This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

Detecting the Capacitance-based Gamepad for Protecting Mobile Game Fairness

Shilei Bai, Bin Liang, Jianjun Huang, Wei You, Jiachun Li, Yaping Li, Wenchang Shi

Abstract—Mobile game has become a big industry, whose success heavily depends on the game fairness. Recently, a new type of physical cheating instrument, the capacitance-based gamepad (CBG), has been wildly used in popular mobile games. CBG players can obtain an unfairly overwhelming control advantage (e.g., more sensitive clicking and sliding) over benign players. Moreover, as a physical peripheral, CBG is completely transparent to the game application and the underlying system. This makes it inherently immune to existing cheating detection techniques. In this study, by disassembling the CBG device, we find a leverageable physical limitation that the distributions of generated clicking and sliding are more concentrated around a limited area or a boundary respectively. Accordingly, a novel method is proposed to detect the CBG-based cheating. Specifically, to detect the CBG clicking, we employ the entropy to measure the uncertainty of the clicking coordinates; and to detect the CBG sliding, we introduce the convex hull identification algorithm to recognize the potential sliding boundary. We have applied our detection method to four popular mobile games. The evaluation results demonstrate the effectiveness of the proposed method. We believe that the proposed method can be easily adopted by the manufacturers to fight against the CBG-based cheating and protect the game fairness.

Index Terms—Mobile game, game cheating, capacitance-based gamepad, detection, entropy, convex hull.

1 Introduction

With the popularity of smart phones, the mobile game industry has grown exponentially in recent years. According to a report from Newzoo [36], mobile gaming will continue to be the largest segment of the whole games market and reach $70.3 billion following ten years of double-digit growth. Tens of thousands of mobile games have been developed. Some of them have even attracted hundreds of millions of players, e.g., Game for Peace (a.k.a. Player Unknown's Battle Grounds, PUBG) [30].

Obviously, the fairness of a game is vital to its success. However, game cheating techniques can seriously compromise the game fairness and break the game balance, resulting in a decrease of player satisfaction, and even retirement from the game [33]. In practice, the traditional game cheating often requires modifying the game itself or the infrastructure of user devices, e.g., repackaging game applications to introduce cheating code or rooting smart phones to replace critical libraries. Accordingly, some defense techniques have been proposed to check the integrity of the game application [14], [22], [27] or analyze the runtime environment [34].

Recent years, physical peripherals are leveraged to perform sophisticated game cheating. Gamepad is a common type of handheld peripherals, which is used to simulate clicking and sliding actions in the game and provides much more sensitive and agile game control than playing only with human fingers on the smart phone screen. Allowing players to use gamepads will compromise the fairness of a mobile game. For instance, in a shooting game, the normal players who play games just with their fingers are more likely to be defeated by the players equipped with gamepads. To ensure fairness, some game manufacturers have begun to discriminate the gamepad players from normal ones. Tencent [12], the biggest mobile game manufacturer in China, prohibits the use of gamepads in the normal game zones. In some game events, the user account will be blocked if he or she is found to use a gamepad [1].

The early-age gamepads are usually connected to the smart phones as an external device via wire or Bluetooth and require the users to install some assistant applications. To safeguard the game fairness, the mobile game manufacturers can employ various techniques (e.g., detecting suspicious connections and applications) to sensor the existence of such cheating instruments and block them [26].

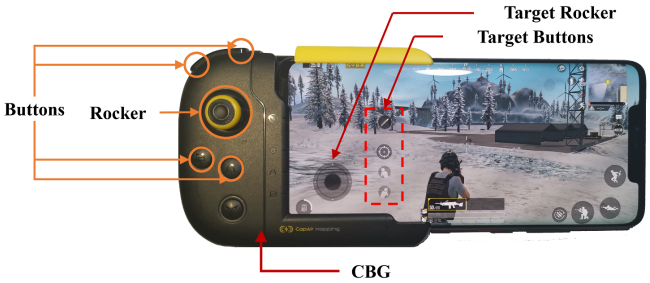


Fig. 1: Using CBG in PUBG.

