Privacy Leaks When You Play Games: A Novel User-Behavior-Based Covert Channel on Smartphones

Wen Qi†, Yichen Xu*, Wanfu Ding*, Yonghang Jiang†, Jianping Wang*, Kejie Lu†‡
† Department of Computer Science, City University of Hong Kong, Hong Kong SAR
‡ School of Computer Engineering, Shanghai University of Electric Power, Shanghai, China
Email: {qi.wen, yhjiang4-c}@my.cityu.edu.hk, {yichenxu3, wanfding, jianwang}@cityu.edu.hk, kejie.lu@upr.edu

Abstract—To protect user privacy, many smartphone systems, such as Android and Windows Phone, adopt the permission-based mechanism in which a user can evaluate the request of private information by a mobile app before installing it. However, recent studies show that the permission-based mechanism is vulnerable to application colluding attacks because two apps, which appear to be harmless individually, can establish a covert channel and use it to leak confidential information. In general, existing known covert channels usually work in a way that one app can modify the status of a system component, while the other can read the status. Even though several covert channel detection schemes have been proposed recently to fight against this type of covert channels, we point out that such designed covert channel detection schemes are not sufficient. In this paper, we demonstrate the proposed recently to fight against this type of covert channels, though several covert channel detection schemes have been system component, while the other can read the status. Even though several covert channel detection schemes have been proposed recently to fight against this type of covert channels, we point out that such designed covert channel detection schemes are not sufficient. In this paper, we demonstrate the possibility of establishing novel covert channels that work in quite different ways, in which one app (e.g., a game) can be designed deliberately such that the user will be induced to voluntarily modify the status of a system component (e.g., a motion sensor), while the other app can read the status of the system component. To validate our design, we implement three covert channels on Android. Our experiments show that these channels can bypass existing detection schemes. Moreover, we also measure the achievable throughput, error rate, and energy consumption in devices. The results demonstrate that our covert channels can achieve a transmission with high accuracy and low energy consumption. Our work sets a new alarm for the security issue of using smartphones.

Keywords—Smartphone security; Covert channel; Application colluding attack; Motion sensor

I. INTRODUCTION

In the past few years, smartphone users are increasing explosively. It is predicted that such growth will keep speedy in the near future. According to a recent study by Ericsson [1], the number of smartphone subscribers worldwide is 2.7 billion in 2014 and is expected to reach 6.1 billion by 2020. With such a rapid growth of users, smartphones are becoming a new domain for various hackers to steal sensitive data of users, such as leaking contacts [2], sending messages to contacts furtively [3], and stealing credit card information [4].

With respect to protecting private information on smartphones, a major threat is that some mobile apps may abuse the permissions of data or API access. Therefore, as a preventive approach, common smartphone platforms, like Android and Windows Phone 8, implement a permission-based mechanism to make sure users are aware of those permissions given to apps at installation. With this mechanism, if a user notices that an app requires too many sensitive permissions, such as the permissions to access both Contacts and Internet, the user may cancel the installation. Moreover, some anti-virus software can analyze permissions requested by each app as well.

Despite the wide adoption of the permission-based mechanism, recent studies show that a smartphone is vulnerable to some hidden attacks, namely, application colluding attacks, as first discovered by Schlegel et al. [5]. In a typical application colluding attack, two apps are deliberately designed such that each of them only requires a subset of permissions. But they can establish a covert channel to share and then leak confidential information. For example, a user may think that two apps are not malware because one app only requires for contact accessing while the other only requires Internet accessing. If the user installs both apps, they can collude and leak the user contacts through a covert channel to the Internet. Clearly, this kind of attack can bypass the permission analysis of current security products because apps are analyzed independently and application collusion is not considered in most current security products [6].

Under the concept of application colluding attacks, many smartphone covert channels have been discovered [6]–[10], in which the covert channels can be established based on various system functions (e.g., vibration, settings, volume setting, screen status), system log, file status, etc. Fig. 1 (a) shows a general model of information flow for these existing covert channels. In the model, one app collects users’ sensitive data and encodes the data to a sequence of “0”s and “1”s by changing the status of sensors or system components. Meanwhile, another app keeps monitoring the changes of the status and decodes the data accordingly.

For the model in Fig. 1 (a), existing covert channels are based on a principle that one app can modify the status of a system component, while the other can read the status. Based on such understanding, several covert channel detection schemes have been proposed recently, including...
Taint-tracking [11] (in 2010) and Advanced monitoring [12] (in 2011), which can detect the direct communication path. Although these detection schemes are viable for existing application colluding attacks, they may not be sufficient because the aforementioned principle is just one way of constructing a covert channel. In this paper, we discover a novel type of covert channels that work differently, i.e., they do not need one app to change the status of any system component. Fig. 1 (b) illustrates the main idea of the new covert channels. In this model, user’s behavior can be exploited to establish the covert channels between two apps. For instance, the first colluding app steals sensitive data from the user and encodes the sequence of “0” and “1” to different patterns of sensor status changes. However, the first app does not change the status directly. Instead, by deliberately designing game scenarios, the app can induce the user to make desirable actions that lead to changes to sensor status, such as keystroke, tapping, and rotation. Meanwhile, the other app monitors the status of the specific sensors and decodes the data according to sensors’ status. In brief, our new framework utilizes the user as one component of the covert channel, and thus existing mitigation solutions cannot detect the new covert channels.

