Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

From keyloggers to touchloggers: Take the rough with the smooth

*D. Damopoulos**, *G. Kambourakis*, *S. Gritzalis*

Info-Sec-Lab Laboratory of Information and Communications Systems Security, Department of Information and Communication Systems Engineering, University of the Aegean, Samos GR-83200, Greece

ARTICLE INFO

Article history:
Received 1 June 2012
Received in revised form
16 September 2012
Accepted 15 October 2012

Keywords:
Behavioural biometrics
Keyloggers
Touchloggers
Smartphones
Malware
iOS
Security

ABSTRACT

The proliferation of touchscreen devices brings along several interesting research challenges. One of them is whether touchstroke-based analysis (similar to keylogging) can be a reliable means of profiling the user of a mobile device. Of course, in such a setting, the coin has two sides. First, one can employ the output produced by such a system to feed machine learning classifiers and later on intrusion detection engines. Second, aggressors can install touchloggers to harvest user's private data. This malicious option has been also extensively exploited in the past by legacy keyloggers under various settings, but has been scarcely assessed for soft keyboards. Compelled by these separate but interdependent aspects, we implement the first-known native and fully operational touchlogger for ultramodern smartphones and especially for those employing the proprietary iOS platform. The results we obtained for the first objective are very promising showing an accuracy in identifying misuses, and thus post-authenticating the user, in an amount that exceeds 99%. The virulent personality of such software when used maliciously is also demonstrated through real-use cases.

1. Introduction

Over the last few years, mobile devices have gained increasing popularity due to the variety of the data services they offer, such as texting, emailing, browsing the Internet, document editing, playing games, along with the traditional voice services. Such devices, commonly referred to as smartphones, are getting constantly smaller, cheaper, more convenient and powerful, and are able to provide a plethora of advanced data input interfaces enabling the user to interact with the device more productively. Typical examples of such advanced features include software keyboards displayed on a touchscreen instead of hard ones, magnetometer and gyroscope for measuring or maintaining the orientation of the device etc.

However, at the same time, modern smartphones comprise an attractive target for any potential intruder or malicious code. On the one hand, such expensive devices are attracting the attention of occasional or even petty thieves. Note that the target of such incidents may not only be the device itself (e.g., sell it for profit) but in some cases the data stored on it. On the other hand, ultra-portable devices now represent a promising target for malware developers that struggle to expose users' sensitive data, compromise the device or manipulate popular services (Polla et al., 2012). Also, as detailed in Section 4, other types of traditional malware seem to also evolve in an effort to catch up with the so called *mobile era*.

Under this prism, it is obvious that with the exception of a limited number of very expensive devices, the majority of

* *Corresponding author*. Tel.: +30 22730 82281.
E-mail addresses: ddamop@aegean.gr (D. Damopoulos), gkamb@aegean.gr (G. Kambourakis), sgritz@aegean.gr (S. Gritzalis).
0167-4048/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2012.10.002

smartphones still use traditional authentication and access control methods such as Personal Identification Number (PIN), Screen Lock Password (SLP), which in many cases are not sufficient to offer integral protection against intrusions. To exemplify this, it is certain that such approaches do not safeguard the private data on stolen devices after authentication has been carried out (post-authentication state). In the simplest case, if a mobile device comes under the possession of the attacker and is in an unlocked state, private data can be exploited. In these situations it is desirable to equip the device with a mechanism able to constantly track and identify its owner(s) behavior and thus enable it to detect misuses by itself. Consequently, and as discussed in Section 5, the research community is increasingly interested in developing intelligent post-authentication controls based on biometric technologies for bolstering the security of mobile devices. To this end, keystroke analysis can be a fruitful means of identifying (profiling) the legitimate user of a mobile device. Actually, as detailed further down in Section 5, this option has been investigated in the past, but for mobile devices equipped with physical keyboards. This possibility however has hardly been explored for modern smartphones having a touchscreen to interact with the user. Note that the data produced when using such an interface is of great amount and diversity. That is, the touch data does not solely originate from the soft keyboard of the device but from all, say, finger movements the user makes on the device's display (e.g., sliding movements including up/down gestures).

The potential applications of such touch-oriented user profiling method are numerous and varied. For instance, touchlogging data can be used to block unauthorized access to a device; The touchlogger software module running on the device realizes that the person using the device is not the owner and reacts by locking it. User personalization is another field where touchlogging may be proved very handy. Let us assume, for example, a touchlogger tracking the responses of a user during their interaction with an application, say, a game. If the user interacts with the screen aggressively maybe it is a sign that the game must (automatically) bring down its difficulty. On the negative side, a touchlogger can be exploited by attackers to harvest sensitive user information such as passwords, account numbers, emails, social security numbers etc.

Our contribution: This work focuses on touchloggers for modern mobile devices. Our aim is twofold. First, to show that touchlogging can be a reliable and very accurate means of profiling the legitimate user(s) of a device. This means, for example, that the touch events collected by the touchlogger can be readily utilized by behavioral-based Intrusion (IDS) to detect misuses and/or intrusions (although the evaluation of such an IDS remains out-of-scope of this paper). Second, to demonstrate that when compared to traditional keyloggers, a touchlogger can be at least equally hazardous to the user(s) of the device. Toward these goals, we implement a full-fledged touchlogger for devices on the iOS platform (formerly known as iPhone Operating System). The log files of the touching events are then fed to popular machine learning algorithms to classify user behavior with the aim to assess the feasibility of this type of software to be used as the core part of some sort of user post-authentication mechanism. As far as we are aware of, this is the first work on touchloggers in literature. Finally, a side contribution of the paper is to provide a comprehensive review of the state-of-the-art in this area of research.

The remainder of the paper is organized as follows. The next section brings into the foreground design challenges and basic requirements toward realizing touchlogging software. Particular emphasis is given on the iOS platform. User profiling based on touchlogging data is assessed in Section 3. Section 4 gives some preliminary results based on real-use cases that prove the malevolent potential of this type of software. Related work is addressed in Section 5. The last section concludes and outlines future work.

2. Design challenges, requirements, and implementation guidelines

In practice, several obstacles must be surpassed before one is able to collect the touch events happening on the display of a device. This is because smartphone OS restrict privileges granted to applications. In most cases, an application cannot acquire touchstrokes unless it is active and receives the focus on the screen. This alone makes the collection of touch events highly difficult. Also, in contrast to the typical mobile device with a (fixed) hardware keyboard and a small display screen, a touchscreen mobile device uses all the surface of the screen to display software views, buttons, check boxes, radio buttons or soft keyboard as the input data interface to the user.

Nowadays, Google's Android and Apple's iOS are dominating the market of touchscreen-equipped smartphones. Considering these two disparate options, we selected the most challenging one, that is to implement a touchlogger using the proprietary iOS platform. In fact, both the aforementioned OS restrict access to their internal functions. However, in contrast to iOS, the Android source code is freely available for download and tinkering.

The primary aim of a touchlogger is to collect every touch event taking place on the screen. To do so, it needs to fulfill two fundamental requirements: (a) gain root permissions to be able to hook and override internal OS methods which are responsible for the detection and management of touch events, and (b) run in the background of the OS and constantly track and collect user's touch behavior.

Rightly or wrongly, only software signed by Apple's Certificate Authority is allowed to run on an iPhone or iPad device. So, taking into account the first requirement, the only way to gain root permissions on an iOS device is by exploiting an existing vulnerability. This process is generally referred to as *Jailbreak* (rooting on Android). Jailbreaking allows the creation and execution of third-party software without an official SDK from Apple. Moreover, Apple does not offer any frameworks that override iOS Application Programming Interface (API) methods. To fill the gap, J. Freeman has created the *MobileSubstrate* extension, a framework that allows developers to deliver run-time patches to system methods using Objective-C dynamic libraries (*dylib*) (iDW, 2012). Also, D. L. Howett has contributed *Theos*, a cross-platform suite of programming tools for managing and deploying jailbreak-oriented iOS development (iDW, 2012). By creating a dylib and linking it with the MobileSubstrate extension, developers

are able to build applications capable of hooking internal system methods.

