Towards Application-Centric Implicit Authentication on Smartphones

Hassan Khan, Urs Hengartner
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON Canada
{h37khan, urs.hengartner}@uwaterloo.ca

ABSTRACT

Implicit authentication schemes are a secondary authentication mechanism that provides authentication by employing unique patterns of device use that are gathered from smartphone users without requiring deliberate actions. Contemporary implicit authentication schemes operate at the device level such that they neither discriminate between data from different applications nor make any assumption about the nature of the application that the user is currently using. In this paper, we challenge the device-centric approach to implicit authentication on smartphones. We argue that the conventional approach of misuse detection at the device level has inherent limitations for mobile platforms. To this end, we analyze and empirically evaluate the device-centric nature of implicit authentication schemes to show their limitations in terms of detection accuracy, authentication overhead, and fine grained authentication control. To mitigate these limitations and for effective and pragmatic implicit authentication on the mobile platform, we propose a novel application-centric implicit authentication approach. We observe that for implicit authentication, an application knows best on *when to authenticate* and *how to authenticate*. Therefore, we delegate the implicit authentication task to the application and let the application provider decide *when* and *how* to authenticate a user in order to protect the owner's personal information. Our proposed application-centric implicit authentication approach improves accuracy and provides fine grained authentication control with low authentication overhead. Future research in this domain will benefit from our findings to provide pragmatic implicit authentication solutions.

1. INTRODUCTION

Smartphones contain their owner's personal data including emails, texts, contacts, photos and stored passwords. In addition, people use their smartphones for security-sensitive tasks such as online banking. To protect smartphones from misuse, PIN/pass-lock, draw-a-secret, face recognition, and fingerprint recognition based schemes are used as primary authentication mechanisms. However, these authentication mechanisms have usability issues due their all-or-nothing access nature and 62% of smartphone owners do not configure them on their devices [2, 7]. In addition to usability issues, pass-lock and draw-a-secret schemes are subject to operating system flaws and shoulder surfing attacks [5] and researchers have demonstrated how facial and fingerprint recognition schemes can be exploited [21, 22]. To mitigate these limitations, researchers have proposed implicit authentication schemes to complement or enhance existing explicit authentication mechanisms [3, 4, 6, 7, 8, 16, 17].

Implicit authentication schemes provide authentication by using distinctive, measurable patterns of device use that are gathered from mobile device users without requiring deliberate actions [16]. For example, a keystroke pattern based classifier learns to recognize the device owner's keystroke patterns and then monitors for anomalies in real-time keystroke patterns to flag misuse. Contemporary implicit authentication schemes are device-centric in such a way that the device's OS monitors usage behavior to compute an authentication score without discriminating between data from different applications and without making any assumptions about the nature of the application that the user is currently using [5, 7, 16, 17]. While these implicit authentication schemes work in principal, we challenge their device-centric approach on smartphones. We argue that the traditional approach towards misuse detection at the device-level has inherent limitations for the mobile platform. To this end, we analyze and empirically evaluate the device-centric nature of implicit authentication schemes and show their limitations in terms of: (i) detection accuracy; (ii) authentication overhead on the device; and (iii) fine grained authentication control. During our analysis we also trace these limitations to their cause — the application-oblivious nature of device-centric implicit authentication.

To mitigate these limitations and for effective and pragmatic implicit authentication we propose a novel approach: application-centric implicit authentication. We observe that for implicit authentication, an application knows best on: (i) when to authenticate a user; (ii) which behavior-based classifier to authenticate with; and (iii) which features of a classifier to use for authentication. In the application-centric approach, the decisions of *when* to authenticate and *how* to authenticate are delegated to the application and the application provider defines the logic to make these decisions. This enables the application provider to provide fine grained authentication control (by authenticating only *when* required) with high accuracy (by choosing a suitable classifier with appropriate configurations). For example, an application developer may choose a classifier based on touchscreen input behavior for his browser application (instead of a classifier based on keystroke patterns) and he may invoke the classifier every time a user accesses a webpage that has the owner's credentials saved using the browser's password manager (instead of authenticating a user even when he is reading news).