However, a new generation of the mobile gamepad has emerged recently, which leverages the capacitance screen technique to enhance the control over the game. In this paper, we call it the *capacitance-based gamepad* (CBG). As shown in Figure 1, a CBG device has some buttons and a rocker. A button is used to produce a simulated clicking

S. Bai, B. Liang, J. Huang, W. You, J. Li, Y. Li and W. Shi are with the School of Information, Renmin University of China, Beijing, China; and also with Key Laboratory of DEKE (Renmin University of China), MOE, China. Email: {baishilei, liangb, hjj, youwei, jclee, liyp, wenchang}@ruc.edu.cn.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

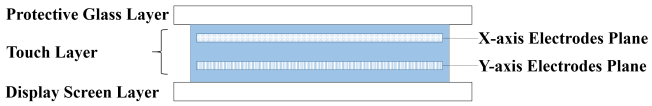


Fig. 2: Layers of the mutual capacitive touch screen.

action at a specific position, and the rocker can generate a simulated sliding action and control its direction and speed. Compared with the players' fingers, the CBG device can give the users an overwhelming micro control advantage over benign players [4]. Unlike the early-age gamepads, the CBG device is immune to existing detection techniques. It does not need any wired or wireless connection to the smart phones, neither does it need installing any assistant applications, rooting the phones, repackaging or modifying the game applications. What the players need is just to plug the phone into the CBG's slot. The game application and the smart phone runtime environment remain unchanged. In such cases, existing cheating detection techniques turn out to be ineffective as no assistance programs or active connections could be found in the phone equipped with a CBG.

Naturally, how to detect the CBG-based game cheating has become an important and urgent problem. In order to design an effective detection method, we first need to understand the working mechanism of CBG. In this study, we physically disassemble the CBG device and conduct some analysis experiments to investigate its principle. We successfully learned that the CBG simulates the clicking and sliding actions by directly transmitting the electric signals to the smart phone screen and monitoring the responses. In this way, CBG can work without any influence on the target game application and its runtime environment. Fortunately, we further found that there is significant difference between the distributions of the touching coordinates generated by the CBG and by human fingers. Due to the physical limitation of the CBG device, the distributions of the CBG clicking and sliding are more concentrated around a limited area or a boundary respectively.

Based on the above observation, we propose a novel method to detect the CBG-based game cheating by analyzing the coordinate sequences of the clicking and sliding actions. Specifically, we leverage the *entropy* [16] to measure the uncertainty of the clicking coordinates. Since the CBG clicking produces a more concentrated distribution (i.e., less entropy) than the finger clicking, we can effectively distinguish them from each other. Besides, the *convex hull identification* technique [13] is introduced to recognize the potential boundary of the sliding coordinates. In practice, the extremum points of a sliding produced by the CBG rocker will be more concentrated around the boundary. As a result, the CBG sliding can be effectively detected by computing the possibility of the extremum points being covered by the identified boundary gap.

The proposed method has been applied to four popular mobile games and evaluated on Android and iOS devices. The experiment results show that our method can effectively detect the CBG clicking and sliding actions in real-world mobile games with very high precision, especially no false

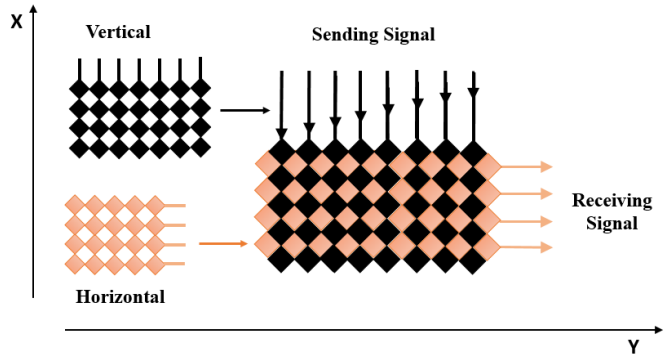


Fig. 3: Structure of the touch layer.

positives or false negatives for the CBG clicking detection.

In summary, this paper makes the following contributions.