Considering the second requirement, the touchlogger needs to run continuously in the background of the underlying OS. Until iOS ver. 4, multitasking was not officially supported. Nevertheless, jailbroken iOS could support applications that run in the background as daemons or use Objective-C dylib. Also, iOS, being a Unix-based OS, can provide multitasking by using *launchd*, a launch system that supports daemons and per-user agents as background services. After iOS has been jailbroken, any installed application or shell script is able to behave as daemon by creating a launch plist and placing it into the iOS "/Library/LaunchDaemons" folder. Another way to support multitasking is with dylib. When launching an application, the iOS kernel loads the application's code and data into the address space of a new process. At the same time, the kernel loads the dynamic loader, that is "/System/MobileSubstrate/DynamicLibraries" into the process space and passes control to it. It is also possible to load a dylib at any time through an Objective-C method. This is actually the main reason we use this backgrounding method for implementing our touchlogger. Note that starting from ver. 4 Apple provided seven APIs that allowed applications to run in the background. Although these APIs are the native method for providing multitasking, as it is explained in (Damopoulos et al., 2011), is not the best way to create and launch a touchlogger application.

Given the above restrictions, we implement iTL a fully-fledged touchlogger for iOS devices. iTL is written in Objective-C and compiled with Theos for iPhone ARM CPU and tested to run on iOS ver. 4 and above. As already pointed out, iTL has been implemented using the unofficial ways for backgrounding (dylibs), the public and private frameworks and the MobileSubstrate framework, with the *substrate.h* header that overrides iOS methods (The iPhone Wiki, 2012). Fig. 1 depicts the overall iTL architecture and details how it interacts with the touchscreen and the iOS. Note that while the aim of this paper is not to elaborate on implementation details about the developed prototype, some details about its internal mechanics should be discussed here for facilitating the reading of the next sections and for the sake of completeness.

The main application that manages the iOS home screen is SpringBoard (iDW, 2012). The User Interface Kit (UIKit) framework is responsible for handling user interaction through the touchscreen with SpringBoard or any other application. The same framework also includes a set of standard subclasses a user can utilize, which range from simple buttons to complex tables. The User Interface View (UIView) class defines a rectangular area on the screen and the interfaces for managing the content in that area. At runtime, a view object handles the rendering of any content in its area and also takes care of any interactions with that content. Because an application interacts with the user primarily through UIView objects, these objects have a number of responsibilities such as drawing and animation, management of layout and subview, and event handling. Each UIView object acts as a responder that handles touch events also known as User Interface Events (UIEvent). A UIEvent is defined by the User Interface Responder (UIResponder), an interface for objects that is able to detect touch events and at the same time handle common gestures.

Every time, say, a finger touches, is dragged on, or is lifted from the screen, the digitizer, a thin film over the device display, tries to determine the shape of the touch area in order to calculate the exact location of the touch and instantiate an UIEvent object (Fig. 1(a)). Then, all UIEvent objects get grouped (Fig. 1(b)). Each UIEvent object contains User Interface Touch (UITouch) objects for all the fingers on the screen or just lifted from it.

The general syntax for UIEvent-handling API methods (Fig. 1(d)) for managing touch events are (Apple Inc, 2012):

- touchesBegan:withEvent: (one or more fingers touch down in a view)
- touchesMoved:withEvent: (one or more fingers associated with an event move within a view)
- touchesEnded:withEvent: (one or more fingers are raised from a view)
- touchesCancelled:withEvent: (when a system event – such as a low-memory warning – cancels a touch event)

Overall, the parameters of these methods associate touch object instances with their events – especially instances that are new or their field values have changed – and thus allow UIEvent (responder) objects to track and handle the touches as the delivered events progress through the phases of a multi-touch sequence.

This is the phase where iTL intervenes. Normally, after the UIEvents get grouped, the primary UIEvent-handling methods for touches inform the UIView object about the detected touch event or the multi-touch gesture (Fig. 1(f)). The main difficulty in acquiring touchstrokes is that each application (either native or custom) has its own UIview object. The direct effect of this problem is that one cannot collect touch events unless the application is active and receives the focus on the screen. The novelty of our implementation is that we intervene and hook against the UIView class, thus managing to collect all touch events regardless if the application UIView focus is on the screen or not (Fig. 1(c)). Also, we override the primary UIEvent-handling methods for touches toward defining the exact location of the touch which drags, lifts, moves or gets canceled from the screen. Furthermore, as depicted in Fig. 1, iTL hooks against: (a) the UIKeyBoard class by overriding the activate/deactivate method to detect when the virtual keyboard is activated or deactivated, and (b) the Springboard SBApplicationIcon class by overriding the launch method to detect which application is activated by the user each time.

Recall, that a touchlogger can be used both defensively and offensively. So, iTL has been designed in line with this goal. It consists of two modules namely iGestureLogger (iGL) and iKeylogger (iKL). The first one is responsible to track every touch event or gesture happening on the device's display in an effort to collect enough data to build the user's profile for use by, say, an IDS. The other, tries to identify touch events that occur inside the area of a pre-defined soft keyboard. Then, it attempts to translate every touch to the corresponding (actual) key. If not, the corresponding touch event is discarded. These two modules are depicted in Fig. 1 (as (d) and (e) respectively) and as we can observe, they trigger different

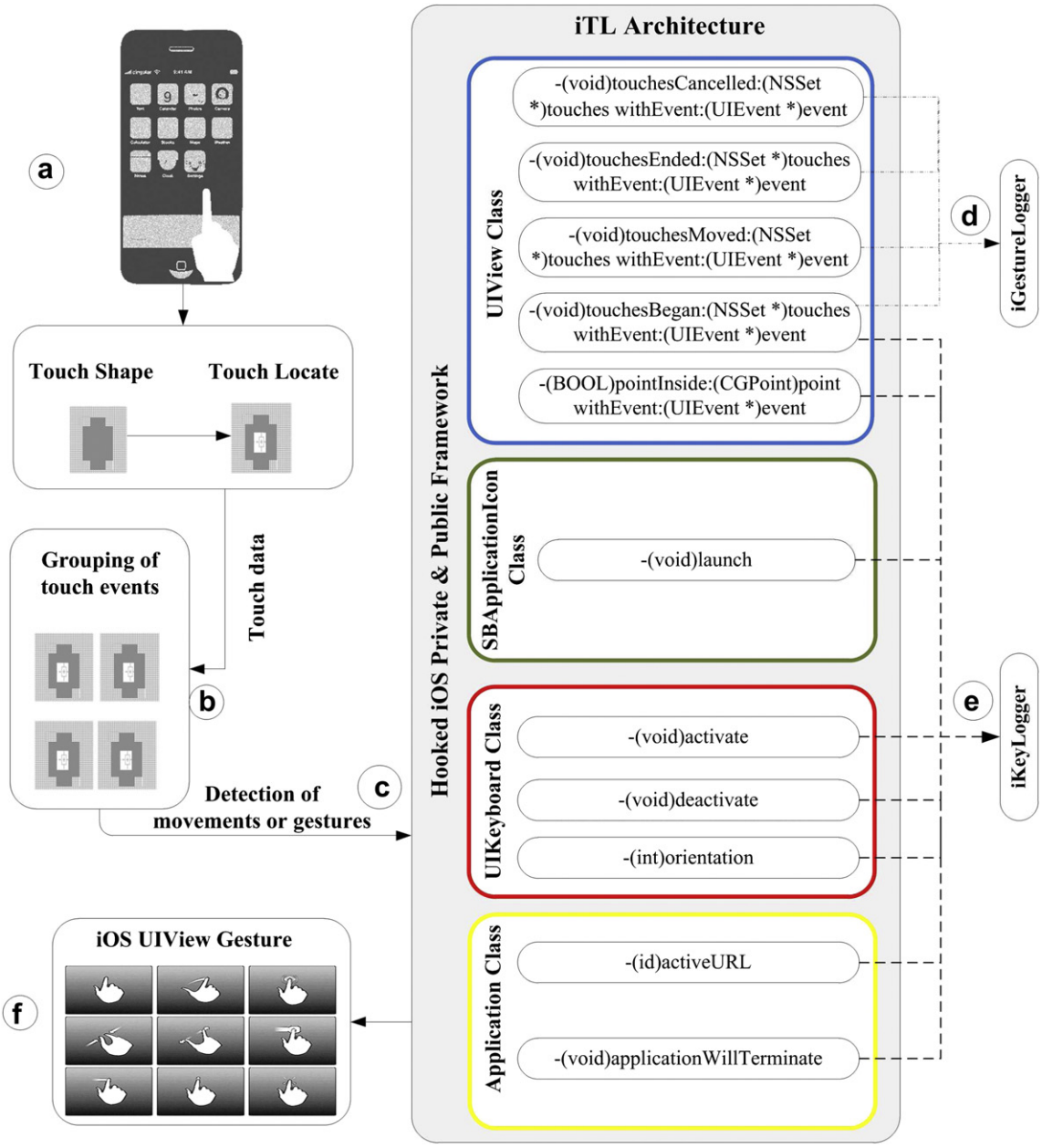

Fig. 1 – iTL high-level architecture (note that all classes are being hooked while the class methods are being overridden).

methods but one. Also note that these modules can operate either in tandem or independently.

