Curbing Mobile Malware based on User-Transparent Hand Movements

Babins Shrestha*, Manar Mohamed*, Anders Borg*, Nitesh Saxena* and Sandeep Tamrakar†
*University of Alabama at Birmingham, USA, Email: {babins,manar,andy91,saxena}@uab.edu
†Aalto University, Finland, Email: sandeep.tamrakar@aalto.fi

***Abstract*—In this paper, we present a run-time defense to the malware that inspects the presence/absence of certain *transparent* human gestures exhibited naturally by users prior to accessing a desired resource. Specifically, we focus on the use of transparent gestures to prevent the misuse of *three critical smartphone capabilities* – the phone calling service, the camera resource and the NFC reading functionality. We show how the underlying natural hand movement gestures associated with the three services, *calling*, *snapping* and *tapping*, can be detected in a robust manner using *multiple* – motion, position and ambient – sensors and machine learning classifiers. To demonstrate the effectiveness of our approach, we collect data from multiple phone models and multiple users in real-life or near real-life scenarios emulating both *benign* settings as well as *adversarial* scenarios. Our results show that the three gestures can be detected with a high overall accuracy, and can be distinguished from one another and from other activities (benign or malicious), serving as a viable malware defense. In the future, we believe that transparent gestures associated with other smartphone services, such as sending SMS or email, can also be integrated with our system.**

I. Introduction

Mobile devices, especially smartphones, are rapidly becoming ubiquitous. These devices open up immense opportunities for everyday users offering valuable resources and services. In addition to traditional capabilities, such as voice calling, SMS and web browsing, most smartphones today have high-resolution cameras and many of them come equipped with the NFC (Near Field Communication) functionality. An NFC phone can be used as an RFID contactless payment token, such as a credit card. It can also be used as an RFID reader that can "read" other RFID cards or NFC phones in close proximity. All these features on mobile devices have not only attracted millions of consumers but have also motivated the developers to build wide range of apps.

At the same time, these devices are also becoming an easy target for malicious activities. At present, mobile malware has become a burgeoning threat to mobile devices and their users [7, 8, 10, 13, 17, 20]. One of the primary reasons for such malware explosion is user's ignorance; users often download applications from untrusted sources which may host apps with hidden malicious codes. Once installed on a smartphone, such malware can exploit the smartphone in various ways. For example, it can access the smartphone's resources to learn various sensitive information about the user, secretly use the camera to spy on the user or make premium rate phone calls without user's knowledge, or use NFC reader to skim for physical credit cards within its vicinity. Indeed, many practical instances of such malware have been reported either in the wild or by academicians and practitioners. For instance, PlaceRaider [24] is a visual malware that can take pictures of a user's surroundings, building a 3-D model of the user space and gleaning sensitive information. An NFC pickpocketing Trojan program has also been developed [1] that can sniff nearby credit cards.

Unfortunately, current operating systems (e.g., Android and iOS) provide inadequate security against these malware attacks. For granting permission to an application requiring access to the resources, these operating systems either require out-of-context, uninformed decisions at the time of installation via manifest [2, 19] or prompt users to determine their interest via system prompt [14, 19]. This approach relies upon user diligence and awareness. It is well-known that most users do not pay attention to such "Yes/No" prompts and frequently just select "Yes" so as to proceed with the installation. Once granted the permission, applications have full authority over the resources and can access them without the owner's consent. In addition to relying upon user permission, applications are also reviewed. However, review process has failed in the past [11, 26] and users gaining the root permission/ jail-breaking the phone can easily install the third party applications which may not have been reviewed.

In this paper, we set out to defend against mobile malware that can exploit critical and sensitive mobile device services, especially focusing on the *phone's calling service, camera and NFC*. In order to remain stealthy, mobile malware attacks occur in scenarios where the device user has no intention to access the underlying services. Thus, if the user's intent to access the services can be captured in some way, these attacks could be prevented. We propose to elicit a user's intent via gestures that are transparently and naturally performed by the user prior to accessing the services. In other words, whenever the user wants to access the service, she will naturally exhibit a particular gesture. On the other hand, if the malware attempts to access the service, the gesture will be missing and the access request can be blocked. We focus on *authorizing an app* with the use of transparent human gestures, *not on authenticating users* (which is an independent problem).

This general idea of human gesture-centric malware prevention was first introduced in our own recent work [12, 23]. However, this line of work only considered natural gestures to protect the NFC resource (such as tapping) or required the users to perform certain "explicit" gestures (such as hand waving in front of the phone). The latter form of gestures impose additional burden on the user every time access is needed thereby undermining system's usability, and may eventually

degenerate into nothing more than a "Yes/No" prompt if used frequently thereby reducing security. In this paper, we extend the scope of transparent gestures to protect the phone calling service and the camera resource, and propose a novel and more robust gesture detection mechanism based on multiple sensors to protect the NFC reading service.

The **contributions** of this paper are three-fold:

1. *Malware Prevention Based on Transparent Gestures:* We propose the use of transparent gestures to prevent the misuse of three critical smartphone capabilities – the phone calling service, the camera resource and the NFC reading functionality. We show how the underlying hand movement gestures associated with the three services can be detected in a robust manner using *multiple* – motion, position and ambient – sensors and *machine learning classifiers*. In the future, we believe that transparent gestures associated with other smartphone services, such as sending SMS or email, can also be integrated within our system.

2. *Gesture Detection Design and Implementation:* We design and implement a suite of three novel transparent gesture detection mechanisms as part of the proposed malware defense on the Android platform. The *calling* gesture for an incoming or outgoing call involves holding the phone then moving it close to the ear. The *snapping* gesture involves holding the phone in hand, and raising and holding it to snap a picture or video. The *tapping* gesture involves holding the phone in hand, and tapping and holding onto a NFC/RFID tag.

3. *Experiments and Evaluation:* To demonstrate the effectiveness of our approach, we have collected data from multiple phone models and multiple users in real-life or near real-life scenarios emulating both *benign* settings as well as *adversarial* settings, and report on the overall accuracy of our gesture detection approaches. Our results show that the calling, snapping and tapping gestures can be detected with high overall accuracy, and can be distinguished from one another and from other activities.