- We demystify the principle of the CBG device via physically disassembling it and performing analysis experiments. From our investigation, the researchers and game manufacturers can learn how the CBG simulates clicking and sliding actions in games and develop corresponding defense techniques. To the best of our knowledge, this is the first work that deeply investigates the working principle of the CBG from the viewpoint of detection.
- We propose a novel CBG detection method based on the entropy analysis and convex hull identification. The experiments on four popular real-world mobile games demonstrate that the proposed method can effectively detect the CBG-based cheating. We believe that the manufacturers can easily integrate our method into their mobile games to defeat the CBG-based cheating.

The rest of the paper is organized as follows. Section 2 investigates and presents the principle of the CBG device. Section 3 elaborates our detection method and discusses its robustness. Section 4 evaluates the proposed method on Android and iOS. After discussing the limitations and possible future works in Section 5, we review the related work in Section 6 and conclude the paper in Section 7.

2 The CBG Demystified

In this section, we first provide the background of mutual capacitive touch screen, then analyze the working mechanism of the capacitance-based gamepad and illustrate how it simulates the clicking and sliding, the most commonly used operations in real-world mobile games.

2.1 Mutual Capacitive Touch Screen

Due to its high resolution and multi-recognition capability, mutual capacitance has become the most popular type of touch screens adopted by recent smart phones [10]. For example, since iPhone 6, all iPhones are equipped with mutual capacitive touch screens.

As shown in Figure 2, a mutual capacitive touch screen consists of three layers, i.e., the protective glass layer, the

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

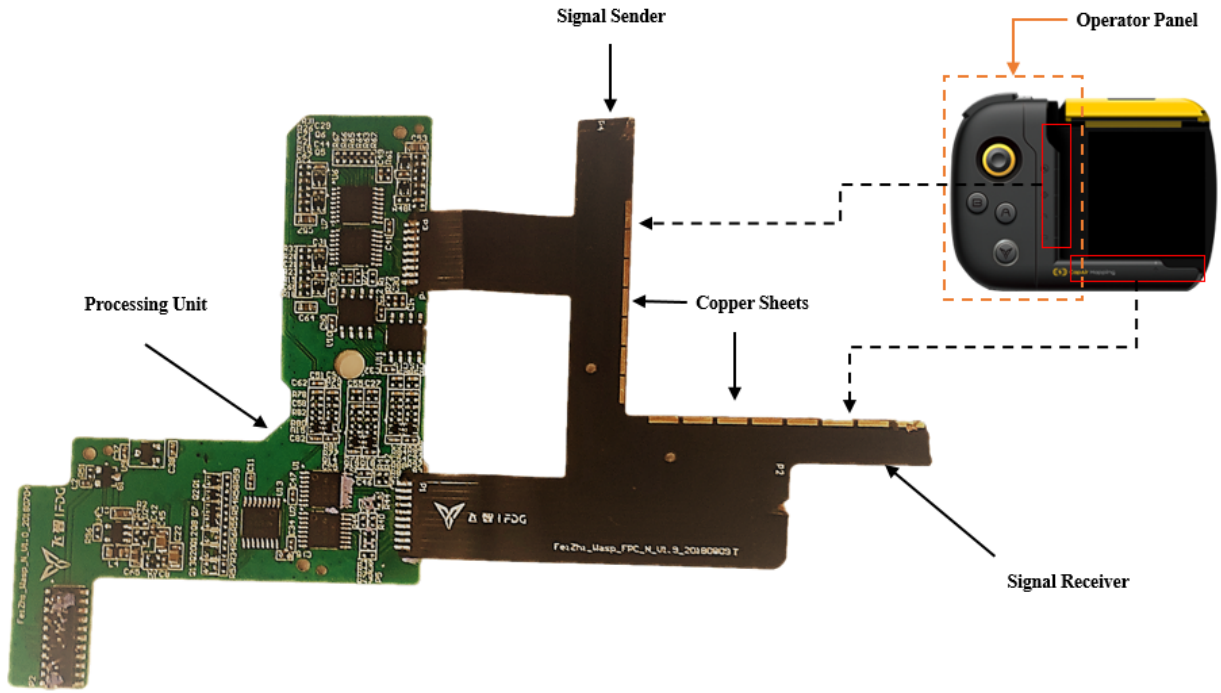


Fig. 4: Disassembling the CBG.

touch layer and the display screen layer, from top to bottom. In particular, the touch layer is responsible for locating the touch points on the screen. Figure 3 zooms in the structure of the touch layer. It is composed of two separate electrode plates, each equipped with parallel lines of electrodes in the horizontal rows and the vertical columns, respectively. The horizontal and vertical electrodes are inter-connected by capacitors, forming a rectangular coordinate system on the screen.