3. User profiling based on touch patterns

Still today, the majority of mobile devices consist of a hardware keypad and a small screen as the I/O interfaces for the user. In most cases, to interact with the OS menu a user needs to utilize specific hardware buttons (e.g., up, down, right, left). Additionally, the user employs the hardware keyboard to write text messages, emails etc. Based on this fact, so far, all keystroke analysis systems but one presented in the literature capitalize on physical keyboard data and more specifically those collected during texting or other text entering activities to authenticate the user. As detailed in Section 5, Cai and Chen do consider touchlogging but in an indirect way demonstrating that motion is a significant side channel, which may leak confidential information on smartphones (Cai and Chen, 2011).

So, we can argue that iTL reinvents and expands keystroke logging but for touch-based surfaces and sets new directions on the data and features need to be used to (post)authenticate, say, a smartphone user. Putting it another way, post-authentication requires building the profile of the legitimate user based on touch (behavioral) patterns. The iGL module accomplishes this by detecting and logging every touch event generated by the user the exact same time that they interact with the OS, e.g., when writing messages or using applications in general. Therefore, the output produced by such a system

can be fed to an IDS to detect misuses. However, before everything else, we need to assess the effectiveness of such system in correctly classifying user's touching behavior. This is achieved with the help of machine learning techniques. This process would provide evidences of the potentiality of touchlogging to be used as the core part of any post-authentication scheme or mechanism. The following two sections discuss our methodology and present the results.

3.1. *Methodology and data structure*

We used iGL to collect touch events generated by eighteen participants (iPhone owners) in the age range of 22–36 years. Each person used their own device for 24 h performing their usual everyday activities. After the data collection process ended, the behavioral log files were retrieved from the devices. To ease the data collection and acquisition process we implemented an application shell for iGL (Damopoulos et al., 2012). Once downloaded and installed by the user, the application collects touch data for 24 h. Then, it will automatically try to connect via Wi-Fi to the Internet to transmit anonymously the log file to our server.

Each file contains an arbitrary number of records where each of them corresponds to a vector of related features per touch event as described in Fig. 2. For the classification process to take place, each file contains the data of the corresponding legitimate user and the data of the rest seventeen users that represent the potential intruders. This means that for each user in the dataset, the corresponding data file contains: (a) the user's data, referred to as normal behavior data, and (b) all other users' data that represent potential intrusive behaviors. Every record of the touch data file is composed of collected features represented by the following quintuplet: *{Type, X, Y, Timestamp, Intruder/Legit}*. Where *Type* refers to the type of the event, *X*, *Y* correspond to the Cartesian coordinates where the event took place, *Timestamp* refers to a UNIX timestamp (based on seconds since the standard epoch of 1/1/1970) representing the date and time a touch event occurred, and *Intruder/Legit* is the binary representation of the two nominal classes, i.e., if this piece of data belongs to the legitimate user (no) or the intruder (yes). An example of such a record is given by the following quintuplet {*B, 289.000000, 315.000000, 1338039343.262504, no*}.

The analysis procedure takes into account and cross-evaluates four supervised machine learning algorithms, i.e., Bayesian Networks, Radial Basis Function (RBF), K-Nearest Neighbor (KNN) and Random Forest. Also, for all the experiments, the k-fold cross-validation method – and more specifically a 10-fold one (this option provides us with more chunks of data to work with) – has been employed to divide the dataset into different sub-samples. This means that the original sample is randomly divided into k equally (or nearly equally) sized sub-samples, and the cross-validation process is repeated k times (the folds). Each time, one of the k sub-samples is used as the test set while the other k-1 sub-samples are put together to form the training set. Finally, the average error across all k trials is computed.

The analysis of the collected data has been performed on a laptop machine with an 2.53 GHz Intel Core 2 Duo T7200 CPU and 8 GB of RAM. The OS of this machine is OS X Mountain Lion. The experiments have been carried out using the well known machine learning software package namely Waikato Environment for Knowledge Analysis (Weka, 2012). The upper memory bound has been set to 1 GB aiming to resemble the memory reserves of a modern smartphone.

3.2. *Results*

Legacy keystroke analysis uses two error rates, namely False Acceptance Rate (FAR), in which an intruder is accepted by the system, and False Rejection Rate (FRR), in which the authorized user is rejected by the system (Bergadano et al., 2002). A third metric known as Equal Error Rate (EER) is also employed in literature to assess the potential of a keystroke system. Specifically, EER is a kind of percentage rate which both accepts and rejects errors as equals (EER = (FAR + FRR)/2). That is, the lower the error rate value, the higher the accuracy of the system. In our analysis, we consider all three aforementioned metrics to estimate the effectiveness of touchstroke-based classification.

Fig. 3, summarizes the FAR% and FRR% metrics logged per participant, per classifier. Also, Table 1 contains the maximum and minimum as well as the average and standard deviation values of each of the aforementioned metric per classifier, but this time calculated for all the participants. We easily observe that Bayesian Networks and Random Forest

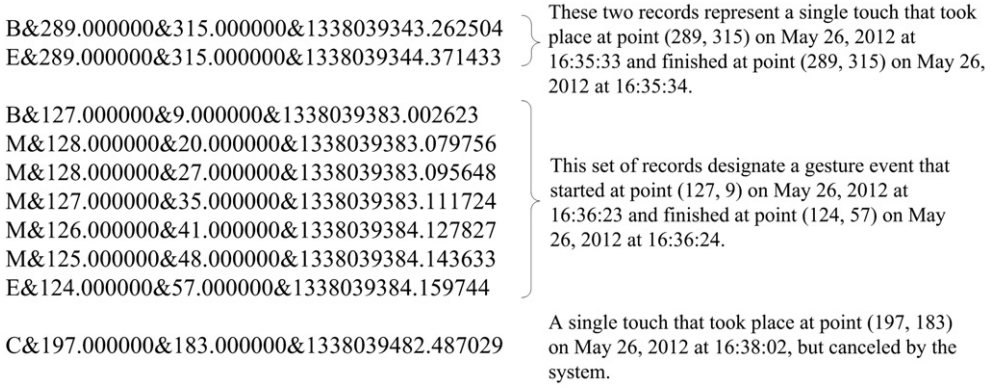

Fig. 2 – iGL log file example records (B=Begin, M = Move, E = End, C=Cancel. The character & is used as a separator between the fields).