II. Background

A. Threat Model

In our security model, similar to [12, 23], we assume that a user (Alice) has already installed a malicious application (malware) on her device. She is unaware of the presence of this malware on the device as it can hide inside a benign looking application such as a game. This can happen, for example, when the user downloads an app from an untrustworthy source. Malware can also spread to user device via various paths/communication channels without any user suspicion. Preventing malware from being installed on the phone is orthogonal to the scope of our model.

We also assume that the OS kernel is healthy and is immune to the malware infection [12, 23]. Strengthening the kernel is again an orthogonal problem [18, 21]. To be specific, malware is not able to maliciously alter the kernel control flow. We also assume that hardware is immune from the malware and the malware cannot manipulate the device's on-board sensors: A malware capable of manipulating the sensors can also produce appropriate gesture required to access the sensitive resources.

The eventual goal of the malware is to access the device's sensitive resources/services to make premium phone calls, take pictures of user's surroundings [24] or read nearby NFC enabled credit cards/tags [1]. It may continuously try to access these services or at random times to remain stealthy. We do not enforce any restrictions on how frequently the malware attempts to access a given service.

Finally, we assume that the attacker does not physically possess the phone. If the attacker has physical access to the phone, then he can simply grant permission to malware by providing the necessary gesture. In other words, our mechanism is *not intended for the purpose of user authentication*, but designed to *prevent malicious applications from accessing phone's resources* .

B. Design Goals and Metrics

For our scheme to be effective and practical, we need to consider following design goals (adopted from [12, 23]):

1. The scheme should be *lightweight* in terms of memory, computation and power consumption.

2. The approach should be *efficient*, i.e., not incur a perceptible delay. If a user has to wait for a perceptibly long time while using the scheme, this will affect the overall usability.

3. The scheme should be *robust* to errors. That is, the app must be granted permission with high probability when a user is actually trying to access the services/resources (usability), while the malicious apps must be denied the permission to access them (security). A detailed set of metrics to evaluate the robustness of our approach is provided in Section V.

4. The approach should be *transparent* to the users. The users should not be required to perform additional tasks or gestures (such as explicit gestures [12, 23]) because such actions may degrade usability and security, and reduce chances of adoption.

III. Our Approach: Curbing Malware Using Transparent Gestures

1) Approach Overview: The Android OS, one of the most popular smartphone operating systems, provides APIs to support different categories of sensors such as those that measure motion, position and environment. Motion sensors such as accelerometer and gyroscope provide information regarding movement of the phone. Position sensors, such as magnetometer and orientation sensors, provide information about the orientation and position of the phone with respect to the world's frame. Environmental sensors, such as temperature, pressure and illuminance, monitor ambient physical properties around the phone. In our malware defense mechanism, we leverage these sensors, especially motion sensors, to identify if the phone has been moved in a way that the user is trying to perform certain activities. We show that such hand movements can be used to protect three critical mobile device resources – voice dialing, camera and NFC – against malicious use. These gestures are transparent to the users as they are performed implicitly as part of the activity itself – no additional involvement or explicit gestures from the user is needed.

Now let's consider the case of dialing or receiving a phone call. As part of making/answering a call, a user either presses/swipes a "Call" button which can be a "Dial" button or an "Answer" button, or an icon, and then brings the phone close to the ear. This involves a particular motion of the phone

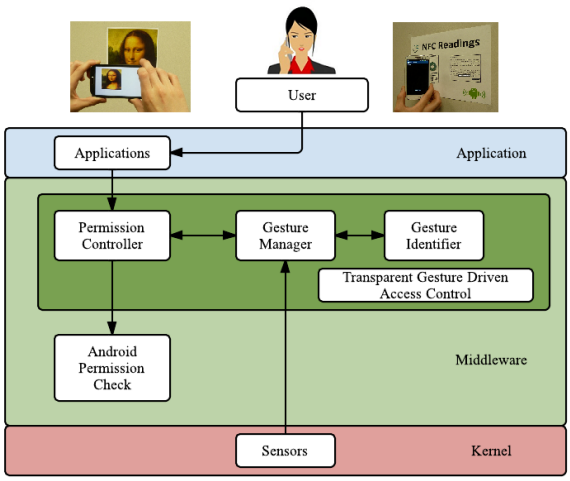

Fig. 1. System Architecture (our current system supports calling, snapping and tapping gestures, but can also incorporate other transparent gestures in the future).

which can be measured using the motion sensors. In addition, the phone may be subject to a certain hand pressure which can be detected using the pressure sensor. Similarly, when a user tries to take a picture/video with a camera on her device, she first opens a camera app then orients her device to compose the picture in the viewfinder and finally captures the snapshot. This process involves a particular hand motion which can be detected using the different sensors. In the same vein, before "reading" an NFC tag or another NFC device, the user has to move the phone towards the NFC tag/device, tap on it, hold until the information is read, and then bring the phone back. Similar to phone call and camera access, the NFC reading gesture can be detected using various sensors.

In this paper, we show that these three particular gestures can be detected in a robust manner using machine learning classifiers based on data drawn from multiple phones' sensors. We will demonstrate in the following sections that our approach meets all of our design goals such as being lightweight, efficient, robust and user-friendly.

2) System Model: The idea of our approach is to add another layer of permission control on top of the original Android permission granting system. When an application requests a resource access, such as to make phone call or use camera hardware or use NFC, android checks the resource access permission associated with the app before allowing any access to the resource. If the user has provided the app with the permission it needs during the installation time, the app is allowed to access these resources. Our approach adds another layer of security on top of this mechanism to prevent malicious application from using the sensitive resources. The gesture detection mechanism should follow a trusted path to the OS, as stated in Section II-A, assuming the kernel control flow is not compromised.

To identify if the corresponding gesture has been performed by the user while accessing a resource, such as making/answering a call, taking a picture with the camera or tapping an NFC tag, our system first waits for the user to trigger the event. When such an event occurs, the Android OS throws the corresponding intent. Next, the intent is intercepted and checked for the permission. The permission token is granted only when the underlying gesture is detected. In this model, we need three entities: (1) a *Gesture Identifier* which is a trained classifier that can identify a gesture; (2) a *Permission Controller* that checks for the permission token; and (3) a *Gesture Manager* that communicates with the hardware, gets the phone movement data from the sensors and provides the analysis results to the Permission Controller.