To demonstrate the effectiveness of the proposed framework, we implemented three types of covert channels. The first one utilizes different frequencies of users’ tapping action on the touch screen to infer “0” and “1”. Since both the sending and receiving sides are required to measure the timing to calculate the frequency, this kind of covert channel belongs to timing based covert channels. The second type of covert channel employs the tap position (area) on the touch screen to indicate “0” and “1”, and the third types of covert channels exploit the orientation of the phone to transmit data. Both the second and the third types of covert channels are exploitable without the time reference, so they are storage based covert channels.

The major contributions of this paper are summarized as follows:

- We propose a framework of user behavior based covert channel (UCC) on smartphones, which takes the advantages of users’ behavior to the application colluding attack, making the information flow of the covert channel not trackable by existing detection solutions.
- We introduce three covert channel implementations based on the UCC framework. We design the apps that can induce the user to make desirable sensor status changes. We implement these covert channels on smartphones and demonstrate the effectiveness of UCC.
- To reduce the error rate of covert channels, we use the technique of pattern recognition to detect the motions (tapping and tilting) from users. Results show that we can achieve motion detection with high accuracy, thus reducing the error rate of covert channels.
- We evaluate the stealthiness of our framework by trying to detect the three covert channels with popular covert channel detection tools. The result shows that covert channels designed by us cannot be detected.
- We analyze the energy consumption of the three covert channels. The result shows that the energy consumption introduced by the proposed covert channels is very little.

The remainder of this paper is organized as follows. In Section II, we discuss our preliminary studies on analyzing the relationship between users’ behavior and app scenarios as well as introducing two sensors on smartphones. In Section III, we introduce the architecture design of the UCC framework and challenges of implementation. Section IV presents the technical details. The UCC framework evaluation is given and analyzed in Section V. In Section VI, we introduce the background and related work on smartphone covert channels. Finally, Section VII concludes this paper.

II. PRELIMINARY STUDY

In this section, we first use some experiments to demonstrate that a deliberately designed app can lead to expected user’s behaviors with high accuracy. We then discuss two common sensors on smartphones, accelerometer and orientation sensor, and we show how their status can be changed according to users’ behaviors. This preliminary study enables us to further design user behavior based covert channels with low error rates in the following sections.

A. User Behavior Induced by Apps (Games)

In practice, many apps, especially video games, require users’ real time reactions such as tapping, shaking, and tilting. Moreover, almost all games use rewards and scoring mechanism to encourage users to react with accuracy. In other words, to obtain higher scores or accomplish certain target, users need to follow the rules in the game. For example, to gain scores, the user is required to whack on
moles in the game Whack–A–Mole and to slice the fruits in Fruit Ninja.

To understand the accuracy of a user’s action, we have designed a game and conducted a survey to confirm: First, users will follow the app (game) scenarios and perform expected motions; Second, different motions and different patterns of these motions can be differentiated.

In our survey, we have invited 50 volunteers to play a modified Whack–A–Mole game on one Android phone. Once the game starts, m moles pop up one by one at a fixed time gap, \( \Delta T \). Each mole appears in a random position in the screen and disappears automatically before the next mole appears. The player is asked to tap the mole on the screen before the mole disappears. The app monitors each tap action from the player and record \( m \), \( \Delta T \), and \( m' \) which is the number of taps made by the player.

Each volunteer is asked to play the game completely for 2 rounds. Before the game, the player is told to try to whack on moles to gain credits in the game and there are no other limitations. The settings of each round are as follows.

- \( m \) is set at 50.
- In the first testing round, \( \Delta T = 1000ms \).
- In the second testing round, \( \Delta T = 550ms \).

For different \( \Delta T \), the difficulty of the game is different. The smaller \( \Delta T \) is, the more difficult the game is, since the user has less time to react after the popping up of moles.

In Fig. 2, we report the average tap frequency of each user. We can see that users’ tap frequencies are close to the frequency of moles’ popping up. In round 1, all tap frequencies of different users are about 1 tap/s, which is the same as the frequency of moles’ popping up. And the deviation of data among different users is very small. In round 2, moles pop up with a frequency 1.8 moles/s and all of the users’ tap frequencies are above 1.75 taps/s while 92% of users’ tap frequencies are in the range [1.75, 2.00]. In this figure, a clear gap can be observed between the tapping frequencies in different rounds. With this observation, we can see that it is feasible to utilize different tap frequencies to transmit data “0”s and “1”s.

We now explain how the tap frequency can be made available to another app. Note that the exact user’s action, such as the location of a tap, is only reported to the game app. However, the user’s tap motions can cause the change of orientation and accelerometer, which can be observed by all apps. Next, we further explain more details of these sensors.