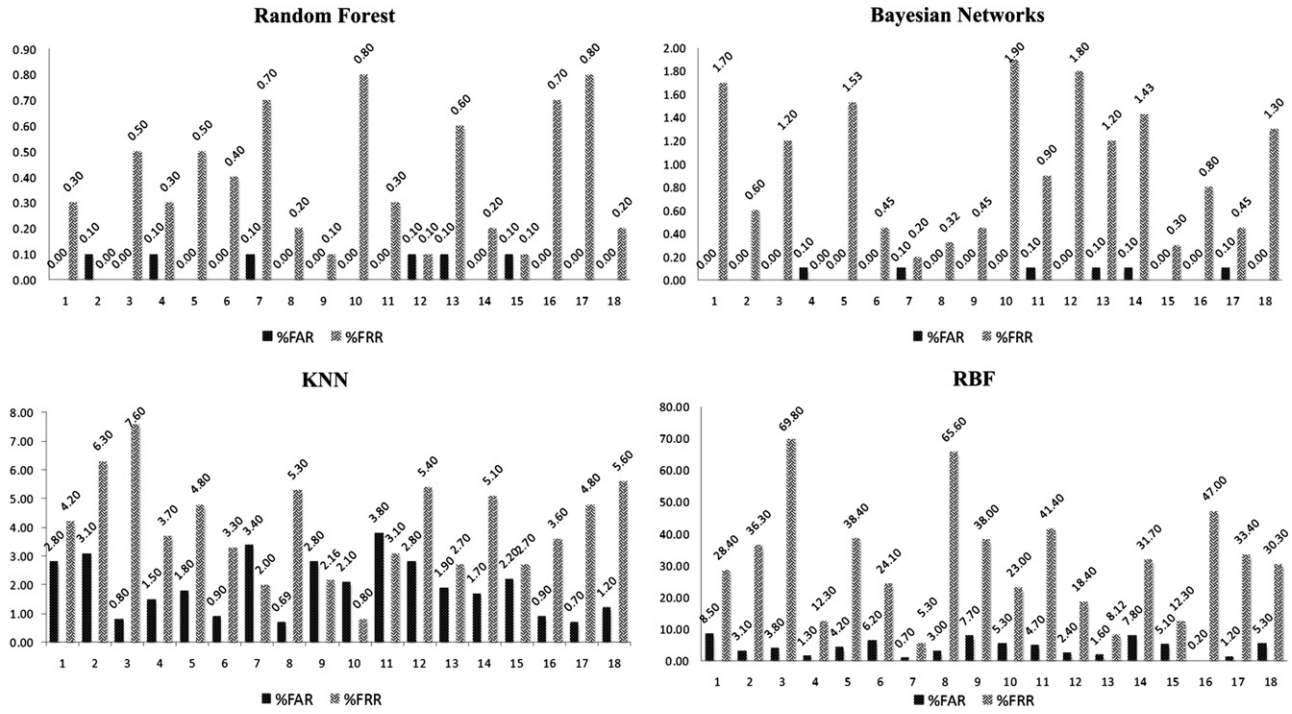

Fig. 3 – FAR% and FRR% metrics per participant per classifier.

obtained very competitive results when compared to those scored by RBF and KNN. Specifically, the maximum FAR%, FRR % value pairs for Bayesian Networks and Random Forest are {0.1, 1.9} and {0.1, 0.8} correspondingly. This means that in the worst case, an intruder is rejected by the system in a percent equal to 99.9%. Also, taking into account the percentages scored by Random Forest, the system accepts the legitimate user in 99.2% of the cases. Overall, Random Forest seems to be the best choice as the results it produced are optimal across all the metrics. This observation is further validated by the calculated standard deviation values, that is, 0.05% and 0.26% for the FAR and FRR metrics respectively. As a consequence, the EER% for this algorithm is the lowest (0.205%) when compared to those scored by Bayesian Networks (0.475%), KNN (3.005%), and RBF (17.67%). These results clearly illustrate the adequacy of the proposed touchlogging scheme to be used toward identifying misuses.

To further exemplify the above findings, in Fig. 4 we cross-projected the touch profiles of three different, randomly selected, participants. Bear in mind that each profile - compiling the touch events of a whole day - is actually a series of Cartesian coordinates as recorded by iGL in the corresponding behavioral log file for that user. From the figure, it becomes apparent that each behavioral profile is too far from being characterized as similar to the others. In fact, when examining the dataset, there is no touch profile that can be said to be close to one another. Of course, this is quite plausible because, each user employs and personalizes very differently their smartphone. For example, they put the applications icons in different places on the screen (or inside different folders), create variant interfaces for their applications, and have their own repertoire of sliding movements/gestures etc.

Taking into account the above findings, we can safely argue that touch-based behavior classification presents significantly better results compared to keystroke studies for mobiles devices presented in literature so far (see Section 5 for details on these works). This naturally stems from the fact that iGL collects every touch event happening on the screen and not just those associated with the virtual keypad. So, even for relatively short-term interactions with a device (as in this study), touchlogging seems to be able to profile the user with very high accuracy.

Table 1 – Aggregated classification results (all participants).

	Random Forest		Bayesian Networks		KNN		RBF	
	%FAR	%FRR	%FAR	%FRR	%FAR	%FRR	%FAR	%FRR
Mean	0.03	0.38	0.03	0.92	1.95	4.06	4.01	31.32
Min	0.00	0.00	0.00	0.00	0.69	0.80	0.20	5.30
Max	0.10	0.80	0.10	1.90	3.80	7.60	8.50	69.80
St. Dev	0.05	0.26	0.05	0.60	0.99	1.71	2.53	17.76

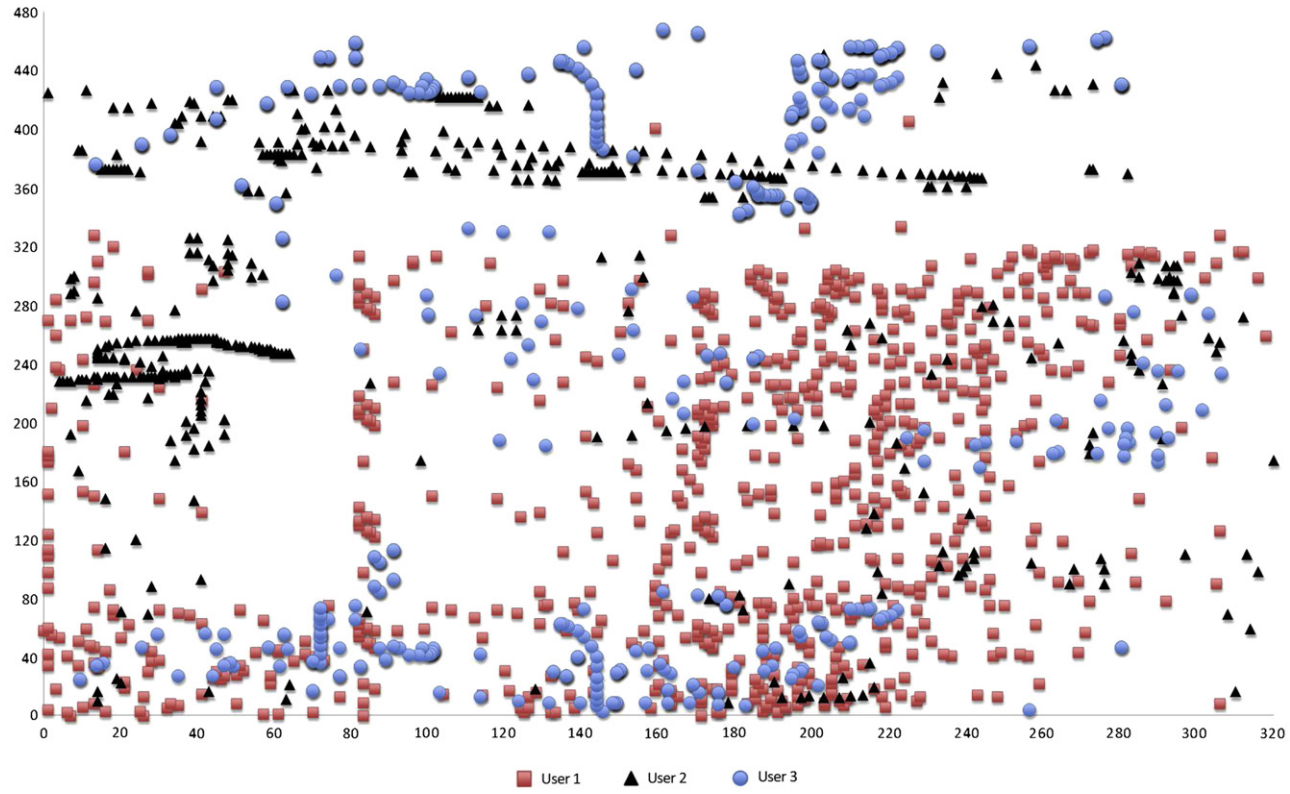

Fig. 4 – Cross-projection of 24-h touch profiles corresponding to 3 different users (the plot area recreates the iPhone screen resolution 320 × 480).

It is also to be noted that while these results provide strong evidences that touchstroke-based classification may be a very accurate means of profiling the user, more research is needed to better assess its potential. For example, iGL can be used to create the profile of a given user based solely on touch events collected when the user interacts with a particular application(s), say, the SpringBoard. Also, extended experimentation could take into account more participants using their devices for longer periods of time, e.g., a week, or augment the sample to include subjects of different age, sex etc. Clustering could also be an interesting research direction, e.g., perform classification based on touch events collected during particular periods of time when using certain types of applications. Nevertheless, as already pointed out, this is the first work on touchlogging. So, its scope is narrowed down to analyze basic touchlogger implementation aspects and thus bound to make an initial assessment of the competence of such software.