The Permission Controller stands in front of the default Android permission granting system. When an app request to access a sensitive resource, the Permission Controller will intercept the request. The Permission Controller then checks the permission token associated with the request. It forwards the request to the Android permission system only when the request consists of the permission token. Otherwise it interacts with the Gesture Manager to check whether a certain gesture protects the service or not. If it requires, then the Gesture Manager interacts with the Gesture Identifier that begins collecting data from different sensors and recognizes a valid gesture for the corresponding request. If the gesture is identified as a valid gesture, it attaches a permission token to the request. Our permission model builds on and extends the model of [12].

IV. Data Collection

A. App Design

To develop and evaluate our gesture-centric malware defense mechanism, we first needed to collect data from users to recognize the various gestures exhibited by them in varying scenarios. To this end, we created a suite of four separate apps , each of which collects the data from users while: (1) making/answering phone calls (the *Call App*), (2) taking pictures using the phone's camera (the *Snap App*), (3) reading NFC tags via tapping (the *Tap App*), and (4) various random movements and activities captured at random times or in controlled settings (the *Snoop and Control App*). The app supports Android OS 4.0 (API Level 14) or later. The data collected from this app system is used to recognize the Call, Snap and Tap gestures, and to determine the possibility of these gestures matching with one another and with other controlled activities.

To prevent the overuse of sensors and excessive drainage of the battery, the periods of recording data must be limited to the events identified as the capture periods. Android provides/throws an intent to all listening applications whenever the user intends to perform certain activities. Hence, we configured the app to trigger the sensor readings whenever a corresponding event occurs. Once the event is received, the data from sensors are recorded as shown in Figure 2. The specifics of our apps are discussed below.

Call App: The phone calling intent is triggered when the call is initiated/answered and the state becomes "OFFHOOK". With this intent, the app starts recording the sensor data. This means that the user has made/answered the phone and the associated Call gesture, i.e., the motion to bring the phone to the ear, has been initiated. The app stops reading sensor data as soon as the proximity sensor value changes indicating the phone has reached the ear of the user. To preserve the integrity of the data, calls made using a headset or in the speaker phone are not considered since the motion expected for answering the phone is not likely to be exhibited during these events. In such cases, we need a fallback mechanism (see Section VI).

Fig. 2. Data Collector Flowchart (horizontally laid out to optimize space)

Snap App: In the Android system, the camera hardware can be used by any application that is registered with the permission to use it in its manifest file, i.e., there is no system-wide intent that can be intercepted by another application any time the camera is in use. Because of this restriction, we developed a custom application which is to be used as an alternative to other camera applications to test our hand movement gestures. The camera is capable of capturing the intent to start the camera "MediaStore.ACTION_IMAGE_CAPTURE". Once the camera is started, the sensor data is recorded until the user either exits the screen or takes a picture. A flag is set while the camera is on, that is changed once either the application exits the screen or the user takes a picture, at which point the sensors are stopped.

Tap App: For reading NFC, our app waits for the NFC intent and starts recording the sensor data as soon as this intent is captured. Since the gesture/motion between the first time a device detects NFC tag and stops motion is very short, we captured the data for four seconds after it detects NFC tag. In our scheme, the gesture detection may be too late to be captured while the transaction may have already taken place through NFC. For this, we can place the received NFC data in quarantine or stall the NFC command received until the gesture is fully detected and analyzed if it is a valid gesture.

Snoop and Control App: Along with different gestures, the app system also records data from various sensors at random point of times in order to compare these gestures with other activities. We refer to this data as "Snoop". To verify that our classifiers are indeed robust, we need to test them with the active motion data (besides the random Snoop events) which is different from the Call, Snap and Tap gestures, but still may have a chance to match with these gestures. For this purpose, we added extra features in our app system to collect controlled data (referred to as *Control* gestures). It provides a text box for the tester to specify the activity being performed and once the button is pressed for recording, it records all the sensor data for ten seconds.

The app displays the count of how many times each gesture has been performed and provides a button to upload the recorded gesture data to our server. The app is designed to upload the data at regular intervals (24 hours). A user can also explicitly upload the data by pressing the upload button.

B. Data Collection Procedures

We used our data collection app system to extensively collect data for Call, Tap and Snap gestures as well as various Control activities. We distributed the apps to volunteers who were willing to provide the sensor data collected from their devices through their normal activities. We distributed the app to the users in our respective Universities in US and Finland. Before distributing the app to these users, we explained what sensor data was being recorded, for how long it was recorded, at what occasions it was recorded and for what purpose the data was being used. Those who consented to our explanation were provided with apk files for the installation. There were a total of **23 users** recruited for our study. They were students of the Computer Science departments of the two Universities. Due to the real-world nature of our data collection, it was challenging to recruit users for our study, and, as such, our sample size is slightly lower than a typical lab-study, but still sufficient to demonstrate the promising feasibility of our approach. The devices used by the participating students were popular Android smartphones which have all the sensors needed for our tasks. Their devices had Android OS 4.0 (API 14) or later versions. Not all users could provide the required amount of data corresponding to all three of our apps (30 calls, 30 snaps and 30 taps).

Our experiment for the Call gesture was performed in real-world settings, i.e., the data was recorded when the volunteers made or received calls under normal use. We collected the data from each user until we got minimum of 30 samples for each gesture. This took about a month for most users. The Call App is fired whenever the participant made/received phone calls as explained in Section IV-A. Along with recording the sensor data when user made/received a call, it also collected snoop data at regular intervals. The app notifies the user whenever the sensor data is being recorded by displaying an icon with a message in the notification bar.