B. Accelerometer

A typical accelerometer monitors the phone’s acceleration along three axes (x, y, and z), as demonstrated in Fig. 3. Since the accelerometer has a high sensitivity to any tiny change of accelerations, even small motions from the user, like taps, can be reflected from the change of three axes. Therefore, an app can induce the user to conduct tap motions to transmit “0”s and “1”s, while another app can receive the change of accelerations in the background and detect the motions of the user based on the acceleration data.

C. Orientation Sensor

The orientation sensor monitors the orientations and postures of the smartphone. Each orientation event reflects the change of phone’s orientation by measuring angles of the smartphone along three dimensions:

- **Azimuth**: The angle between the phone’s y-axis and north when the smartphone rotates along the z-axis. As illustrated in Fig. 4 (a).
- **Pitch**: The angle between perpendicular of touchscreen side and z-axis when the phone rotates along the x-axis. As demonstrated in Fig. 4 (b).
- **Roll**: The angle between perpendicular of touchscreen side and z-axis when the phone rotates along the y-axis. As shown in Fig. 4 (c).

Motions, like taps on different area of the screen and tilting the phone to different directions will make the phone’s orientation change with different patterns, which can be observed by another app and thus utilized to transmit “0”s and “1”s.
III. ARCHITECTURE DESIGN AND CHALLENGES

In this section, we first introduce the architecture of UCC. Then, we present the design of different covert channel implementations.

A. Architecture

As shown in Fig. 1 (b), the UCC framework includes four key components, namely, a sender, a smartphone user, motion sensors in the phone, and a receiver which reads sensors’ status changes and transmits decoded data to the Internet.

1) Sender: The sender can be an app like a game that has two main functions, namely, data collection and data encoding, respectively. The data collection function collects sensitive data from the phone, like phone ID (IMEI), contacts, bank account, current location, etc. A lot of applications require the access to sensitive data, e.g., games have functions like sharing to friends, in-app payment, and location tracking. To encode data, the sender induces the user to conduct different motions with different frequencies as well as at different time points where the pattern of these motions can be used to represent “0” or “1”. The motions can then consequently cause the changes of sensors which can be differentiated by analyzing from sensors. The expected motions are made possible by offering the user different positions and tilting motions. These motions directly affect the status of sensors, like accelerations, orientations, etc.

2) User: The user plays a passive role in the framework, which conducts different motions based on different scenarios given by the sender app. These motions directly affect the status of sensors, like accelerations, orientations, etc.

3) Sensor: The sensor is another passive role in the framework. Due to the user’s motions, the sensor’s status changes and these changes can be read by the system and apps. Thus, the patterns of sensors’ changes caused by the user’s motions can be captured by the receiver which receives notifications from the sensor.

4) Receiver: The receiver is another app running in the background. It has three main functions, sensor data collection, data decoding, and data leaking, respectively. First, the receiver collects sensor data by registering sensor event listeners and receiving notifications from the sensor manager continuously. Then, with collected sensor data, the receiver detects the patterns and decodes the sequences of “0”s and “1”s according to predefined patterns. Finally, the receiver is responsible for leaking the data, e.g., to a remote server via Internet. One remaining issue is how to launch the receiver. To this end, we envision that the receiver can be designed as a useful tool, such as an e-health app that can monitor users physical activities. Then the sender can recommend the user to download and launch the receiver by periodically prompt a message to encourage the user to install the receiver, which is a common practice in many apps. In such a way, although we cannot guarantee that the receiver is always launched when the sender is running, in practice, many users may eventually install and launch the receiver.

B. Challenges

At the sender side, to let the user conduct motions according to the app’s scenarios, we need to carefully design the app, making sure the user to make motions as expected. Thus, the design of the sender app is tricky in the sending process. In Section IV, we will show how we design different game scenarios which induce different user behaviors.

From the sensor’s aspect, motions of the user can be based on the tapping, shaking, and tilting of the phone. The sensors utilized will directly impact the reliability of the channel. Besides, motion detection requires a very high accuracy. Otherwise bit loss or insertion will happen and noises are introduced to the channel. Thus, how to detect the motions becomes another challenge. In our implementations, we will introduce two patterns in the accelerometer data to detect tap motions and utilize the orientation sensor to detect tap positions and tilting motions.

The synchronization between the sender and the receiver is another important problem in the covert channel design. Without synchronization, the receiver does not know the start point and end point of the data transmission. Sensors’ data before the transmission or after the transmission may be decoded as extra bits which are noises to the channel. There can be various methods to synchronize between the sender and the receiver. For example, specific motions can be required to start and finish the game, such as shaking the phone for at least a predefined number of times. Once these motions are conducted by the user, they will be detected by the sender and the receiver at the same time since the receiver is receiving the change of motions in the background. Then, the start points of the transmission process at both sides are aligned. After that, the data stream from sensors is recorded by the receiver.

IV. IMPLEMENTATION DETAILS

In this section, we discuss the implementations of three different covert channels based on the UCC framework, and we elaborate on the technical details involved in the implementation.