4. The flip side of the coin: turning malware

Keyloggers can be classified into hardware keyloggers, where a tiny electronic device is used to log the data between a keyboard and the I/O port, and software keyloggers where a software program hooks the methods of the keypad in order to monitor the pressed keys (Sagiroglu and Canbek, 2009). The focus of this paper is on software keyloggers which are variously known as tracking software, keystroke monitor systems, keyboard sniffers etc. This kind of software is also embedded in seemingly innocuous and useful applications in the form of spyware (Sipior and Ward, 2008; Zaitsev, 2010). In any case, the primary aim of a keylogger is to share system resources with legitimate programs remaining hidden while recording passwords, private conversations or e-mails (Sreenivas and Anitha, 2011). In a nutshell, keyloggers for both fixed computers and mobile devices are expected to share the same basic architecture with the only difference that the first hook the keyboard methods from a hardware keyboard while the latter from a software one. Very recently, we have witnessed a limited number of touchlogging commercial software even for (jailbroken) iPhone devices (iKeyGuard, 2012). However, these solutions are able to monitor only the native soft keyboard of the iOS and therefore incapable of recording touches that occur in custom-made virtual keyborads used by many websites (e.g., those used by mobile banking websites). This means that in contrast to what is propossed in (Cai and Chen, 2011) and other keylogging schemes for hard or soft keyboards, iKL does not hook any keyboard methods to log the pressed key. Instead, it locates the touched point (actually, a small rectangular surface) on the screen and translates that location to the actual key pressed on the virtual keypad, based on a pre-defined module as explained in the following. More specifically, iKL hooks against SBApplicationIcon, UIKeyboard, Application, and UIView classes and overrides the corresponding class methods which are (Fig. 1(e)):

- *-(void)launch*; launches an application
- *-(void)touchesBegan:withEvent:*; detects the beginning of a touch event

- *-(BOOL)pointInside:withEvent:*; returns true on a touch event and false otherwise
- *-(void)active*; activates the soft keyboard
- *-(void)deactive*; deactivates the soft keyboard
- *-(int)orientation*; returns 0 if portrait or 1 otherwise (landscape)
- *-(id)activeURL*; returns the URL loaded in Mobile Safari
- *-(void)applicationWillTerminate*; terminates the Mobile Safari application

The iTL routine constitutes of two main parts; the *Location Module Manager* (LMM) and the *Location Module(s)* (LM). The first is responsible for deciding which LM is appropriate to capture key-touch events depending on the case, while the second is in charge of detecting a touch event and translating it to the pre-defined key. iTL is developed as a dylib able to run continuously in the background of the OS staying hidden from the legitimate user. Because of that, it can easily fuse with other malware like iSAM (Damopoulos et al., 2011) and be part of a botnet. Also, iKL can be programmed so as to automatically "tweet" the intercepted data (log files, images) to a private Twitter account (e.g., by using the native SDK provided by Twitter) or to a server that is under the control of the attacker.

Based on the loaded application, the LMM attempts to define which LM will be dynamically loaded. After that, it is up to that LM to realize which (virtual) key has been pressed. LMs are dylibs which map pre-defined touch locations to the virtual keys of the (soft) keyboard. It is therefore deduced that LMs can be created and added by the attacker based on the keyboard interfaces they desire to trace. By default, iKL has two LMs, namely KeyLandscape (key_land.dylib) and KeyPortaint (key_port.dylib). That is, the KeyLandscape module contains pre-defined key locations for the native landscape virtual keyboard, while KeyPortaint contains the same information but for the native portrait virtual keyboard.

First off, the SBApplicationIcon, Application, and UIKeyboard API class methods have been hooked by the LMM. To successfully hook the methods of the first two aforementioned private classes, it was necessary to class-dump both the SpringBoard and Mobile iOS applications and retrieve the class headers. Based on the loaded application the LMM tries to perceive if the virtual keyboard is active or not (by default all iOS applications use the native virtual keypad). Therefore, if the virtual keyboard pops up, the LMM checks the orientation (landscape or portrait) of the device and loads KeyLandscape, if landscape, or KeyPortaint otherwise. Once the LM is loaded, iTL attempts to define if the location of the touch point is within the rectangular area that confines a virtual key. This area is defined by four Cartesian coordinates per key. If true, then the corresponding key will be logged into a text file. Keep in mind that native iOS soft keyboard consists of four levels. That is, lowercase alphabetic keys, uppercase alphabetic keys, numeric keys and symbol keys corresponding to levels 0, 1, 2, and 3. Both the KeyPortaint and KeyLandscape modules are able to perceive to which level of the iOS keyboard the user is touching and hence log the correct key.

To test iTL, we conducted three real use-cases which are described further down. It is stressed that all experiments had 100% accuracy in logging the keys (usernames, passwords etc), thus bypassing any security mechanisms such as https sessions or custom virtual keyboards presented by websites.

4.1. *Scenario I: a sitting target*

According to this scenario, we used an iOS proprietary application for m-banking transactions developed by (EFG, 2012). To login into their bank account, the user needs to type their credentials using the native iOS keyboard. Once a finger touches on a text entering box, the virtual keyboard gets activated and at the same time the LM loads the KeyPortaint module. Since then, all pressed keys will be logged. In case the user changes the orientation of the device to landscape the KeyLandscape module is automatically loaded. Using Safari mobile we visit a bank website (EFG Eurobank, 2012) to make some m-banking transactions. Once more in order for a user to login into their account, they need to type their credentials using the iOS native keyboard. Depending on the orientation of the device, the KeyPortaint or KeyLandscape will log again the credentials.

4.2. *Scenario II: dealing with zooming*

In this second scenario we developed another, but this time more intelligent LM, namely KeyVirtual (keyvirtual.dylib) that works only with a specific bank website (Syndicate Bank, 2012), and is able to detect the keys from any virtual custom keyboard presented by this website. Once the LMM detects Mobile Safari and (Syndicate Bank, 2012) as the loaded URL, loads the KeyVirtual module. This time, the module contains the pre-defined location of all virtual keys presented by the website when: (a) the zoom level is set to zero, and (b) the page is aligned to the center of the device's display. Every time the user performs a zoom in or out to the webpage or tries to relocate the position of the website view, the module recalculates on-the-fly the pre-defined key locations based on the new zoom level and the new webpage view position. So, to prepare for the attack, the aggressor must first perform a degree of reconnaissance to record the layout of the virtual keyboad the website of interest uses. The *Hovering keyboard* is another retaliatory tactic used against keylogging when a mouse is available. Specifically, this method enables the user to enter their private information (e.g., a password) by just pointing the mouse on the relevant characters. This is also known as "MouseOver". However, as it is obvious, in our case this method does not prohibit adversaries from spying because of the touchscreen.

4.3. *Scenario III: evading scrambled keyboards*

In this last scenario we developed an even smarter LM able to bypass the state-of-the-art security mechanism, namely *Scrambled keyboard* employed usually by bank websites as a last resort protection against keylogging. The Scrambled keyboard is a server-side script that implements a keyboard that is both virtual and dynamic in nature. This means that the position of characters displayed on the virtual keyboard changes every time the user touches on a key. Once again, the LM must be designed especially for the target-website.

To demonstrate this situation we implemented an LM, namely KeyScram (keyscram.dylib) that works only with mobile banking services offered by a specific bank (PanCaribbean Bank, 2012). More precisely, once the LMM detects Mobile Safari and (PanCaribbean Bank, 2012) as the URL, it automatically loads the aforementioned virtual module. Recall from the previous subsection that normally an LM stores the static location for the website's virtual keys along with their names. For instance, if the user touches on an area of the screen defined by four Cartesian coordinates, i.e., (x1, y1), (x2, y2), (x3, y3), (x4, y4) the module automatically translates it to the corresponding key, say, 'a'. However, in this case, it is practically impossible to pre-define the position of each key and then associate it with its name (character) because their position changes randomly every time the user touches on a key. So, this time, the module contains the Cartesian coordinates of all virtual keys presented by the keyboard (when the zoom level is set to zero and the page is aligned to the center of the device's display) but it assigns to all of them the null value.

First, iKL detects if the touch event corresponds to a key location. If true, the module instantly captures a screenshot of the area defined by the four Cartesian coordinates of that key and saves this tiny image using a name in the form {*device_unique_ID, serial_number, time_in_msecs*}. Again, every time the user performs a zoom in or out on the webpage or tries to relocate the position of the website view, the module recalculates on-the-fly the actual positions of all virtual keys based on the new zoom level and the orientation of the mobile device.