In this paper, we justify the need of a different design for implicit authentication that is more appropriate for the mobile platform. To this end, we first evaluate a classifier based on touchscreen input behavior using the device-centric approach to understand the limitations imposed by the conventional design. We then discuss the goals and challenges of designing an application-centric counterpart. Preliminary evaluations on four sample applications show that the application-centric approach can exploit the knowledge of an application's nature to provide consistent and significant accuracy improvements with less authentication overhead and fine grained authentication control.

2. WHERE TO AUTHENTICATE?

The focus of the majority of the implicit authentication literature is on identifying suitable behavior-based classifiers for implicit authentication [4, 5, 6, 8] and only a few papers discuss when to implicitly authenticate [2, 7, 19]. To the best of our knowledge, the impact of *where* to authenticate has not been investigated. In this section, we provide a comparison between two options on *where* to authenticate — at the device level or at the application level. For the comparison, we analyze how the outcome of *where* to authenticate impacts the decisions of *when* and *how* to authenticate.

2.1 When to Authenticate?

A device-centric implicit authentication scheme may operate continuously in the background or get triggered when an application marked as sensitive is launched. However, some applications may not require implicit authentication for all usage scenarios. For example, consider a banking application that enables a customer to query his account or locate a nearby ATM. For this application, there is no need to authenticate a user when he is trying to locate an ATM. Similarly for a browser application, there is no need to authenticate a user when he is reading news. Since a device-centric approach is unaware of the task that the user is performing within an application, it cannot provide authentication control at the task level.

On the other hand, if an application can control when to authenticate, the banking application only authenticates a user when he is querying his account. Similarly, the browser application can detect when a user is reading news and it may decide not to authenticate him. But when the user switches to his social network website with saved credentials, implicit authentication may be performed. By delegating the decision of *when* to authenticate to the application, we can perform task-aware implicit authentication, which reduces unnecessary authentication overhead. Another advantage of task-aware authentication control is its inherent support for multi-user scenarios. Smartphone owners share a mean of 12% of their smartphone applications and these applications are primarily entertainment applications such as games or web browsers [2]. For these multi-user scenarios, the non-owner is not supposed to access personal information of the owner on the device. An application-centric approach allows a non-owner to access content that does not leak personal information and denies access to the content that may leak personal information (e.g., access to a mail portal with saved credentials).

2.2 How to Authenticate?

A device-centric approach may employ a specific behavior-based classifier for implicit authentication. However, a classifier may not be suitable for a particular type of application. For example, during a session in which a user accesses his social network website on the browser using saved credentials, enough sample values may not be available for a classifier based on keystroke patterns to compute an authentication score. Similarly, a classifier based on touchscreen input behavior may not have enough sample values for a messenging application (due to lack of swipes generated by the user). To cater for these behavioral differences, a device-centric approach may employ multiple classifiers; however, this will result in significant overhead in terms of feature sampling. On the other hand, an application-centric approach can leverage the knowledge of an application's nature to choose the appropriate behavior-based classifier.

Another limitation of the device-centric approach due to its application-oblivious nature is the loss of valuable information that may be useful for classification. If the classifier has additional knowledge about a task that the user is currently performing, it may use that for robust classification. For example, in Section 4 we show that a user's touch behavior is slightly different when he is finding POI in maps as compared to browsing. Consequently, in the banking application, if the classifier is aware of the task being performed by the user (finding nearest ATM or querying his account), it may use this additional information to improve its accuracy by tuning its features (we demonstrate this in Section 5).

2.3 Discussion

The comparison of application- and device-centric approaches shows that due to the application-aware nature of the former, fine grained authentication control is acquired and implicit authentication is performed only when required. This fine grained authentication control also reduces authentication overhead. The application-centric approach can use its knowledge about an application to determine the appropriate classifier for that application. Finally, the application-centric approach can leverage its knowledge about an application's nature to improve the accuracy of the classifier by tuning its features.

3. CLASSIFIER SELECTION AND DATA COLLECTION

In the last section, we discussed different authentication scenarios from device- and application-centric perspectives. Before proceeding to the empirical evaluations of these approaches, we discuss behavior-based classifier selection and data collection.