In a similar manner, we provided the Snap App to the users to collect data in real-world settings. They were asked to take pictures using our Snap App instead of the original camera app conforming to real-world settings. However, following this data collection methodology, we obtained little amount of data during the first phase of data collection. We came up with two conjectures for collecting so little data from such use of the Snap App. First, a user might take many pictures when she is on vacation whereas during normal routine she might not take any pictures for weeks. Second, the app we developed was inferior to the default camera app developed by smartphone manufacturers. Our app did not provide various features (e.g., face recognition, touch focus, zoom in/out or HDR mode), usually provided by the default camera app. Therefore, users might have preferred the default camera app to Snap App. Hence, although the Snap experiment could have been done in real-world settings, we had to ask volunteers to take pictures in a lab setting mimicking real-world scenarios with our app. We posted a photo of "Mona Lisa" in the lab and asked volunteers to take her mugshot. Thus, our Snap experiment is a semi-controlled experiment. We observed that some users preferred to take snaps in landscape mode while other preferred portrait mode.

For the Tap experiment, we provided our Tap App to the volunteers. All of our participants possessed NFC enabled smartphones. However, since the NFC reader was not widely used in real life or by merchants, we had to limit our experiment to the lab settings. We attached a NFC tag in the student lab and asked users to tap on the tag at regular periods. Whenever user tapped the NFC tag, Tap app handled the NFC intent as discussed in Section IV-A. User held back the device

TABLE I. SENSORS UTILIZED FOR GESTURE DETECTION

Type	Sensor	Description
Motion	Accelerometer (A)	The acceleration force including gravity
Motion	Gyroscope (Gy)	The rate of rotation
Motion	Linear Acceleration (LA)	The acceleration force excluding gravity
Motion	Rotation Vector (R)	The orientation of a device
Motion	Gravity (G)	The gravity force on the device
Position	Game Rotation (GR)	Uncalibrated rotation vector
Position	Magnetic Field (M)	The ambient magnetic field
Position	Orientation (O)	The device orientation
Environment	Pressure (P)	The ambient air pressure

as soon as the app displays the toast message "NFC Detected". The App notifies about the data recording in notification bar.

Although we collected snoop data, we needed to make sure our app is robust against different kind of user activities. Since most of the random Snoop data might correspond to the device being stationary with no noticeable change in any sensor data, it might lead to false belief that our gesture classifiers were good enough to classify. Hence, we set out to evaluate the likelihood of false positives under different activities. We conducted several tests to emulate different user activities which might have a chance to match the Call, Snap or Tap gestures. For this experiment, our Control App was set to collect the sensor data for ten seconds as discussed in Section IV-A. We (volunteers) then performed different activities such as walking (upstairs/downstairs), running and jumping. We also mimicked reading phone's screen when there is a notification on device due to email or SMS, i.e., picking up the phone from table/desk to check message. Moreover, we mimicked writing email/SMS, and playing games in different orientations (landscape/portrait). We also performed a "drop test" to see if our classifier provides access to the permission requesting app when phone falls from pocket. For this, we dropped our phones from height of approximately 40 cm onto bed/couch. The activities could be exhibited by smartphone users in their day-to-day life (benign setting). An attacker could also coerce or fool the users into performing these activities with the hope that a false positive would occur allowing malware with access to the resource (adversarial setting).

V. GESTURE DETECTION: DESIGN, AND EVALUATION

In order to detect our Call, Snap and Tap gestures, we used the machine learning approach based on the underlying readings of the motion, position and ambient pressure sensors. We conducted several classification experiments to determine which off-the-shelf machine classifiers and which underlying sensor features provide the optimal performance. We used a total of nine sensors. The sensors used in our analysis and their descriptions are depicted in Table I.

All of the motion and position sensors used have three components corresponding to the three physical axes (X, Y, Z) at each instance. We calculated the scalar value, i.e., the square root of the sum of squares for each instance, which captures the significance of all the three axes. From this scalar value for each instance, we calculated mean and standard deviation for each of the sample for our different gestures, such as Call, Snap, Tap, Control and Snoop. This gave us eighteen features which we used for training and testing various off-the-shelf classifiers using Weka. We developed a Java program that utilizes Weka library to test different classifiers across different sensors subset that would result in best accuracy. We tested different machine learning algorithms provided by Weka: *Trees* – Logistic Model Trees (LMT), Random Forest (RF) and Random Tree (RT); *Functions* – Logistics (L) and Simple Logistic (SL), and *Bayesian Networks* – Naive Bayes (NB), on all sensors subsets.

The eighteen features were used as input to train each classifier to differentiate Call, Snap and Tap gestures from each other and from other gestures collected. We evaluated three training models for the classification task: (1) *user-specific model*, (2) *device-specific model* and (3) *generalized model*. The user-specific model requires each individual user to train a classifier herself before using the app. The device-specific model would require the app developer to build specific classifiers for different phone models. The generalized model uses all the data from all different devices and all users to a build global classifier. These models have their own pros and cons. User-specific model would give better accuracy as it is tailored to an individual user but will require the user to train the classifier before using the system. The other two models do not have a user-centric training phase and will work right after the user installs the app, but the accuracy of this model might be lower than the user-specific model.

In all of the classification tasks, the positive class corresponds to Call/Snap/Tap and the negative class corresponds to other gestures (referred to as "Others"). Therefore, true positive (TP) represents Call/Snap/Tap that is correctly classified as Call/Snap/Tap, true negative (TN) represents Others that is correctly classified as Others, false positive (FP) represents Others misclassified as Call/Snap/Tap and false negative (FN) represents Call/Snap/Tap misclassified as Others.

As performance measures for our classifiers, we used Precision, Recall and F-measure (F1 score), as shown in Equations 1 & 2. Precision measures the security of the proposed system, i.e. the accuracy of the system in detecting the malware. Recall measures the system usability as low recall leads to high rejection rate of legitimate users' actions. To make our system usable, ideally we would like to have recall as close as 1.

$$precision = \frac{TP}{TP + FP}; \quad recall = \frac{TP}{TP + FN}; \tag{1}$$

$$F\text{-}measure = 2 * \frac{precision * recall}{precision + recall} \tag{2}$$

A. Call Detection

For the Call gesture, we could receive the desired data, i.e., the one corresponding to 30 (incoming/outgoing) phone calls, from **14 users**, as discussed in Section IV-B.