Taking into account the above discussion, and setting aside user discontent, the only way to temporarily evade such a touchlogger is having the virtual keyboard to constantly change the size of the virtual keys as well. But even in this case, the malware can capture a screenshot of all the screen area after placing a small transparent dot on the point the user touched the screen. Of course, this can lead to several images that occupy a considerable amount of memory in the device's permanent storage space which in turn may eventually expose the malware.

5. Related work

Over the last few years, face recognition, fingerprints and iris biometric data have been employed as an authentication method for improving the security of smartphones. One of the most prominent and at the same time cheap and efficient biometric method used in the security field for user (post) authentication and verification is keystroke dynamics. According to (Ahmed et al., 2008), keystroke dynamics is considered as a strong behavioral biometric-based authentication method, where the system monitors the keyboard as the user types in an effort to identify the authenticity of the user based on habitual patterns. Actually, it comes as no surprise that the way an individual interacts with a device is very specific to this person. The pioneering work by Bryan and Harter showed that telegraph operators had very individual patterns of entering messages over telegraph lines (Leggett et al., 1991).

Biometric keystroke authentication systems have been classified, according to the *Verification* or *Identification* mode, where the system each time is set to one of these modes, and pattern classification techniques are employed to analyze the keystrokes (Bergadano et al., 2002). According to the Verification mode, the system attempts to validate user's identity profile by comparing the user's input sample against a previous reference template. When using the Identification mode, the system attempts to match a user input pattern against a collection of a priori known profiles so as to identify the legitimate user. Keystroke authentication can be also classified as either static, where the classification analysis is conducted only at specific time (e.g., login process), or continuous, where the classification analysis is performed repeatedly after a successful login (post-authentication). Normally, keystroke systems try to classify a pattern using either machine learning techniques, which have been successfully employed in anomaly IDS, or common statistical methods.

So far, several research works have been devoted to keystroke analysis for mobile devices. In this section we categorize them at a high level into keystroke proposals that have been conducted for mobile devices equipped with a hardware keyboard, and others that directly or indirectly consider virtual keyboards used by modern smartphones. Also note that this section concentrates on keystroke authentication systems. Thus, approaches that deal with keylogging as a way to generate better models of user-to-mobile-device interaction in the context of Human-to-Computer Interaction (HCI) like (Schulz, 2008; Park et al., 2008; Lee and Zhai, 2009) and others that examine keystroke on desktop (fixed) keyboards or computer mouse like those in (Nakkabi et al., 2010; Findlater et al., 2011; Feher et al., 2012; Stefan et al., 2012) have been intentionally neglected. The same applies for works capitalizing on side (or covert) channels, such as electromagnetic and optical emanations, in an effort to leak out information about which key has been pressed on a desktop keyboard (Vuagnoux and Pasini, 2009; Adhikary et al., 2012). All the approaches presented below are arranged in chronological and thematic order in Fig. 5.

5.1. *Hard keyboard-oriented keystroke proposals*

Clarke et al. (Clarke et al., 2002) examine a number of classification algorithms based on feed-forward multiple layer perception neural networks in an effort to evaluate the potential to authenticate users by the way they type text messages with a qwerty mobile hardware keyboard. Their results have been promising, with an average classification of 18% EER and individual users achieving an EER as low as 3.2%. A follow up work by the same authors (Clarke and Furnell, 2007) also evaluates the potential to authenticate users by the way they type text messages succeeding an average EER of 12.8%. A year later, Karatzouni and Clarke (2007) identified that the hold-time (which is the interval between the pressing and releasing of a single key) was not a beneficial feature for use on a qwerty mobile device but a combination of both inter key (which is the interval between two successive keystrokes) and hold time measures would provide better results.

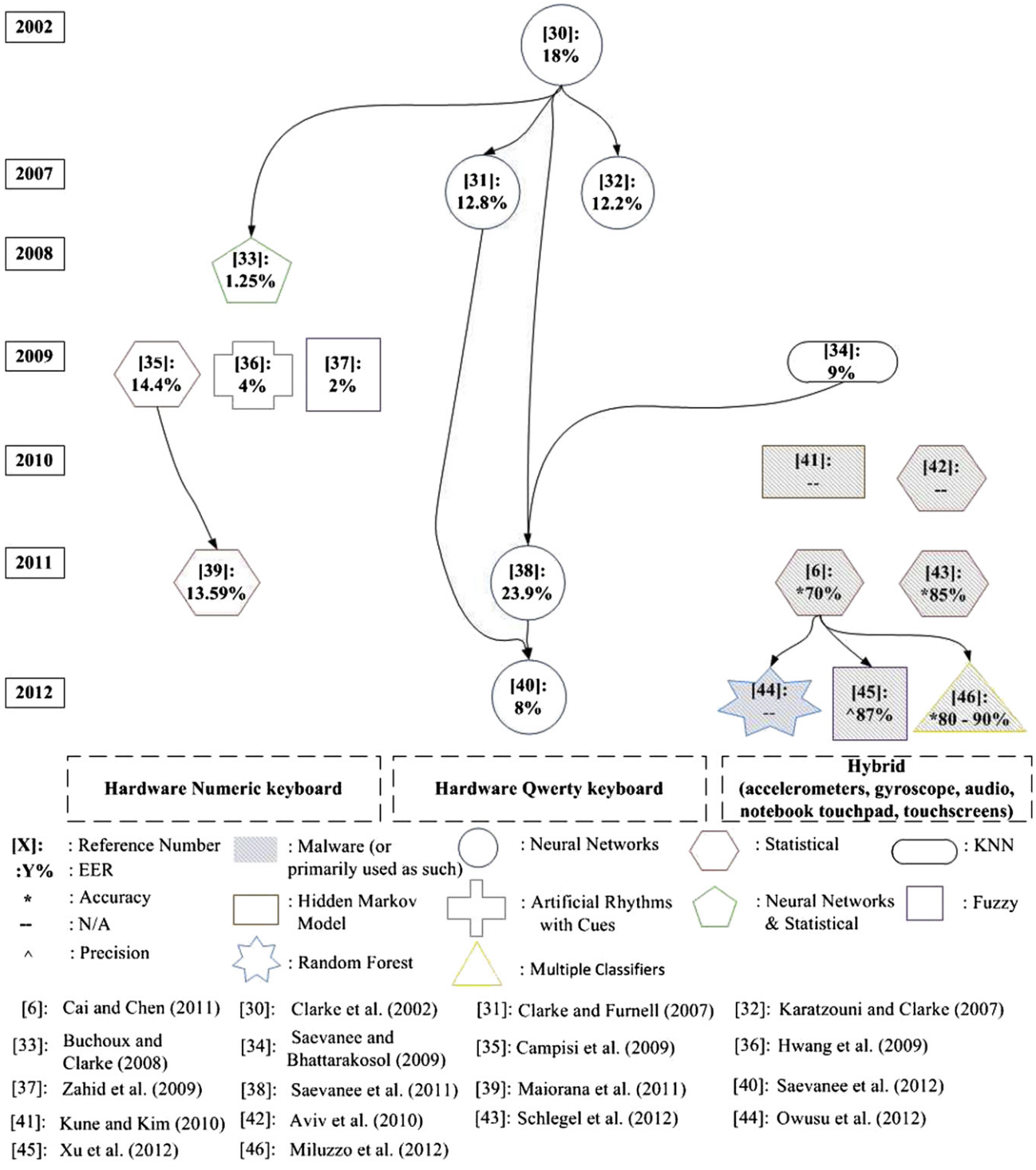

Fig. 5 – Major mobile device keystroke approaches in chronological order. The arrows indicate interrelation between proposals ((i.e., [b] has been influenced by [a]).

Buchoux and Clarke (2008) capitalized on keystroke analysis on a qwerty smartphone and highlighted that the large amount of processing power required to effectively use a neural network as classifier exceeded that of the device.

A year later, Saevanee and Bhattarakosol (2009) suggested new metrics such as finger pressure and unique combinations of keystroke latencies to authenticate modile users. Their study conducted on a sample of 10 participants with a notebook touchpad had the lowest EER of 9% using keystroke dynamics and the KNN classification algorithm. During the same year, Campisi et al. (2009) focused on keystroke biometrics within the framework of secure user authentication using a numeric mobile hardware keyboard. They proposed a statistical approach able to produce

satisfactory verification rates (of 14.46% EER) even in cases where the number of samples contributed by the participants is low. The authors worked with data taken from a sample of 40 users who have typed each password 20 times during 4 distinct sessions. Hwang et al. (2009) proposed a Keystroke Dynamics-based Authentication (KDA) method with "artificial rhythms" and "tempo cues" for mobile user authentication. Their aim was to come through problems resulting from short PIN length. They experimented with standard keypad mobile devices and found that the proposed strategy reduces ERR from 13% to 4%. Zahid et al. (2009) proposed a user identification system that takes into account 6 distinct keystroke features and can be used for user identification with average ERR of 2%. They showed that specific keystroke features for different users are diffused and therefore a fuzzy classifier is well-suited for clustering and classification of those data.