A. Accelerometer based Covert Channel: TapFreq

We first introduce the design of a timing-based covert channel, namely TapFreq, which exploits the tap frequency of the user with the help of the accelerometer.
Based on this, we design the following rules for the receiver.

1) Covert Channel Design: In our design, the sender is a game app like Whack–A–Mole which is illustrated in Fig. 5. When the game starts and ends, the user is required to conduct a predefined number of shaking motions to send signals to the receiver. This synchronization process will also be applied to all other proposed covert channel designs. Once the game starts, the sender continuously displays moles on the screen and each mole disappears after a fixed time period.

To facilitate data transmission via the covert channel, a possible encoding method can be as follows. Assuming that the receiver app can detect the tap motions, we can utilize different frequencies of users’ tap actions to transmit different bits. Specifically, from the beginning of the game, the sender divides the time into fixed-size time-slots with a fixed length $T$. The number of moles that appears during each time-slot is controlled by the sender. In each time-slot, the sender transmits 1 bit. For each bit $b_i$, the sender shows $n_i$ moles during the time-slot. $n_i$ is a random number uniformly chosen from a range depending on the bit to be transmitted:

\[
\begin{align*}
N_{\text{min}}^0 &\leq n_i \leq N_{\text{max}}^0 & &\text{if } b_i = 0 \\
N_{\text{min}}^1 &\leq n_i \leq N_{\text{max}}^1 & &\text{if } b_i = 1
\end{align*}
\]

where $N_{\text{min}}^0 < N_{\text{max}}^0 < N_{\text{min}}^1 < N_{\text{max}}^1$ are predefined parameters that are also known by the receiver.

The receiver continuously monitors the acceleration data to detect the “shaking” signals. Once the receiver detects the beginning of a transmission, the receiver starts monitoring the tap motions from the user. Similar to the sender, the receiver divides the time period into fixed-size time-slots and detects the number of tap actions in each time-slot. From the preliminary study in Section II, we learn that a user usually follows the appearance of moles in the game to tap the preliminary study in Section II, we learn that a user usually follows the appearance of moles in the game to tap the preliminary study in Section II, we learn that a user usually follows the appearance of moles in the game to tap.

The receiver continuously monitors the acceleration data to detect the “shaking” signals. Once the receiver detects the beginning of a transmission, the receiver starts monitoring the tap motions from the user. Similar to the sender, the receiver divides the time period into fixed-size time-slots and detects the number of tap actions in each time-slot. From the preliminary study in Section II, we learn that a user usually follows the appearance of moles in the game to tap.

The receiver continuously monitors the acceleration data to detect the “shaking” signals. Once the receiver detects the beginning of a transmission, the receiver starts monitoring the tap motions from the user. Similar to the sender, the receiver divides the time period into fixed-size time-slots and detects the number of tap actions in each time-slot. From the preliminary study in Section II, we learn that a user usually follows the appearance of moles in the game to tap.

Therefore, even $n_i'$ does not equal to $n_i$, the receiver can still correctly classify $n_i'$ if $N_{\text{min}}^1 - N_{\text{max}}^0$ is large enough. Based on this, we design the following rules for the receiver to determine the transmission bit.

\[
b_i = \begin{cases} 
0 & \text{if } n_i' \leq \frac{N_{\text{max}}^0 + N_{\text{min}}^1}{2} \\
1 & \text{if } n_i' > \frac{N_{\text{max}}^0 + N_{\text{min}}^1}{2}
\end{cases}
\]

With the above protocol, the sender can transmit bits “0”s and “1”s to the receiver. The next issue is how the receiver correctly detects the taps in each time-slot, which will be explained next.

2) Tap Detection: Before introducing the detection details, we first explain why we can utilize the accelerometer to detect tap motions. We know that a typical tap event consists of three phases, action down, hold, and action up. During the action down phase, an external force is applied to the phone and the direction is $-z$ axis. During the action up phase, the phone is affected by the return force from the hand that holds the phone. The return force is mainly in the $z$-axis.

In our experiments, the quadratic sum of the acceleration from three axes shows clear patterns for each tap motion. The quadratic sum is denoted as $F^2$ and defined as:

\[
F^2 = \sqrt{F_x^2 + F_y^2 + F_z^2}
\]

where $F_x^2$, $F_y^2$, and $F_z^2$ are the acceleration in $x$, $y$, and $z$-axis, respectively. In Fig. 6, we show some real trace of $F^2$ corresponding to three tap actions. From the figure, we can clearly observe a peak and a trough in the wave for each tap motion. Therefore, tap motions can be detected with the accelerometer by recognizing these patterns.

Since each user’s tap behavior is different, we cannot apply a unified detection model to detect all users’ tap motions. Therefore, for each user, we first collect labeled data as the training set and extract user-specific features which represent the user’s tap behavior. Then, the receiver detects the tap motions with these user-specific features.