3.1 Classifier Selection

Various behavior-based approaches have been proposed for misuse detection on smartphones, which use location patterns [9], call and text patterns [8], keystroke patterns [14], proximity to known devices [7], gait patterns [12], voice pattern [18], micro-movement patterns due to a user's actions [6], and touchscreen input behavior [4, 5]. We select a classifier based on touchscreen input behavior [4] for empirical evaluation, which relies on finger movement patterns of a user for implicit authentication.

The classifier based on touchscreen input behavior uses data that are generated as a result of a user's normal interaction with the phone unlike approaches based on location and proximity patterns, which use data from power-hungry GPS and bluetooth sensors, respectively. Furthermore, for the classifier based on touchscreen input behavior, it is more likely that recent sample values are available as a result of user actions unlike gait, call/text, and voice pattern based approaches. Finally, the classifier based on touchscreen input behavior has more data samples available for the majority of applications as compared to an approach based on keystroke patterns [5].

3.2 Data Collection

Our goal for data collection was to collect a dataset that captured natural behavior of the participants when they used applications. We did not want the participants to perform predefined tasks. We also wanted to study their touchscreen input behavior across a diverse set of applications. To satisfy these data collection goals, we instrumented four diverse Android applications for data collection including a browser application, a maps/navigation application, a launcher application and a comic viewer application. We uploaded these applications to Google Play. To advertise for participants, we used our university-wide mailing list for participants who would be interested in a study on "User Interaction with Smartphone Applications". Participants were expected to install and use these applications for ten weeks and they were paid 5$ per week. We did not ask the participants to explicitly perform any tasks and participants were expected to use these applications as per their needs. Finally, in order to avoid any bias, we did not mention the real purpose (implicit authentication) of the study until the post-debriefing session. This allowed us to capture participants' *in the wild* touchscreen input behavior.

For data collection, every time a participant interacted with the touchscreen on one of the applications, we recorded: time stamp in milli-seconds, x and y co-ordinates of the touch point, finger pressure on the screen, area covered by the finger on the screen, values from the accelerometer sensor, finger orientation, the screen's orientation, and the phone's orientation (azimuth, roll and pitch). These values were temporarily stored on the participant's device and then transmitted to a server. Before data transmission, we established ground truth (only the participant used the applications) by asking participants to label the intervals for which they were *absolutely certain* that the device was in their possession.

Our applications were downloaded and used by 61 participants. Out of 61 participants, 14 participants did not provide enough data across all four applications and 15 participants stopped submitting data before completing five weeks of the study. For empirical evaluations, we only consider data from the 32 participants who successfully completed the study. In total, we logged about 2.48 million touch points comprising over 53,000 swipes in ten weeks. The details of swipes, their distribution across applications and their distribution across user sessions is provided in Table 1.

Application	**Num. of touchpoints**	**Num. of Swipes**	**Sessions**
Launcher	642442	19740	4417
Browser	1164011	20139	826
Maps	236878	4664	365
Comics	445538	8928	272
Total	**2488869**	**53471**	**5880**

Table 1: Statistics of Collected Data

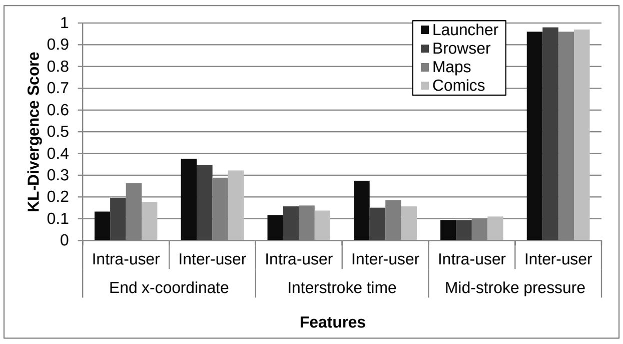

Figure 1: KL-Divergence score of features across 4 applications used in this study

4. EVALUATION OF DEVICE-CENTRIC APPROACH

We now evaluate Frank et al.'s touch behavior-based classifier [4] on our dataset. We apply the classifier in a device-centric manner such that the classifier neither discriminates between data from different applications nor it makes any assumptions about the nature of the application. We employed the 31 features proposed by Frank et al. [4] (also listed in Table 2) using an SVM-based classifier and we set the classifier's parameters to the values recommended in the original paper.