User-Specific Model: We divided the data into fourteen sets based on the users' ids. In order to build a classifier to distinguish Call from Other gestures for each set, we define two classes. The first class has the Call data from each user, and the other class contains the data collected from Snap, Tap, Snoop from the same user and the Control data collected from three out of the fourteen users. The average time for the collected Call readings was around one second, so we compared it with one second of every other gesture. To find the best subset of features and classifiers, we applied all the combination of sensors subsets and classifiers (specified above) to each of the fourteen user sets. The best features and the measurement values are calculated from running a 10-fold cross validation as shown in Table II. The gesture detection performance can be termed as quite good. The recall and F-measure of the classifiers were on average 0.95 and 0.92

TABLE II. RESULTS OF USING THE OPTIMAL FEATURE SUBSET IN THE CLASSIFICATION OF CALL AND OTHERS

Classification Model	User ID /Device	Classifier	Features Subset	Precision	Recall	F-Measure
User-Specific	1	L	A,G,LA,P	0.91	0.97	0.94
	2	SL	A,G,LA,O,P	0.97	0.93	0.95
	4	RT	GR,Gy,LA	0.87	0.90	0.89
	5	RT	GR,LA,O,P,R	0.87	0.83	0.85
	6	RF	G,LA,P,R	0.83	1.00	0.91
	7	SL	G,LA,M,P	0.88	0.97	0.92
	9	RF	A,Gy,M,P,R	0.94	0.97	0.95
	11	RT	A,G,Gy,M,P,R	0.85	0.97	0.91
	12	RF	G,Gy,LA,M,P	0.79	0.90	0.84
	13	RF	G,GR,M,P	1.00	0.97	0.98
	14	RF	GR,LA,M,R	0.97	1.00	0.98
	15	RF	LA,M,P,R	0.88	0.93	0.90
	16	L	G,Gy,M,P,R	0.97	1.00	0.98
	20	RF	G,Gy,P	0.88	0.93	0.90
Device-Specific	Google Nexus	RF	G,Gy,LA,R	0.90	0.83	0.86
	Samsung Galaxy	RF	A,GR,G,Gy, P,R	0.83	0.88	0.86
	HTC	RF	GR,LA,M,R	0.97	1.00	0.98
Generalized	ALL	RF	A,GR,LA,M,R	0.92	0.83	0.87

respectively. While the user is performing a Call gesture, he moves the phone which can be measured by the change in the acceleration and air pressure applied on the device. For this reason, Linear Acceleration and Pressure appear in almost all of the best sensors subsets. The Call gesture is unique per user, as shown in a prior work [6] (see Section VII), which justifies why different sensors subset works well for different users.

Device-Specific Model: We grouped the data from the users who used the same devices. Seven users out of the fourteen users used Google Nexus, six used Samsung Galaxy and one used HTC. As in the previous model, we applied all subset of features and different classifier onto these resulting three data sets. The best features and the measurement values are calculated from running a 10-fold cross validation are shown in Table II. Combining the data from different users degrade the classifiers accuracy. The recall, precision and F-measure were above 0.83 for all the classifiers. The sensor subset for each device is subset of the sensors used by those users. For example, user 5, 6, 12, 15, 16 and 20 are the users who had Samsung Galaxy devices, therefore the best classifier subset for Samsung Galaxy is a subset of the sensors used by those users. User 14 is the only user who had HTC device. This is why the result for HTC is same as that for user 14.

Generalized Model: We combined the features from all the user into a single dataset. We used these features to generate a generalized classifier. The results are shown in Table II (last row). Similar to device-specific model, aggregating the data from different users degrades the classifier accuracy. The recall, precision and F-measure were all above 0.83.

B. Snap Detection

For the Snap gesture, we collected the data from **19 users** as described in Section IV-B. Each of these users performed 30 Snap operations. Similar to the Call gesture, we experiment three different classification settings for the Snap gesture.

User-Specific Model: We divided the data into nineteen sets based on the user id. Then, we built a classifier to distinguish the Snap gesture from other gestures collected by Call, Snap, Snoop and Control apps for each of the sets. The average time taken by the users to take a picture was four seconds. Therefore, we compared the Snap gesture with the four seconds of each other gesture. The best features and the measurement values are calculated from running a 10-fold cross validation as shown in Table III. The precision, recall and F-measure are on average 0.98, 0.99 and 0.98 respectively. While the user is taking a picture, she moves the phone and adjusts the orientation of the phone, this can be measured by the Accelerometer, Pressure, Orientation, Gyroscope, Magnetic Field, Rotation and Game Rotation sensors. For that reason, all the sensors subset includes at least one of those sensors. The accuracy of Snap classifier is better than for the Call, which might be because the average time for Snap is four seconds and that for Call is one second.

TABLE III. RESULTS OF USING THE OPTIMAL FEATURE SUBSET FOR THE CLASSIFICATION OF SNAP AND OTHERS

Classification Model	User ID /Device	Classifier	Features Subset	Precision	Recall	F-Measure
User-Specific	1	RF	P,Gy,A,R,LA	0.97	1.00	0.98
	2	SL	Gy,O,LA	0.97	1.00	0.98
	3	NB	M,P,GR	0.97	1.00	0.98
	4	L	G,P,M,GR	1.00	1.00	1.00
	5	NB	A,M,GR	1.00	1.00	1.00
	7	RF	A,P,M	0.97	1.00	0.98
	8	NB	O,A,M	1.00	1.00	1.00
	9	L	A,M,LA,GR	0.97	1.00	0.98
	10	RF	GR,M,O,P	0.97	1.00	0.99
	11	RF	G,Gy,P	1.00	1.00	1.00
	12	RF	A,G,Gy,P,R	1.00	1.00	1.00
	13	RF	GR,P	0.97	0.93	0.95
	14	RT	A,GR,G,R	0.91	1.00	0.95
	16	SMO	M,P,R	1.00	0.97	0.98
	18	L	A,LA,P	0.97	0.97	0.97
	19	RF	GR,Gy,P	0.97	1.00	0.99
	21	RT	G,LA,O,P,R	1.00	1.00	1.00
	22	RF	GR,Gy,P	1.00	1.00	1.00
	23	NB	A,GR,M,R	1.00	0.97	0.98
Device-Specific	Google Nexus	RF	A,GR,G,M, O,P	0.89	0.95	0.92
	Samsung Galaxy	RF	G,Gy,M,O,P	0.96	0.98	0.97
	HTC	RT	A,GR,G,R	0.91	1.00	0.95
	LG	RF	LA,M,P,R	1.00	0.98	0.99
Generalized	ALL	RF	A,LA,M,O,P	0.89	0.93	0.91

Device-Specific Model: We aggregated the users' data based on their devices. The best features and the measurement values are calculated from running a 10-fold cross validation are shown in Table III. The F-measure of the classifiers is above 0.92 for all the classifiers. We had only one user who had HTC phone in the Snap experiment; the result for the HTC is same as the results for user 14. For the other devices, the sensor subset is a subset of the best sensors subset of different users. Again, the classifier accuracy degraded when we combined data from different model phones.