Training Set and Feature Extraction: The training set consists of the acceleration data for each tap motion. When the user interacts with the receiver app, the receiver can legally receive the tap event from the system. Thus, the start point and end point of each tap motion can be marked in the acceleration data stream based on the arriving time of events from the system. And the receiver records the acceleration data during the tap duration. With the training set, the receiver then extracts user-specific features.

In our experiments, there are two patterns in the accelerometer data for each tap event, as shown in Fig. 7, in which we name them as the DownUp pattern and the UpDown pattern, respectively. Most of the observed patterns belong to the DownUp pattern, which is similar as reported in [13]. In the DownUp pattern, $F^2$ decreases significantly due to the action down phase of a tap and jumps up due to the reaction of the hand holding the phone. Another pattern,
UpDown, has not been reported before. In UpDown, the $F^2$ increases dramatically and goes down quickly. This pattern appears when there is a react of the phone (upward) before the finger touches the screen. For these two patterns, we consider the following features respectively:

- Extreme values: the minimum ($\min$) and maximum ($\max$) value of $F^2$ during the tap event as well as the difference between $\min$ and $\max$.
- The standard deviation, which measures the fluctuation of the data for each tap event.
- Time intervals: the time interval between the start point and the time point of $\min$, as well as the time interval between the start point and the time point of $\max$.

Training and Detection: With extracted features from the training set, there are many choices to do training and classification. Due to the consideration of computing power in the phone and the “online” requirement of the classification process, we apply a static approach which is similar as proposed in [13]. The steps can be summarized as follows.

In the training stage, for each feature $i$, we generate a range $[L_i, U_i]$ based on the distribution of feature $i$. Specifically, $L$ and $U$ are defined as:

\[
\begin{align*}
L &= Q_1 - k(Q_3 - Q_1) \\
U &= Q_3 + k(Q_3 - Q_1)
\end{align*}
\]

where $k$ is a constant, $Q_1$ and $Q_3$ are the lower and upper quartiles, respectively.

In the detection stage, the receiver maintains a data stream window which keeps the latest accelerometer data. First, we try to identify a potential DownUp pattern by verifying whether (and when) the $\min$ appears, or a potential UpDown pattern by verifying whether (and when) the $\max$ appears. If a potential DownUp or UpDown is found, then we extract features from the data in this window and compare them with the features obtained in the training stage. If all extracted features are in their corresponding range $[L_i, U_i]$, then a tap motion is detected.

3) Shaking Detection: The basic idea of shaking detection is similar as the tap detection: the user’s motions cause the change of accelerometer’s status. To detect the shaking motions, both the sender and the receiver receive the changes from the accelerometer and once the $F^2$ reaches a threshold, the shaking motion is detected. In order to differentiate that the shaking is conducted by the user, but not due to the activities like running or walking, an intuitive approach is to set the threshold extremely large and can only be reached when the user shakes the phone vigorously.

B. Multi-sensor based Covert Channel: TapPos

In this section, we present the design of a storage based covert channel, TapPos, which exploits the tap position with multiple sensors. This kind of covert channel is similar to TapFreq, where the sender is a game app like Whack-A-Mole.

1) Covert Channel Design: The sender divides the screen to multiple parts and defines the bit(s) to transmit if a specific part is tapped by the user. Once the game starts, according to the data to transmit, the sender displays moles in different parts to induce the user to tap. A simple design is to divide the screen into two parts, the top part and the bottom part, as shown in Fig. 8 (a). In this way, the sender can transmit “1” and “0” by showing moles in the top part and the bottom part, respectively. Similarly, when the screen is divided into 4 parts, as shown in Fig. 8 (b), each tap can be used to transmit 2 bits. To decrease the number of unexpected taps or missing taps, the sender should leave the user enough time to react after showing a mole, including the interval between two moles and the time of existence of a mole.

The receiver infers the data transmitted by the sender by detecting the position of each tap correctly. Since the tap motions changes the status of both the accelerometer and the orientation sensor, the receiver receives the data from both sensors to detect the tap position. In particular, the receiver first detects the tap motion with the same technique as introduced in TapFreq, then analyzes the tap position based on the data from the orientation sensor. Based on the detected tap position, the receiver can infer the data according to the rules defined by the sender. Next, we discuss more details of tap position detection.

2) Tap Position Detection: We first introduce why we can utilize the accelerometer and the orientation sensor to infer the tap position. As introduced in TapFreq, we can detect the tap motion with the accelerometer. Then, we focus on
how the orientation sensor helps to detect the tap position. We know that when different parts of the phone are tapped, the orientation of the phone changes differently.

Intuitively, when the top of the screen is touched, the top of the smartphone goes down, while the value of the Pitch increases. Similarly, when the phone leans right, the right-hand side of the smartphone goes down, while the value of the Roll decreases. Based on such facts, when we divide the screen to two parts, as shown in Fig. 8 (a), we only need to focus on the Pitch value of the phone. On the other hand, if the screen is divided to more than two parts, as shown in Fig. 8 (b), other axis, like Roll, can be introduced to differentiate whether the tap is acted on the left-hand side or the right-hand side.