We evaluate the device-centric classifier using five-fold cross-validation and average the results across all the users. For evaluation metrics, similar to [4, 7], we use the false-acceptance rate (FAR), and the false rejection rate (FRR). FAR is the fraction of times the classifier incorrectly classifies a swipe of the non-owner as a swipe of the owner and FRR is the fraction of times a classifier incorrectly classifies a swipe of the owner as a swipe of the non-owner. The accuracy evaluation of Frank et al.'s approach on our dataset provides a FAR of 0.17 (with FRR of 0.08). The results on our dataset show significantly higher FAR as compared to the FAR of 0.07 (with FRR of 0.07) reported by Frank et al. for 35 users on their dataset [4].

We hypothesize that the variance in application nature and usage behavior is responsible for this accuracy degradation. Since Frank et al. only used two applications (an application to read Wikipedia articles and a spot-the-difference game) in a lab setup and participants performed predefined tasks, their experimental setup did not capture this behavior. In order to evaluate our hypothesis, we use the Kullback-Leibler (KL) information divergence measure [23]. A brief description of the KL-divergence measure is provided below. For two Probability Mass Functions (PMFs) p and q of a discrete random variable X, KL-divergence quantifies

1. Start x-coordinate 2. Start y-coordinate 3. End x-coordinate 4. End y-coordinate 5. Avg. direction 6. Avg. velocity 7. Stroke duration 8. Mid-stroke area 9. Mid-stroke pressure 10. Direction of end-to-end line 11. Direct end-to-end distance 12. Length of trajectory 13. 20%-perc. pairwise velocity 14. 50%-perc. pairwise velocity 15. 80%-perc. pairwise velocity 16. Median velocity at last 3 points 17. 20%-perc. pairwise acceleration 18. 50%-perc. pairwise acceleration 19. 80%-perc. pairwise acceleration 20. Ratio of end-to-end distance and length of trajectory 21. Largest deviation from end-to-end line 22. 20%-perc. deviation from end-to-end line 23. 50%-perc. deviation from end-to-end line 24. 80%-perc. deviation from end-to-end line 25. Mean resultant length 26. Median acceleration at first 5 points 27. Inter-stroke time 28. Phone orientation 29. Mid-stroke finger orientation 30. Up/down/left/right flag 31. Change of finger orientation

Table 2: Touchscreen Input Behavioral Features Proposed by Frank et al. [4]

the difference between two PMFs as:

$$D(p\|q) = \sum_{i \in \Lambda} \ln\left(\frac{p(i)}{q(i)}\right) p(i), \qquad (1)$$

where Λ is the image of X and $p(i)$ and $q(i)$ respectively represent the probability of feature value i in p and q. To deal with the case when $D(p\|q) = \infty$, if $p(i) \neq 0, q(i) = 0$; we perform standard Laplace correction. We calculate the inter-user and the intra-user KL-divergence score across all the 31 features in Table 2 for every application. The inter-user KL-divergence score is calculated between two partitions of a user's data and the intra-user KL-divergence score is calculated between a user's data and data samples from the rest of the users. If a feature is a good discriminating feature, it should have low intra-user KL-divergence score and high inter-user KL-divergence score.

Due to space constraints, we only show KL-divergence scores for three representative features in Figure 1 and omit other qualitatively similar results. Our results show that the majority of the features have low intra- and high inter-user KL-divergence score for a subset of applications only. For example, in Figure 1, end x-coordinate is a good discriminating feature for the launcher application, but for the maps application it has a relatively high intra-user KL-divergence score. Looking at the raw data, we can see that this is due to the nature of the maps application. In the maps application, users' start and end points are more random since they are locating some POI on the map unlike the launcher application, where a user generally swipes at a specific location on the screen. Similarly, the browser application fails to provide a high inter-user KL-divergence score for inter-stroke time feature. We suspect that this is due to the dependence of inter-stroke time on the content that a user is browsing (reading an article might result in lower inter-stroke times as compared to skimming it). While the majority of the features are not good discriminating features for all the applications, mid-stroke pressure and mid-stroke area covered were the only features that had consistently low intra-user and high inter-user KL-divergence scores. Since only two features cannot provide a good separation boundary between different users' behavior (due to more chances of collisions), for efficient classification all the good discriminating features for an application should be used.