The processing unit of mobile phone (PUMP) sends scanning signals via vertical electrodes in a round-robin fashion, which are received by all horizontal electrodes. When a finger touches a capacitor on the screen, its capacitance will be reduced, which in turn weakens the intensity of the scanning signal. The location of the touch point can be determined by identifying the pair of horizontal and vertical electrodes that exhibits a weaker signal with roll polling. As an instance, assuming that during the period of sending signal from the y-th vertical electrode, the x-th horizontal electrode receives a weaker signal (than other horizontal electrodes), the coordinate of the touch point is determined as (x, y).

2.2 Capacitance-based Gamepad

Clicking and sliding are two elemental operations for mobile games. The CBG device can leverage the capacitance screen technique to perform more sensitive and agile clicking and sliding than the normal players can do with their fingers.

Figure 4 presents the disassembly diagram of a CBG device, which is composed of four components, including the signal sender and receiver, the operator panel and the processing unit. In particular, the signal sender and receiver are equipped with a set of copper sheets to simulate the screen touch. The operator panel contains several buttons for clicking and a rocker for sliding. The processing unit of the CBG (PUG) connects the buttons and the rocker in the operator panel with the copper sheets of the signal sender and receiver.

We present the principle of the CBG in Figure 5. To simulate clicking and sliding on the screen, the CBG converts the pressing of the buttons and the shaking of the rocker to electric signals. During a game play, when the player operates the rocker or the buttons, the corresponding control commands will be sent to PUG and interpreted to determine the coordinates of the touch points on the screen, say (x, y). The PUG will accordingly assign a specific copper sheet on the signal receiver side (responding for the y-th vertical electrode) to monitor the scanning signal. When the target signal is captured, a copper sheet on the signal sender side (responding for the x-th horizontal electrode) will be commanded to send an interfering signal to weaken the capacitance of the corresponding capacitors. As a result, the PUMP is deceived to believe that there is a finger touch on the screen at the coordinate of (x, y).

Since the CBG directly interacts with the screen at the electric signal level, the player only needs to plug the mobile device into the CBG's slot and operates the rocker and the buttons when playing with a CBG, without any wired or wireless connections to the smart phones or any additional applications installed.

We illustrate how the CBG simulates the clicking and sliding below.

Simulation of Clicking. Suppose the player presses a button of the CBG to simulate a click at point $A(X_3, Y_2)$ on the smart phone screen. The PUG will choose the second copper sheet on the signal receiver side (denoted as R_2) and the third copper sheet on the signal sender side (denoted as S_3) for the simulation. As shown in Figure 5, once the scanning