To briefly summarize, the receiver can recognize the tap position by sensing the changes from two sensors: (1) the accelerometer by reading $x$, $y$, and $z$-axis, and (2) the orientation sensor by reading the Pitch value and the Roll value.

In our design, to detect the tap position, we first identify the tap motions in the data stream based on the techniques introduced before. Then, for each detected tap motion, we record the start time point $t_s$ and the end time point $t_e$. With $t_s$ and $t_e$, we obtain corresponding data segment in the orientation data stream for further analysis.

We now use examples to illustrate how we determine the tap position in the $y$-axis. In Fig. 9, there are three tap actions. The first and third taps are acted on the bottom of the phone and the second tap is conducted on the top of the phone. We can see that, when the top of phone is tapped, the Pitch value rises quickly and decreases after that, which is similar to the pattern $UpDown$ as introduced before. In contrast, when bottom of the phone is tapped, a $DownUp$ pattern of the wave is observed. With these two observed patterns, we use a quite naive but also very effective method to distinguish the tap position between the top part and the bottom part.

The main steps are as follows:

- For all orientation data $f(t_i)$ where $t_a \leq t_i \leq t_e$, find the indexes of both peak and trough, denoted as $t_{\text{max}}$ and $t_{\text{min}}$, respectively. Thus, we have:
  \[
  f(t_{\text{min}}) = \min_{t_a \leq t_i \leq t_e} f(t_i), \\
  f(t_{\text{max}}) = \max_{t_a \leq t_i \leq t_e} f(t_i)
  \]

- Classify the pattern of the orientation Pitch data based on $t_{\text{max}}$ and $t_{\text{min}}$. The classification result is denoted as:
  \[
  c(i_p, i_t) = \begin{cases} 
  U p D o w n & \text{if } t_{\text{max}} < t_{\text{min}}, \\
  D o w n U p & \text{if } t_{\text{max}} > t_{\text{min}},
  \end{cases}
  \]

  where DownUp means the tap action on the bottom side is detected and UpDown indicates the tap action on the top.

To distinguish the tap position in the $x$-axis, we apply the same method of analyzing the Roll reading of the orientation sensor. When the DownUp pattern is observed for the Roll values, a tap on the right hand side is detected. Similarly, when an UpDown pattern is observed for the Roll values, a tap is acted on the left hand side.

C. Orientation Sensor Based Covert Channel: TiltGame

We now discuss the third type of covert channel, TiltGame, which only exploits the orientation sensor to establish a covert channel and is different from both TapFreq and TapPos.

1) Covert Channel Design: The sender of TiltGame can be a typical racing game with a vertical UI which requires the user to tilt the phone along the $x$-axis to change the direction of the car or motorcycle in the game. The sender transmits different data by generating paths with different directions.

For example, a path that requires the user to tilt the phone left to indicate “0” and tilt the phone right to indicate “1”. The tilting angle of the phone is monitored by the sender. When the angle is smaller than the predefined minimum threshold or larger than the maximum threshold, a tilting motion is detected and the position of the object in the game will be moved. To make consecutive “0”s or consecutive “1”s detectable in the receiver side, the sender can apply a policy that the user should return the phone back to the horizontally status, such that there is a gap between two “0”s or two “1”s. A sample UI design is shown in Fig. 10, in which the user needs to tilt the phone left to avoid collision.

To receive the transmitted data correctly, the receiver of TiltGame should detect the direction of the tilting motion. When the user tilts the phone, the phone’s orientation changes and these changes can be reflected by the orientation sensor. Therefore, the receiver should read the status change of the orientation sensor continuously. To make sure that the receiver has the same definition of a tilting motion (the angle of the tilting motion), the receiver shall apply the same thresholds as defined by the sender. Once a tilting with left
direction is detected, a bit “0” is received. In contrast, when a tilting with right direction is detected, a bit “1” is received.

2) Tilting Motion Detection in TiltGame: As discussed before, the tilting motion changes the orientation of the phone, and thus we can utilize the orientation sensor to detect the tilting motion. Besides, the sender has a vertical UI designed in a way such that the tilting motions are along the x-axis and mainly affect the Roll reading of the orientation sensor.

In our design, the tilting motion usually takes hundreds of milliseconds and the wave of the Roll reading is quite stable. Some examples for the tilting motions are shown in Fig. 11, where we can observe that a threshold based method can work well. Next, we summarize the main steps in this covert channel design:

- Find the start of the tilting motion. For each Roll reading \( \text{roll}(t_i) \), if \( \text{roll}(t_i) \geq \text{thr}_1 \), mark \( t_i \) as the start of a tilting motion and the transmitted bit as “1”. If \( \text{roll}(t_i) \leq \text{thr}_0 \), mark \( t_i \) as the start of a tilting motion and the transmitted bit as “0”.
- Find the end of the tilting motion which is the first \( t_i' \) that fulfills \( t_i' > t_i \) and \( \text{thr}_0 \leq \text{roll}(t_i) \leq \text{thr}_1 \).
- Repeat the above two steps to detect the following tilting motions.