Our analysis of the device-centric approach shows its failure to capture the variance in applications' nature. A simple solution to this limitation would be to create a separate behavioral model of each application at the device level. This would allow the device-centric approach to compare the runtime behavior of an application with its behavioral model to get a more accurate authentication score. While this approach would fix the accuracy degradation to some extent, it neither provides fine grained authentication control nor mitigates the limitations due to the device-centric approach's obliviousness of an application's nature.

5. APPLICATION-CENTRIC IMPLICIT AUTHENTICATION

For effective and pragmatic implicit authentication, we propose application-centric implicit authentication in which an application decides when and how to authenticate a user. While designing an application, an application developer first identifies activities that may lead to potential misuse. For example, for the browser application, the application developer may consider access to all websites that ask for a user's credentials as potential sources of data leakage. After identifying the activities, the application developer only implicitly authenticates a user when these activities are performed. The application developer also decides *how* to authenticate by choosing a suitable classifier for his application (as per the application's nature). For example, looking at the user's interaction with his browser application, an application developer may choose a classifier based on touch input behavior (since more swipe data is readily available than keystrokes data). Finally, the application developer tunes the features of the classifier as per his application's nature. We note that these activities increase the application developer's development overhead; however, this overhead can be mitigated by providing a library to the application developer that provides a generic implementation of behavior-based classifiers that can be extended and reused.

We now evaluate the sample applications on our dataset in an application-centric approach. To simplify our evaluations, we assume that these applications require implicit authentication for all activities and a classifier based on touchscreen input behavior is the right choice for them. We now show how we tuned the features for these applications and we then present accuracy results.

5.1 Feature Tuning

For feature tuning, an application developer collects sample data from some test users. He then determines the features that are good discriminating features for his application and trims the rest of the feature from the classifier. The application developer can optionally add new features for robust classification. We now summarize our experience of these feature tuning aspects for our sample applications.

Feature deletion: For each application, we deleted those features that had high intra-user KL-divergence score and low inter-user KL-divergence score. This trimming left 27, 25, 22, and 23 features for the launcher, browser, maps and comics applications, respectively. Feature trimming ensures

	Dev-centric Approach		App-centric Approach		Feature Tuned App-centric	
Application	**FAR**	**FRR**	**FAR**	**FRR**	**FAR**	**FRR**
Launcher	15.96%	6.27%	11.16%	4.13%	7.28%	3.29%
Browser	18.88%	7.95%	12.43%	5.68%	6.31%	3.59%
Maps	16.21%	11.46%	13.38%	7.36%	5.28%	4.02%
Comics	16.70%	7.05%	7.75%	5.79%	6.64%	3.98%

Table 3: Accuracy Evaluation of Device- and Application-centric Approaches

that the accuracy of a classifier will not suffer due to irrelevant features.

Feature addition: We plotted swipes from users interaction with each application to determine application-specific features. For example, by looking at the users' swipes for the browser and launcher applications, we found out that the distance between two consecutive swipes was a good discriminating feature since users had unique swipe cluster patterns. Similarly, we observed that the orientation of a phone along its x-axis (pitch) also affected a user's input behavior and we used it as a feature for all the applications.

5.2 Accuracy Evaluation

For the accuracy evaluation of the application-centric architecture, we tune the application features as discussed in the last section. We then invoke the classifier based on touchscreen input behavior on each application's data and perform five-fold cross-validation on the training data. To quantify the impact of application-specific training and classification, and feature tuning separately, we report accuracy results for application-centric approach with and without feature tuning. The accuracy results of our evaluation on applications using device- and application-centric approaches are provided in Table 3.

The results show that by using an application-centric approach we are able to reduce FAR by 5%, 6%, 3% and 9% for the launcher, browser, maps and comics applications, respectively. These accuracy gains are observed since a separate behavioral model is created for every application and the variation in application nature does not effect the behavioral model. Another 4%, 6%, 8% and 1% reduction in FAR is recorded when we tune features according to the application nature. In addition to reduction in FAR, a significant and consistent reduction in FRR is also observed when the application-centric approach is used. These results show that by using an application-centric approach, we achieved accuracy improvements in addition to low authentication overhead and fine grained authentication control.