Generalized Model: Here, we grouped the features from all the users into a single dataset. The results are shown in the last row of Table III. The precision, recall and F-measure are 0.89, 0.93 and 0.91 respectively. The sensor subset consists of subset of the sensors used in the user-specific model.

C. Tap Detection

For the Tap gesture, we could collect the data from **20 users** as described in Section IV-B. Each of these users performed 30 NFC tappings. We then performed three different experiments to evaluate our three classification models for the Tap gesture.

TABLE IV. RESULTS OF USING THE OPTIMAL FEATURE SUBSET FOR THE CLASSIFICATION OF TAP AND OTHERS

Classification Model	User ID /Device	Classifier	Features Subset	Precision	Recall	F-Measure
User-Specific	1	RT	G,M,GR	1.00	0.97	0.98
	2	NB	P,M	1.00	0.97	0.98
	3	RT	P,A,GR	1.00	0.97	0.98
	4	NB	GY,M,A	1.00	0.97	0.98
	5	NB	P,M	1.00	1.00	1.00
	6	RF	G,R,,GR	1.00	1.00	1.00
	7	RF	P,O,M	0.97	1.00	0.98
	8	NB	P,M	1.00	1.00	1.00
	9	NB	GR,M,P	0.94	1.00	0.97
	11	NB	A,Gy,M	0.97	1.00	0.98
	12	NB	A,G,O,P,R	1.00	1.00	1.00
	13	RF	O,P,R	0.92	0.96	0.94
	14	RF	A,LA,M	0.82	0.93	0.88
	16	RF	GR,M,O,P,R	1.00	0.97	0.98
	17	RF	A,GR,G,M,O	0.91	0.97	0.94
	18	RF	A,GR,G,M,O	0.97	1.00	0.99
	19	L	GR,G,O,P	1.00	1.00	1.00
	21	RT	M,P	0.97	1.00	0.98
	22	RF	Gy,O,P	1.00	1.00	1.00
	23	RT	GR,Gy,P	0.86	0.97	0.91
Device-Specific	Google Nexus	RF	GR,G,Gy,M, O,P,R	0.93	0.92	0.92
	Samsung Galaxy	RF	LA,M,O,P,R	0.96	0.97	0.96
	HTC	RF	A,G,LA,M	0.82	0.93	0.88
	LG	RF	A,G,M,O,P	0.96	1.00	0.98
Generalized	ALL	RF	GR,G,M,O,P	0.89	0.90	0.89

User-Specific Model: We divided the data into twenty sets according to the user id. Then, we built a classifier to distinguish the Tap gesture from other gestures collected by Call, Snap, Snoop and Control apps for each of the sets. The best features and the measurement values are calculated from running a 10-fold cross validation as shown in Table IV. The Tap gesture requires the user to move his device near to the NFC tag, the reading then starts, the user should keep the device on that position and then move it far from the NFC tag. Normally the users hold their devices in certain orientation while reading the NFC tag, finding the comfortable way to attach the NFC sensor in their device to the NFC tag. The phone movements can be measured by the change in Linear Accelerometer, Gravity, Accelerometer and Pressure sensors, and similar to Snap, the orientation can be measured by Orientation, Magnetic Field, Rotation and Game Rotation sensors. For that reason, each of the subsets includes at least one of each of those sets of sensors. The recall and F-measure were on average 0.98 and 0.97, respectively, which indicate very good performance.

Device-Specific Model: We aggregated the users' data based on their devices. The best features and the measurement values are calculated from running a 10-fold cross validation are shown in Table IV. The average recall and F-measure are 0.96 and 0.94 respectively for all of the classifiers.

Generalized Model: We grouped the features from all the users into a single dataset and we used these features to generate a generalized classifier. The results are shown in Table IV (last row). The precision, recall, and F-measure are above 0.89. The best sensors subset consists of the common sensors between all device models and some additional sensors.

VI. DISCUSSION

Local vs. Remote Classification: Our approach is based on machine learning. Once the machine learning classifier is trained offline, we use it to identify different gestures in real-time. This classifier testing phase can either be performed on user's device locally or outsourced to a remote server. The first approach allows the device to independently identify the gesture without relying on a third-party server and data connectivity. However, it may require more resources for the testing task. This is in line with many implementations that use machine learning for the purpose of malware detection. There already exist some apps, such as MyWeka in the Android Play store, which provides Weka implementation with limited machine learning classifiers. Similarly, [9] provides a weka library to implement machine learning approach for Android.

In the remote classification approach, whenever there is a need to identify a gesture, the sensor data would be sent to the server, which will perform the classification and provide the result back to the device (all communication between the device and server takes place over a secure channel). A similar approach of using remote server/cloud has already been proposed by Oberheide et al. for cloud based antivirus [15, 16]. The advantage to this approach is that the device does not require extra resources for testing the gesture and running the classifier. However, it needs to have a data connection. In case there is no data connection, the system may fall back to asking user for explicit gesture such as hand-waving/rubbing [12, 23]. A drawback of this approach is the need to trust the remote service – if this service is malicious and colludes with the device malware app, it will completely undermine the security of the system. The delay introduced in transmitting the sensor data to the server may reduce system's usability.

Power Efficiency: Since the battery-life is one of the most important factors when using smartphones, the power consumed by our approach should be minimal. As the gesture detection procedures in our approach lasts for no more than a few seconds, i.e., the sensors are activated for an average of one second to detect Call and four seconds to detect Snap and Tap, our approach is indeed quite power-efficient. We start the sensors when there is an intent for the activity and stop the sensors as soon as there is another intent which indicates that gesture has been performed, such as proximity change indicating phone is near the ear, clicking of the camera button, etc. Once the sensor recording is stopped, the data is fed to Gesture Identifier for gesture recognition as described in Section III-2. The detection approach itself (i.e., the testing phase of the classification task) is very light-weight and requires negligible amount of power.