V. EVALUATION

In this section, we focus on the evaluation of the three different covert channels introduced before. The aim of our experiments is to verify the feasibility of user behavior based covert channels. All the presented covert channels have been implemented on Samsung Galaxy S5 with Android 4.4.2.

A. Throughput and Accuracy

1) TapFreq: To evaluate the throughput and accuracy, we design the Whack–A–Mole game with the target to send 50 bits for 5 rounds in 5 different time-slot durations in the TapFreq implementation. In the experiments, users are told to try their best to play the game and there are no other limitations. The accuracy accounts for a user’s "mistake", including no motion and wrong motion. The average throughput and accuracy rate for different time-slot duration, \( T \), are reported in Fig. 12 (a). We can see that the throughput of the channel decreases as the increase of \( T \) and the accuracy is more than 95%. Considering the practicable tap frequency of a user, \( T \) cannot be smaller than 500 milliseconds, otherwise the user cannot follow the game with a high accuracy and a high error rate will be introduced to the channel.

Nevertheless, users’ sensitive data, like IMEI and phone number can be leaked within one minute with a very high accuracy. Since both IMEI and phone numbers only consist of numbers, a 4-bit numeric encoding scheme can be used. Due to the high accuracy during the transmission, we assume no retransmission or error-correcting code is needed. Therefore, we only need 60 bits to transmit a fifteen-byte IMEI. When \( T = 1 \)s, the throughput is 1bps and the transmission time is 60 seconds. Similarly, to leak a 10-byte US phone number, it takes only 40 seconds to transmit the 40 bits encoded data.

We apply data compression techniques, like Huffman coding, to compress the data and transfer fewer bits in the channel, which means even less time is needed to leak an IMEI and a phone number in the contacts. We measured the time to transmit five real IMEI with Huffman coding and compare the transmission time with the transmission time when no coding technique is applied. The result is shown in Fig. 12 (b). We can see that, in average, the Huffman coding saves 15.67% of the transmission time.

2) TapPos: For the TapPos implementation, we conducted similar experiments as that for TapFreq. Users played the game 5 rounds for different time interval between the appearances of two moles, denoted as \( \Delta T \). In each round, 50-bit data is transmitted which means the expected number of tap actions on the screen is 50. \( \Delta T \) is varied from 0.6s to 1.0s.

When the screen is divided to 2 parts, the results are shown in Fig. 13 (a). We can see that different \( \Delta T \) results in different throughput and accuracy, between which there is a trade-off. As the increase of \( \Delta T \), the throughput decreases from 1.67bps to 1bps while the accuracy raises from 86.0% to 98.4%. We can see that the throughput is around 1bps and the channel achieves a high accuracy rate which ranges from 96% to 100%.

With TapPos, it is also possible to leak users’ sensitive data in one minute. When \( \Delta T = 0.8 \)s, the throughput is 1.25bps and it takes 48s and 32s to leak a 15-byte IMEI and 10-byte phone number with the 4-bit encoding.
Then, we evaluate the performance when dividing the screen to 4 parts, which means one tap motion is utilized to transmit 2 bits. As we can see from Fig. 13 (b), the throughput increases compared with that in Fig. 13 (a). However, due to the higher requirement of the position detection, the accuracy is slightly lower than that in Fig. 13 (a), which is due to the more complex tap position detection.

If a more precise position detection technique is applied, 3 or even more bits can be transmitted with one tap action by dividing the screen to 8 or more areas. In this case, the throughput can be boosted substantially.

3) TiltGame: For TiltGame, the experimental settings are the same as that in previous testing scenarios. We varied the time interval between two tilting motions, $\Delta T$, from 0.6s to 3.0s. As presented in Fig. 14, the results show that when $\Delta T \geq 2.0s$, there is no error during the transmission. When $\Delta T$ is set as 1.0s, the error rate is still very small, 3%. Thus, it is also practical to leak sensitive data like IMEI and a phone number in one minute. Based on these results, we can see the risk of TiltGame is similar as TapFreq and TapPos.

B. Energy consumption

The energy consumption is essential to the stealthiness of a covert channel. If a covert channel consumes much energy, it is easy to be noticed by the user or monitoring software. In this section, we present the total energy consumption by the sender and receiver in two scenarios: (1) covert channel is turned on, (2) covert channel is turned off, which means covert channel related code will not be executed. For each covert channel implementation, we measure the energy consumption with PowerTutor [14] for a time period of 1 minute.

The results are shown in Fig. 15. Generally, we can see that, when the covert channel is turned on, the two apps consume more energy than that when covert channel features are turned off. Nevertheless, the overhead of energy consumption introduced by these covert channels is really small, only increased by 4.14%, 6.59% and 1.60%, respectively. Since these energy overhead is insignificant. It is hard to be noticed by the phone user. Another thing we should notice is that the overhead introduced by TapPos is much higher than the other two. Because in TapPos, the data from two sensors are handled and the data processing is more complex than the other two.

C. Stealthiness

We evaluate the stealthiness of our framework by evaluating TapFreq, TapPos and TiltGame with popular covert channel detection tools. The results show that these covert channels cannot be detected.