6. LIMITATIONS & COUNTERMEASURES

Some of the limitations of the application-centric approach and possible countermeasures are provided below:

1. The application-centric approach increases development overhead of the application developer. However, this overhead can be mitigated by providing a library as discussed in Section 5.
2. While the application-centric approach requires the same number of test samples for classification when compared to the device-centric approach, the former also requires a separate training model for every application. However, the application-centric approach requires significantly less training samples than the device-centric counterpart. On our dataset, to achieve $\geq$85% accuracy, the application-centric approach only requires 50 training samples whereas the device-centric approach requires at least 275 samples. We also aim to investigate the possibility that there might be classes of applications with broadly similar properties where the training model could be shared within these classes.
3. The application-centric approach can only result in accuracy gains if the underlying classifier employs features that have some correlation with the application (such as touchscreen input pattern [4, 5], keystroke pattern [14], and micro-movement patterns [6] based classifiers). However, if the classifier's features have no correlation with an application's nature, accuracy improvement will not be observed and the accuracy of the application-centric approach will be identical to the accuracy of the device-centric approach.
4. An adversary may write a malicious application to collect the feature values of a victim in order to learn his behavior. The adversary can then gain physical access to the victim's device to launch mimicry attacks [24]. The application-centric approach is as vulnerable to such attacks as the device-centric approach.

7. RELATED WORK

The explicit authentication schemes on smartphones (passlocks, draw-a-secret) have usability issues due to their all-or-nothing access nature [2, 7] and these schemes can be defeated by exploiting OS flaws and shoulder surfing attacks [5]. More recent facial and fingerprint recognition approaches are subject to similar usability issues (due to their all-or-nothing access nature) and have been exploited [22, 21]. These explicit authentication schemes are tangentially related to our work since they are primary defense mechanisms and we only suggest implicit authentication as a second line of defense.

For implicit authentication on smartphones, various behavior-based classifiers have been proposed that employ a user's location patterns [9, 10, 11], call/text patterns [8], keystroke patterns [14], proximity to known devices [7], gait patterns [12], and touchscreen input behavior [3, 4, 5]. Furthermore, some approaches have proposed to combine behavior-based classifiers and contextual information from multiple sources [7, 8, 15] to implicitly authenticate a user. We differ from these prior research efforts since our emphasis is more on where to authenticate than *how* to authenticate. Furthermore, the authors in [16, 17] have proposed implicit authentication frameworks. However, the focus of their work is on designing an efficient framework for probing and storing behavioral features for continuous implicit authentication at the device level. We fundamentally differ from these approaches due to our application-centric nature.

Finally, there are some approaches that have a common goal with ours in terms of providing an optimal trade-off between usability and security. For example, Hayashi et al. [2] discuss the inefficiency of the all-or-nothing access model and suggest that a user should be authenticated only when a sensitive application is launched. They also discuss shared access scenarios and propose an activity lock that can be used by a device owner to share specific screens in an application or a group account to share a specific set of applications between multi-user environments. In a related work, Hayashi et al. [19] use multiple implicit factors to determine how to explicitly authenticate a user. For example, if a user is at a well known location (such as home), quick and easy explicit authentication is used. A similar scheme has been proposed by Gupta et al. [20] for context profiling to determine authentication control. However, our work is different from these approaches since in addition to selectively invoking an authentication module based on the type of an application, we aim to delegate implicit authentication tasks to an application and not to the device.

8. CONCLUSION & FUTURE WORK

We have proposed a novel application-centric approach for providing implicit authentication support on smartphones. We recommend that the decisions of when to authenticate, which behavior-based classifiers to authenticate with and feature tuning of the classifier should be delegated to the application. While this delegation increases development overhead on the application provider (which can be mitigated using a library) it reduces authentication overhead and provides fine grained authentication control. Empirical evaluations of the proposed application-centric approach show that it provides significant accuracy improvements as compared to the device-centric counterpart. Application-centric implicit authentication is part of our on-going research work and in addition to verification of our claims on other implicit authentication schemes, we are developing a library that will enable application developers to effortlessly provide implicit authentication support in their applications.