Quarantine State: In case of NFC reading, when user moves the phone in close physical proximity (few cms) of a tag, i.e., when the OS detects that a tag is nearby, our Tap gesture detection procedure starts. However, the time duration between the detection of the presence of the tag and the tapping on the tag might be too short for the gesture to be meaningfully recognized. To address this situation, we rely upon *lazy authentication*, i.e., the tag information is read by the OS but kept in a quarantine state until the full tapping gesture is recorded and analyzed. Since tapping also involves bringing back the phone from tapping position to normal position, we also incorporate this movement into our Tap gesture. Recall that we used four seconds of Tap data to correctly identify the Tap gesture. The data/command that NFC tag provides to the smartphone will not be executed right away when the phone detects the NFC tag. Rather, the system would display an "NFC

read" message to the user so that the user will bring his device back to normal position. In the meanwhile, the message read from NFC tag will be placed in quarantine state/storage. Once the gesture has been correctly identified, the corresponding NFC data needs to be processed, otherwise the data must be dropped and not delivered to the application.

Fall-Back: Our detection approach has very low FNR with high recall. However, there might be certain situations where, because of the user or device orientation, the activity performed by the user is so distinctive that the detection mechanism may miss classify the gesture. For instance, the user might be on a boat or bus, or on a hammock where the sensor data may be completely different compared to when user is in a normal position. In cases when the phone is connected to a headphone/Bluetooth headset or when phone is in speaker mode, the user does not need to move the phone. In cars, the user may not be able to physically reach the phone. Also, in case of emergencies, users might perform the gesture in a different way than in normal scenarios, which may lead to a false negative. In these scenarios, when the gesture detection mechanism fails, there is a need to fall back to allow the user to access the desired resource. This can be solved either by prompting the user to press a "Yes/No" button, or by asking the user for the explicit gestures such as hand-waving or rubbing, as proposed in [12, 23]. In situations where users are not able to make the gestures, for example under extreme emergency, a voice command could be used. Also, the current smartphone OSs allow users to dial emergency numbers even without unlocking the phone. In a similar fashion, we can bypass our gesture detection module when a user presses the emergency dial button on the phone.

Benign Automated Access: There may be some apps/services which require automation. For example, a user may want to wish his friends "Happy Birthday" on their birthday by leaving a voice message where an app makes an automated call at midnight. In this case, the gesture will be missing and the app will not be granted the permission. For these kinds of services which need automation, the permission token can be provided to the app at the time when the user provides the gesture associated with it, though the related activity needs to be performed at some later point of time. For example, the permission token can be granted at the time when the user records the birthday message. The user can set the time when the activity, let's say Call, needs to be done. When the call is performed by the app, the Permission Controller checks if it already has the permission token. If it has the token, it skips the gesture recognition procedure otherwise it will communicate with Gesture Manager and follow the protocol as before to allow/deny the access to resources/services.

Social Engineering Attacks: The high precision shown in Section V implies that there is little chance that an application can perform Call, Snap or Tap without user knowledge. However, it is possible for a malware developer to perform social engineering attack to fool the user into providing a valid gesture. For example, malware developer can design a game which asks the user to move the phone in ways that emulate either Call, Snap or Tap. Our system is vulnerable under a false positive, i.e., when the malware is continuously requesting a permission to access the resource and the movement of the phone is such that the sensor data satisfies the gesture detection algorithm. In this situation, the malware will gain access to the resource. Such attacks require malware to wait constantly for the gesture synchronized with the benign app which makes the likelihood of detection of such an app easier by OS. Nevertheless, our defense will still significantly raise the bar against many existing malware attacks.

VII. Related Work

The majority of related prior work focuses on either preventing a device from getting infected or detecting a malware as soon as it has been installed on the device. The two most common defense approaches to detect malware are static analysis [22, 25] and dynamic analysis [3, 4]. Both detection techniques have their own drawbacks. Malware authors can avoid static analysis detection through simple obfuscation, polymorphism and packing techniques. Dynamic analysis, although more resilient, is still a posteriori approach which is quite risky to adopt since malicious parties would have already obtained the valuable information.

A user or an app can be authenticated in numerous ways, such as passwords or biometrics. However, these approaches are not user-transparent (e.g. passwords and biometrics) and may not be lightweight or sufficiently robust (e.g., biometrics). CAPTCHAs may also be used for human-vs-bot identification, but are not user-transparent besides being hard for the users, and many variations being vulnerable to automated and relay attacks. Our transparent gestures, on the other hand, address the problem of human-vs-bot identification (not user authentication) and are lightweight, efficient and robust.

Chaugule et al. [5] share a similar philosophy as our work. They look for hardware interrupts to identify software initiated actions vs hardware initiated actions for SMS and audio services. The key/touchscreen press action generates hardware interrupts which are used to identify if the request is initiated by human or malware. The problem with this approach is that it requires explicit human interaction generating the hardware interrupts which may not be always present, such as when user taps onto an NFC tag. The main difference between their work and our work is that they check if there is any user activity while we check if there is any user-aware activity.

Roesner et al. [19] proposed user-driven access control in which an app is granted permission when Access Control Gadgets (ACGs) capture user's permission granting intent. ACGs are UI elements exposed by each user-owned resource for applications to embed. In this work , whenever there is authentic user interaction with corresponding ACG, it grants the permission to an app to access the corresponding resource while our approach grants the permission whenever a user gesture is captured. It requires not only kernel level changes to grant the permission based on ACGs but also require application level modification, and suggests applications to use standard icons for ACGs. Our design has an advantage over their work in that our work requires no application level changes and supports services such as NFC which do not have any specific UI elements or ACGs associated with them. Additional ACGs for UI interaction is required for their design to work with the services like NFC.

Conti et al. [6] have done work similar to ours, but in the context of user authentication (not user-vs-bot identification). They investigated if a user movement while making/answering a phone call can be used as a biometric authentication measure.