1) TaintDroid: TaintDroid [11] works by monitoring apps’ data access and data manipulation. It labels privacy-sensitive data and tracks the information flow of labeled data. Once labeled data is transmitted out of the system, like via Internet, TaintDroid notifies the user.

We have done two contrast experiments to confirm that our covert channel implementations cannot be detected by TaintDroid. The experiments were conducted with TaintDroid 4.3 on Nexus 4. In the first experiment, the app read the IMEI of the phone and sent out via Internet. The TaintDroid detected this behavior and notified user. We show the screen of the TaintDroid notification in Fig. 16. In the second experiment, for each proposed covert channel, the IMEI was collected by the sender app, then transmitted to the receiver app and sent out by the receiver app via Internet. For these three covert channel implementations, no information leakage is monitored by TaintDroid. The results are summarized as in Table I.

2) XManDroid: XManDroid [12] introduces a system-centric system policy to Android and implements a security framework that monitors the direct IPC calls and indirect communication with system API between apps at run-time.
With this framework, several existing collusion attack based covert channels can be prevented [6]. We could not evaluate XManDroid because its code is not public currently. However, after the careful analysis of XManDroid, we believe it cannot detect our covert channels, because there are no direct IPC calls or indirect communication through system components in our covert channel design.

VI. RELATED WORK

In this section, we first introduce the background knowledge of smartphone security and covert channels. Then, we present the principles of application colluding attacks.

A. Covert Channels on Smartphones

A covert channel is a communication channel to secretly transmit data between two ends, i.e., it is not supposed to exist according to the system design. The concept has been proposed for decades [15]. People start the research on covert channels in the area of computer systems, such as network based covert channels [16], [17], computer operating system-based covert channels [18], [19], and computer hardware-based covert channels [20], [21]. Recently, with the development of cloud computing, various covert channels have been discovered among virtual machines, known as cross-VM covert channels [22]–[25]. Meanwhile, with the rapid growth of the smartphone market, smartphone covert channels [5]–[10] also start to attract peoples’ attention.

Modern smartphones are generally equipped with various on-board sensors. These sensors become potential media utilized by covert channels. Some of these sensors, e.g., microphone, camera, and GPS sensor, can be utilized by the sender to collect sensitive data. Other sensors, such as vibrating sensor, accelerometer, and gyroscope, can be manipulated as a vehicle to leak confidential information [13], [26], [27].

B. Application Colluding Attack

Existing covert channels on smartphones are mainly discussed together with colluding attacks. Soundcomber [5] was the first to introduce the colluding attack on smartphones. They implemented a sender to extract sensitive information, such as credit card number, by monitoring microphone during the phone call. The collected data is sent to a colluding app, the receiver, which has Internet permission over covert channels. In the paper, four types of covert channels are proposed, namely, vibration setting, volume setting, screen status, and file locks. After that, [6] comprehensively analyzed existing colluding attacks on smartphones. They also proposed multiple new covert channels which utilize intents type, UNIX sockets, app’s threads number, CPU usage, and so on. Later, [7] designed covert channels with high covertness by minimizing the permission requirement. Covert channels including screen status and process priority are proposed. They also evaluated these designs with TaintDroid and showed the covertness of these designs. In recent years, new smartphone covert channels are proposed continually. In [8], vibration motor and accelerometer are utilized to produce/read vibration patterns of the phone. In [9], the covert channel is constructed with the speaker to produce inaudible sound and the microphone to decode data from received inaudible sound. And in [10], new hardware resources, namely, battery and phone call, are utilized to establish covert channels.

C. Detection Methods for Application Colluding Attacks

Existing mitigation solutions include Taint-tracking [11], Advanced monitoring [12], and Energy-use anomaly detection [28]. Energy-use anomaly detection [28] tracks the abnormal decrease of battery caused by energy consumption of attacks. In [29], an Android Security Modules framework is proposed, with which the developers can define new reference monitors for Android. Thus, security solutions like TaintDroid and XManDroid can be implemented with this framework.

VII. CONCLUSION AND FUTURE WORK

In this paper, we show that a smartphone user’s behavior can be utilized by malware to construct a covert channel which has not been reported before. We have presented three covert channel implementations in Android. The experiment results show that these covert channel implementations can bypass well-known detection solutions, like TaintDroid. We have measured the achievable throughput, accuracy rates, and energy consumption of the three covert channel implementations on Samsung Galaxy S5. The results show that these covert channels have high transmission accuracy and low energy consumption. The risk and stealthiness of these covert channels should be informed to the public.

In the future, we will study the countermeasures to the proposed user behavior based covert channels. Potential solutions include analyzing sensors’ change, detecting the patterns in the sensors, limiting non-system apps’ access.
to high precision data of sensors, and adding noises to the sensor's data.

ACKNOWLEDGEMENT

The work is supported in part by a General Research Fund from Hong Kong Research Grant Council under project 122913 and project 61272462 from NSFC China, and by the Shanghai Oriental Scholar Program.

REFERENCES