9. ACKNOWLEDGEMENTS

We thank Mario Frank, for his helpful comments, and for providing the sourcecode of the Touchalytics classifier [4]. We also thank Google and NSERC for their support.

10. REFERENCES

[1] N. Ben-Asher, N. Kirschnick, H. Sieger, J. Meyer, A. Ben-Oved, and S. Moller, “On the need for different security methods on mobile phones”, MobileHCI, 2011.

[2] E. Hayashi, O. Riva, K. Strauss, A. J. Brush, and S. Schechter, “Goldilocks and the two mobile devices: going beyond all-or-nothing access to a device’s applications”, SOUPS, 2012.

[3] A. D. Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann, “Touch me once and I know it’s you!: implicit authentication based on touch screen patterns”, CHI, 2012.

[4] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song, “Touchalytics: on the applicability of touchscreen input as a behavioral biometric for continuous authentication”, IEEE Transaction on Information Forensics and Security, 8(1), 2013.

[5] L. Li, X. Zhao, and G. Xue, “Unobservable re-authentication for smartphones”, NDSS, 2013

[6] C. Bo, L. Zhang, X. Li, Q. Huang, and Y. Wang, “SilentSense: silent user identification via touch and movement behavioral biometrics”, MobiCom, 2013.

[7] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos, “Progressive authentication: deciding when to authenticate on mobile phones”, USENIX Security Symposium, 2012.

[8] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, “Implicit authentication through learning user behavior”, Int. Conf. on Information Security, 2011.

[9] M. L. Damiani and C. Silvestri, “Towards location-aware access control”, International Workshop on Security and Privacy in GIS and LBS, 2008.

[10] J. Seifert, A. D. Luca, B. Conradi, and H. Hussmann, “TreasurePhone: context-sensitive user data protection on mobile phones”, Pervasive, 2010.

[11] A. Studer, and A. Perrig, “Mobile user location-specific encryption (MULE): using your office as your password”, WiSec, 2010.

[12] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S. M. Makela, and H. Ailisto, “Identifying users of portable devices from gait pattern with accelerometers”, ICASSP, 2005.

[13] A. Kalamandeen, A. Scannell, E. D. Lara, A. Sheth, and A. LaMarca, “Ensemble: cooperative proximity-based authentication”, MobiSys, 2010.

[14] N. L. Clarke, and S. M. Furnell, “Authenticating mobile phone users using keystroke analysis”, Springer Int. Journal of Information Security, 6(1), 2007.

[15] A. Shabtai, U. Kanonov, and Y. Elovici, “Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method,” Journal of System Software, 83(8), 2010.

[16] H. Crawford, K. Renaud, T. Storer. “A framework for continuous, transparent mobile device authentication”, Elsevier Computers & Security, 39(2), 2013.

[17] N. Clarke, S. Karatzouni, and S. Furnell, “Flexible and transparent user authentication for mobile devices”, Springer Emerging Challenges Sec., Priv., Trust, 2009.

[18] R.H. Woo, A. Park, and T. J. Hazen. “The MIT mobile device speaker verification corpus: data collection and preliminary experiments”, Speaker and Language Recognition Workshop, 2006.

[19] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley. “Casa: context-aware scalable authentication”, SOUPS, 2013.

[20] A. Gupta, M. Miettinen, N. Asokan, and M. Nagy, “Intuitive security policy configuration in mobile devices using context profiling”, PASSAT, 2012.

[21] Apple iPhone fingerprint reader confirmed as easy to hack, `http://zdnet.com/apple-iphone-fingerprint-reader-confirmed-as-easy-to-hack-7000021065/`

[22] Android Jelly Bean Face Unlock hacked, `http://androidauthority.com/android-jelly-bean-face-unlock-blink-hacking-105556/`

[23] S. Kullback, and R. A. Leibler, “On information and sufficiency”, Annals of Mathematical Statistics 22, 1951.

[24] A. Serwadda and V. V. Phoha, “When kids’ toys breach mobile phone security”, CCS, 2013.