, G., Notte, P. D., Mercaldo, F., and Visaggio, C. A. (2016). Silent and continuous authentication in mobile environment. In Callegari, C., van Sinderen, M., Sarigiannidis, P. G., Samarati, P., Cabello, E., Lorenz, P., and Obaidat, M. S., editors, *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016.*, pages 97–108. SciTePress.

Chen, Q. A., Qian, Z., and Mao, Z. M. (2014). Peeking into your app without actually seeing it: Ui state inference and novel android attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 1037–1052, San Diego, CA. USENIX Association.

Developers Android (2018). https://developer.android.com/reference/android/app/usage/UsageStatsManager.

Dunphy, P., Heiner, A. P., and Asokan, N. (2010). A closer look at recognition-based graphical passwords on mobile devices. In Cranor, L. F., editor, *Proceedings of the Sixth Symposium on Usable Privacy and Security, SOUPS 2010, Redmond, Washington, USA, July 14-16, 2010*, volume 485 of *ACM International Conference Proceeding Series*. ACM.

Kessler, G. C. (2013). Technology corner: Calculating the number of android lock patterns: An unfinished study in number theory. *JDFSL*, 8(4):57–64.

Lee, J., Seo, J. W., Cho, K., Lee, P. J., Kim, J., Choi, S. H., and Yum, D. H. (2016). A visibility-based upper bound for android unlock patterns. *IEICE Transactions*, 99-D(11):2814–2816.

Lee, J., Seo, J. W., Cho, K., Lee, P. J., and Yum, D. H. (2017). A visibility-based lower bound for android unlock patterns. *IEICE Transactions*, 100-D(3):578–581.

Luca, A. D., Hang, A., Brudy, F., Lindner, C., and Hussmann, H. (2012). Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In Konstan, J. A., Chi, E. H., and Höök, K., editors, *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 987–996. ACM.

Malkin, N., Harbach, M., Luca, A. D., and Egelman, S. (2016). The anatomy of smartphone unlocking: Why and how android users around the world lock their phones. *GetMobile*, 20(3):42–46.

Meng, W. (2016). Evaluating the effect of multi-touch behaviours on android unlock patterns. *Inf. & Comput. Security*, 24(3):277–287.

Nicholson, A. J., Corner, M. D., and Noble, B. D. (2006). Mobile device security using transient authentication. *IEEE Trans. Mob. Comput.*, 5(11):1489–1502.

Wikipedia (2018). https://en.wikipedia.org/wiki/FBI\%E2\%80\%93Apple_encryption_dispute/.