Their work [6] uses Dynamic Time Warping (DTW) algorithm to analyze and detect the call making/answering gesture. Their experiments were done in a controlled setting and no real-world scenario has been captured. In contrast to their data , our data has been collected from users in real-life scenarios. Besides this, our algorithm uses machine learning classifiers to classify the data and uses multiple sensors as features of these classifiers. They consider only accelerometer and orientation sensors to detect the gesture while our approaches consider all the motion sensors and the position sensors as well as pressure sensor for better accuracy. Furthermore, our system involves gestures associated with NFC and camera access, in addition to phone dialing.

Li et al. [12] and Shrestha et al. [23] have shown that different gestures can be used for human-vs-bot identification. Shrestha et al. captured explicit hand waving gesture via light and accelerometer sensors. In the work of Li et al., implicit gestures, such as tapping, have been used to provide permission to an app which uses NFC, while explicit gestures, such as hand waving/rubbing, are captured to grant permission to send SMS and other services where no implicit gesture is involved. They capture the gestures using accelerometer and proximity sensors. Li et al. generate a template from user tapping, and then later calculate correlation between template and test data to make user-vs-bot decision. Their work only uses linear accelerometer sensor to identify the tapping gesture while our work uses all motion gesture provided by Android API to identify tapping, snapping and calling gestures. Also, we use machine learning to detect a gesture which has an advantage compared to storing a template and using threshold to differentiate the activity.

VIII. Conclusion and Future Work

We presented an efficient and lightweight approach to protect smartphone users against malware which may compromise user's privacy or cause monetary loss, specifically concentrating on phone calling, camera and NFC services. Given that these services are very popular among users and might already be under attack by malware, we believe that protecting them is a significant step. We proposed the use of transparent gestures associated with calling, snapping and NFC tapping as a means to defeat malicious applications attempting to misuse the underlying resources, but still allowing the user to have full access when desired. A key usability advantage of our approach is those users do not need to be concerned about our (human-vs-bot) authentication process as it takes place in the background. Besides malware, this approach can protect the users from *unintentional* calling, snapping and tapping.

The results show that our gesture detection mechanisms can fairly detect the three gestures from each other, and from other benign or malicious activities. In the future, we believe that gestures associated with other smartphone services, such as sending SMS or email, or web browsing, can also be integrated with our system. As new sensors become available on these smart devices, subsequent work may use the different sensors to identify other transparent gestures and further improve the accuracy for the calling, snapping and tapping gestures.

Acknowledgments

This work was partially supported by a US NSF grant (CNS-1201927).

References

[1] W. Augustinowicz. Trojan horse electronic pickpocket demo by identity stronghold. Available online at http://www.youtube.com/watch?v=eEcz0XszEic, June 2011.

[2] M. Ballano. Android threats getting steamy, 2011. http://www.symantec.com/connect/blogs/android-threats-getting-steamy.

[3] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Mobile Systems, Applications, and Services (MobiSys)*, 2008.

[4] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2011.

[5] A. Chaugule, Z. Xu, and S. Zhu. A specification based intrusion detection framework for mobile phones. In *Applied Cryptography and Network Security (ACNS)*, 2011.

[6] M. Conti, I. Zachia-Zlatea, and B. Crispo. Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2011.

[7] F-Secure. Bluetooth-worm:symbos/cabir. Available online at http://www.f-secure.com/v-descs/cabir.shtml.

[8] F-Secure. Worm:symbos/commwarrior. Available online at http://www.f-secure.com/v-descs/commwarrior.shtml.

[9] J. Figura. Machine learning for google android. Available online at http://www.cestina.cz/ obo/vyuka/projekty/figura-ml-for-android.pdf.

[10] J. Han, E. Owusu, L. Nguyen, A. Perrig, and J. Zhang. Accomplice: Location inference using accelerometers on smartphones. In *Communication Systems and Networks (COMSNETS)*, 2012.

[11] S. Kolesnikov-Jessop. Hackers go after the smartphone, 2011. www.nytimes.com/2011/02/14/technology/14iht-srprivacy14.html.

[12] H. Li, D. Ma, N. Saxena, B. Shrestha, and Y. Zhu. Tap-wave-rub: Lightweight malware prevention for smartphones using intuitive human gestures. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2013.

[13] P. Marquardt, A. Verma, H. Carter, and P. Traynor. (sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Conference on Computer and Communications Security*, CCS, 2011.

[14] Microsoft. What is user account control?, 2011. http://windows.microsoft.com/en-US/windows-vista/What-is-User-Account-Control.

[15] J. Oberheide, E. Cooke, and F. Jahanian. Cloudav: N-version antivirus in the network cloud. In *USENIX Security Symposium*, 2008.

[16] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian. Virtualized in-cloud security services for mobile devices. In *In Virtualization in Mobile Computing*, MobiVirt, 2008.

[17] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang. Accessory: Password inference using accelerometers on smartphones. In *Mobile Computing Systems & Applications*, HotMobile, 2012.

[18] N. L. Petroni, Jr. and M. Hicks. Automated detection of persistent kernel control-flow attacks. In *Conference on Computer and Communications Security (CCS)*, 2007.

[19] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *Symposium on Security and Privacy*, 2012.

[20] R. Schlegel, K. Zhang, X. yong Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *Network & Distributed System Security Symposium*, 2011.

[21] A. Seshadri, M. Luk, N. Qu, and A. Perrig. Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *Symposium on Operating Systems Principles (SOSP)*, 2007.

[22] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report*, 2009.

[23] B. Shrestha, N. Saxena, and J. Harrison. Wave-to-access: Protecting sensitive mobile device services via a hand waving gesture. In *Cryptology and Network Security (CANS)*. 2013.

[24] R. Templeman, Z. Rahman, D. J. Crandall, and A. Kapadia. Placeraider: Virtual theft in physical spaces with smartphones. In *Network & Distributed System Security Symposium*, 2013.

[25] D. Venugopal. An efficient signature representation and matching method for mobile devices. In *Wireless Internet (WICON)*, 2006.

[26] M. Ward. Smartphone security put on test, 2010. Available online at http://www.bbc.com/news/technology-10912376.