More recently, Saevanee et al. (2011) elaborated on behavioral biometric techniques that can be applied toward authenticating users that utilize mobile devices with a qwerty hardware keyboard. To this end, they study the biometric information, the keystroke dynamics and the finger pressure (the last one using a notebook touchpad) using the RBF algorithm to sift out legitimate users from intruders. The average result they succeeded in terms of FAR and FRR was 52.2% and 1% respectively. These results have been acquired using keystroke dynamics (hold-time) as the key factor to identify users. Moreover, they succeeded in obtaining a FAR of 44.8% and an FFR of 3% by using finger pressure as another key factor to identify users. During the same year, Maiorana et al. (2011) introduced a new statistical classifier for keystroke recognition. They analyzed the verification performances achievable when varying several parameters like the distance between keypress and key release, as well as the number of enrollment acquisitions, and the number of characters contained in the used passwords.

The most recent work is that of (Saevanee et al., 2012). The authors correctly observe that an effective way to augment the reliability of non-intrusive and continuous authentication systems is create a multi-modal behavioral biometric authentication system for mobile devices. Toward this goal they investigate the potential of fusing three different biometric methods, namely behavior profiling, keystroke dynamics, and linguistic profiling, into a multi-modal behavior biometric authentication system. The results they succeeded indicate that such fusion techniques can improve the classification performance with an overall EER of 8%.

5.2. *Soft keyboard-oriented keystroke proposals*

Kune and Kim (2010) attempted to reduce the passcode combination search space by overhearing the acoustic feedback emitted from PIN touch-input devices when a user enter their password. They managed to record audio feedback samples produced by 3 distinct types of keypads; an iPhone device's initial screen passcode lock, an office door keypad, and an ATM. However, their work is in a preliminary stage suggesting that by using a Hidden Markov Model it might be possible to considerably narrow the PIN search space.

Aviv et al. (2010) examine the feasibility of "smudge attacks" on touchscreens for smartphones. They argue that oily residues (smudges) on the touchscreen surface are one side effect of touches from which frequently used patterns such as a graphical password might be inferred. They focus on Android password patterns and investigate the conditions under which a smudge can be easily extracted. The authors also describe how an ill-motivated person could use the information obtained from a smudge attack to augment the chances of guessing users' patterns.

The most relevant work to ours is that of Cai and Chen (2011) appeared in 2011. They proposed a new keylogging scheme based on mobile device motion. They argue that typing (touching) on different locations on the screen causes different vibrations (motion data) which in turn can be used to infer the keys being typed. Their evaluation shows that the proposed system can correctly infer more than 70% of the keystrokes on a number-only virtual keypad when used in the landscape mode.

During the same year, Schlegel et al. (2011) reported their findings on sensory malware. This is an emerging type of smartphone malware that uses on-board sensors to harvest private user information. The authors created Soundcomber, a trojan that is able to snatch a small amount of targeted private information from the audio sensor of the smartphone. According to their experiments, the extraction of sensitive private data such as credit card and PIN numbers from both tone- and speech-based interaction with phone menu systems is possible with an accuracy equal to 85%.

Inspired by the work in (Cai and Chen, 2011), the authors of (Owusu et al., 2012; Xu et al.,; Miluzzo et al., 2012) discuss and evaluate their proposals designed with the aim to extract sequences of entered text on touchscreen keyboards. This is done by taking advantage of only the on-device motion sensors, i.e., the accelerometer and gyroscope. More specifically, Owusu et al. (2012) showed that accelerometer can be used to extract 6-character passwords in as few as 4.5 trials (median). Xu et al., presented TapLogger a stealth trojan for the Android platform which is able to log not only the screen lock password but also the numbers entered during a phone call. Actually, TapLogger implements two schemes: (a) a tap event detection mechanism to discover and utilize the users pattern with statistical measurements on acceleration, and (b) an approach of deducing tap position with observed gesture changes. For the first scheme they report an average precision of 83%, after experimenting with 2 different smartphone models and 3 participants. The last work by Miluzzo et al. (2012) introduces TapPrints, a framework to infer where one taps and what one types on the touchscreen based on accelerometer and gyroscope sensor readings. In their experiments engaging 10 participants and three different mobile platforms the authors show that TapPrints is able to attain up to 90% and 80% accuracy in inferring tap locations across the display and letters respectively.

5.3. *Discussion*

From the above analysis, it is clear that there is significant research interest in this area. Naturally, the main inspiration factor for all these works is found in touchstroke biometric

schemes proposed over the years for both fixed and mobile devices. The research efforts in the field seem to now expand toward capitalizing on the advanced functionalities and features that ultramodern mobile devices bring along (including full qwerty keyboards, cameras, touchscreens, accelerometers etc). In this context, our work is more akin to those included in Section 5.2. Note however, that the common ground of all these efforts is the observation that keyloggers are facing major obstacles on touchscreen devices due to the non-electromagnetic emanation from a virtual keyboard, the OS restricted privileges granted to applications and the difficulty to read keystrokes unless the keylogger software is active and receives the focus on the screen.

This is exactly where the essence of the contribution of the current paper lies. Specifically, in contrast to what is proposed in (Cai and Chen, 2011; Owusu et al., 2012; Xu et al.,; Miluzzo et al., 2012), the novelty of our approach is that the logging of touch events is actual. This means that, every touch event or even gesture happening on the touchscreen is recorded and not indirectly inferred by (or relies on) other factors. Obviously, this advancement – pertaining to low-level OS functionality - enables us to record any touch event occurring anywhere on the screen and not just those taking place on the virtual keyboard. Therefore, to our knowledge, this is the first work that attempts to profile the user based on touchscreen behavioral patterns. Our feeling is that this research field will evolve in the years to come because, as already mentioned, mobile devices embed a unique number of features that can be straightforwardly exploited for implementing such solutions.

6. Conclusions and future work

With the advent of smartphones equipped with touchscreen it is certain that we shall witness the emergence of sophisticated software that can be used either benignly or maliciously. Without doubt, the mutation of keyloggers to touchloggers is a salient paradigm of what “the road ahead” is. The significance of the current study lies in the assessment of the potentiality of such a type of software under two different views into the same prism. Specifically, to the best of our knowledge, our work is the first to demonstrate that this type of software can be used to profile and subsequently post-authenticate the user of the device with extremely high accuracy in the vicinity of 100%. The malevolent personality of such a powerful and stealthy software is also exhibited through practical case studies.

We are currently working on extending this research by gathering more user data and analyzing them more thoroughly, e.g., by enriching the existing dataset and employing advanced machine learning techniques, as sketched in Section 3.2.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and suggestions, which have significantly contributed to improve the quality of this paper.

REFERENCES

Adhikary N, Shrivastava R, Kumarl A, Verma SK, Bag M, Singh V. Battering keyloggers and screen recording software by fabricating passwords. International Journal of Computer Network and Information Security – (IJCNIS) 2012;4(5):13–21.

Ahmed AAE, Traore I, Almulhem A. Digital fingerprinting based on keystroke dynamics. In: Proceedings of the Second International Symposium on Human Aspects of Information Security & Assurance – HAISA; 2008. p. 94–104.

Apple Inc.. UIKit framework reference, http://developer.apple.com/library/ios/#documentation/uikit/reference/UIKit_Framework/_index.html; 2012.

Aviv A, Gibson K, Mossop E, Blaze M, Smith J. Smudge attacks on smartphone touch screens. In: Proceedings of the 4th USENIX Workshop on Offensive Technologies – WOOT; 2010.

Bergadano F, Gunetti D, Picardi C. User authentication through keystroke dynamics. ACM Transactions on Information and System Security 2002;5(4):367–97.

Buchoux A, Clarke NL. Deployment of keystroke analysis on a smartphone. In: Proceedings of the 6th Australian Information Security Management Conference – SECAU, Western Australia; 2008. p. 40–7.

Cai L, Chen H. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In: Proceedings of the 6th USENIX Workshop on Hot Topics in Security – HotSec; 2011.

Campisi P, Maiorana E, Bosco ML, Neri A. User authentication using keystroke dynamics for cellular phones. IET Signal Processing – Special Issue on Biometric Recognition 2009;3(4):333–41.

Clarke NL, Furnell SM. Authenticating mobile phone users using keystroke analysis. International Journal of Information Security 2007;6(1):1–14. Springer.

Clarke NL, Furnell SM, Lines B, Reynolds P. Subscriber authentication for mobile phones using keystroke dynamics. In: Proceedings of the 3rd International Network Conference – INC, UK; 2002. p. 347–55.

Damopoulos D, Kambourakis G, Gritzalis S. iSAM: an iphone stealth airborne malware. In: Proceedings of the 26th IFIP TC-11 International Information Security Conference, IFIP Advances in Information and Communication Technology – IFIP AICT. Springer; 2011. p. 17–28.

Damopoulos D, Kambourakis G, Gritzalis S, http://ibackup.samos.aegean.gr/iTL/; 2012.

EFG Eurobank, http://www.eurobank.gr/online/home/index.aspx?lang=en; 2012.

Feher C, Elovici Y, Moskovitch R, Rokach L, Schclar A. User identity verification via mouse dynamics. Information Sciences 2012;201(0):19–36. Elsevier.

Findlater L, Wobbrock JO, Wigdor D. Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces. In: Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems – CHI. New York: ACM; 2011. p. 2453–62.

Hwang S, Cho S, Park S. Keystroke dynamics-based authentication for mobile devices. Computers & Security 2009;28(1–2):85–93. Elsevier.

iDW, iPhone Development Wiki, http://iphonedevwiki.net/index.php/Special:AllPages; 2012.

iKeyGuard, http://ikeyguard.com/; 2012.

Karatzouni S, Clarke NL. Keystroke analysis for thumb-based keyboards on mobile devices. In: New approaches for Security, Privacy and Trust in Complex Environments. IFIP International Federation for Information Processing, Springer Boston; 2007. p. 253–63.

Kune FD, Kim Y. Timing attacks on pin input devices. In: Proceedings of the 17th ACM Conference on Computer and

Communications Security – CCS. New York: ACM; 2010. p. 678–80.
Lee S, Zhai S. The performance of touch screen soft buttons. In: Proceedings of the 27th International Conference on Human Factors in Computing Systems – CHI. New York: ACM; 2009. p. 309–18.
Leggett J, Williams G, Usnick M, Longnecker M. Dynamic identity verification via keystroke characteristics. International Journal of Man-machine Studies 1991;35(6):859–70. Academic Press.
Maiorana E, Campisi P, Gonzlez-Carballo N, Neri A. Keystroke dynamics authentication for mobile phones. In: Proceedings of the 2011 ACM Symposium on Applied Computing – SAC. USA: ACM; 2011. p. 21–6.
Miluzzo E, Varshavsky A, Balakrishnan S, Choudhury RR. Tapprints: your finger taps have fingerprints. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, And Services – MobiSys'12. ACM; 2012. p. 323–36.
Nakkabi Y, Traore I, Ahmed A. Improving mouse dynamics biometric performance using variance reduction via extractors with separate features. IEEE Transactions on Man and Cybernetics, Part A: Systems and Humans 2010;40(6): 1345–53. IEEE.
Owusu E, Han J, Das S, Perrig A, Zhang J. Accessory: password inference using accelerometers on smartphones. In: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications – HotMobile'12. ACM; 2012.
PanCaribbean Bank, Pancaribbean Bank, https://online.gopancaribbean.com/retail/RetailLoginLang.html; 2012.
Park YS, Han SH, Park J, Cho Y. Touch key design for target selection on a mobile phone. In: Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services – MobileHCI. New York: ACM; 2008. p. 423–6.
Polla ML, Martinelli F, Sqandurra D. A survey on security for mobile devices. Communications Surveys & Tutorials 2012; PP(99):1–26. IEEE Press.
Saevanee H, Bhattarakosol P. Authenticating user using keystroke dynamics and finger pressure. In: Proceedings of the 6th IEEE Consumer Communications and Networking Conference. Ireland: IEEE; 2009. p. 1–2.
Saevanee H, Clarke NL, Furnell SM. Behavioural biometric authentication for mobile devices. In: Proceedings of the 2011 Collaborative European Research Conference – CERC, Ireland; 2011. p. 175–84.
Saevanee H, Clarke NL, Furnell SM. Multi-modal behavioural biometric authentication for mobile devices. In: Proceedings of the Information Security and Privacy Research, IFIP Advances in Information and Communication Technology – IFIP AICT. Springer Boston; 2012. p. 465–74.
Sagiroglu S, Canbek G. Keyloggers. IEEE Technology and Society Magazine 2009;28(3):10–7. IEEE.
Schlegel R, Zhang K, Zhou X, Intwala M, Kapadia A, Wang X. Soundcomber: a stealthy and context-aware sound trojan for smartphones. In: Proceedings of the 18th Annual Network & Distributed System Security Symposium – NDSS'11; 2011.
Schulz T. Using the keystroke-level model to evaluate mobile phones. In: Proceedings of the 31st Information Systems Research Seminaria – IRIS 31, Scandinavia; 2008.
Sipior JC, Ward BT. Trust, privacy, and legal protection in the use of software with surreptitiously installed operations: an empirical evaluation. Information Systems Frontiers 2008; 10(1):3–17. Springer.
Sreenivas RS, Anitha R. Detecting keyloggers based on traffic analysis with periodic behaviour. Network Security 2011; 2011(7):14–9. Elsevier.
Stefan D, Shu X, Yao D. Robustness of keystroke-dynamics based biometrics against synthetic forgeries. Computers & Security 2012;31(1):109–21. Elsevier.
Syndicate Bank, https://netbanking.syndicatebank.in/netbanking/ 2012.
The iPhone Wiki, http://theiphonewiki.com/wiki/index.php?title=/System/Library/Frameworks; 2012.
EFG Eurobank app, http://itunes.apple.com/us/app/eurobankefg/id364587747?mt=8; 2012.
Vuagnoux M, Pasini S. Compromising electromagnetic emanations of wired and wireless keyboards. In: Proceedings of the 18th conference on USENIX security symposium – SSYM'09. USENIX Association; 2009. p. 1–16.
Weka. Weka machine learning project, http://www.cs.waikato.ac.nz/ml/weka; 2012.
Xu Z, Bai K, Zhu S. Taplogger: inferring user inputs on smartphone touchscreens using on-board motion sensors. In: Proceedings of the 5th ACM conference on Security and Privacy in Wireless and Mobile Networks – WISEC'12, ACM, p. 113–124.
Zahid S, Shahzad M, Khayam S, Farooq M. Keystroke-based user identification on smart phones. In: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection – RAID'09. Springer-Verlag; 2009. p. 224–43.
Zaitsev O. Skeleton keys: the purpose and applications of keyloggers. Network Security 2010;2010(10):12–7. Elsevier.

Dimitrios Damopoulos is currently a Ph.D candidate at the Dept. of Information and Communication Systems Engineering, University of the Aegean. He received an MSc in Information & Communication Systems Security from the Dept. of Information and Communication Systems Engineering, University of the Aegean, Greece. He also holds a B.Sc in Industrial Informatics from the Technological Educational Institute (TEI) of Kavala, Greece. His research interest includes Mobile Security, Mobile Device Intrusion Detection, Malware Detection.

Georgios Kambourakis, received the Diploma in Applied Informatics from the Athens University of Economics and Business and the Ph.D. in information and communication systems engineering from the department of Information and Communications Systems Engineering of the University of Aegean. Currently Dr. Kambourakis is an Assistant Professor at the Department of Information and Communication Systems Engineering of the University of the Aegean, Greece. His research interests are in the fields of Mobile and ad-hoc networks security, VoIP security, security protocols, DNS security, Public Key Infrastructure and mLearning and he has more than 85 publications in the above areas.

Stefanos Gritzalis is a Professor at the Department of Information and Communication Systems Engineering, University of the Aegean, Greece and the Director of the Laboratory of Information and Communication Systems Security (Info-Sec-Lab). He holds a BSc in Physics, an MSc in Electronic Automation, and a PhD in Information and Communications Security from the Dept. of Informatics and Telecommunications, University of Athens, Greece. His published scientific work includes 30 books or book chapters, 100 journals and 130 international refereed conference and workshop papers. The focus of these publications is on Information and Communications Security and Privacy.