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

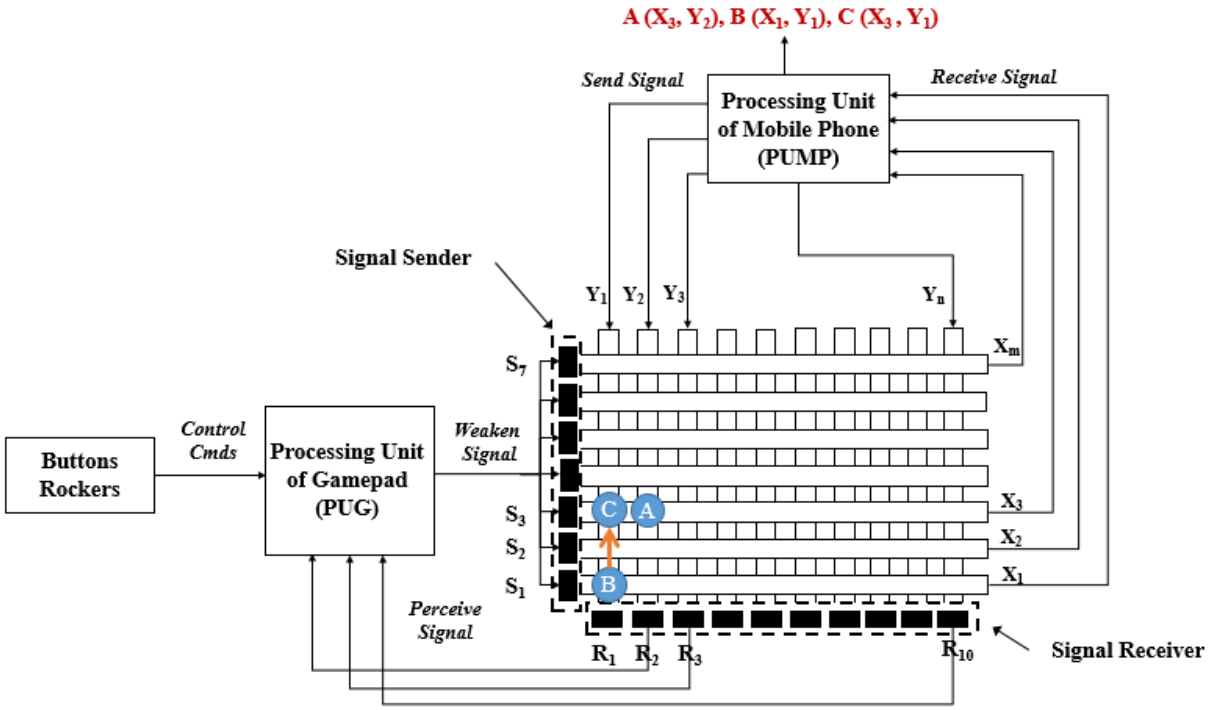


Fig. 5: Principle of the CBG.

signal sent from the PUMP is perceived at R_2, the PUG will emit a weakening signal at S_3 to deceive the PUMP. In other words, when the Y_2-th vertical electrode is scanned by the PUMP, the X_3-th horizontal electrode will be applied with a weakening signal by the PUG. Consequently, a click action on (X_3, Y_2) is simulated.

Ideally, when a button is pressed for multiple times, the corresponding CBG clicking is supposed to hit on the same coordinate on the screen. However, the PUG can emit a weakening signal in the horizontal direction only after it receives the scanning signal in the vertical direction. This fact results in an inevitable scanning deviation in Y-axis. For instance, the PUMP may senses at Y_3 instead of Y_2 for the weakened signal in the above example. In contrast, for a specific target button, the associated signal sender is fixed. As a consequence, multiple CBG clicking points for the same button should be concentrated in the vertical direction (X-coordinates) but relatively discrete in the horizontal direction (Y-coordinates).

Simulation of Sliding. Without loss of generality, we assume the player shakes the rocker in the CBG to simulate a sliding in a vertical direction, from point $B(X_1, Y_1)$ to point $C(X_3, Y_1)$. The R_1 and S_1 copper sheets correspond to the start point, and R_1 and S_3 correspond to the end point. Similarly as done in the clicking simulation, once R_1 perceives the electric signal from the PUMP, copper sheets will send a series of weaken signals in sequence along S_1 to S_3. In practice, when the Y_1-th vertical electrode is scanned by the PUMP, the horizontal electrodes from X_1 to X_3 will be applied a sequence of weakening signals. The sliding from (X_1, Y_1) to (X_3, Y_1) is then simulated.

Because the copper sheets of the signal sender and receiver can only perturb a small portion of the touch screen, the CBG sliding is actually confined within a potential physical boundary. Meanwhile, the rocker is designed very sensitive to motion and the player often shakes the rocker violently for a better performance. As a result, the CBG sliding inevitably approaches or touches the physical boundary frequently. In other words, the extremum points of the CBG sliding will be concentrated around the boundary. Related analysis and examples will be presented in Section 3.2.

3 Approach

In this section, we elaborate our approach to detecting the clicking and sliding simulated by the CBG based on the above investigation. Specifically, we employ the entropy analysis and the convex hull identification to catch the CBG clicking and the CBG sliding respectively.

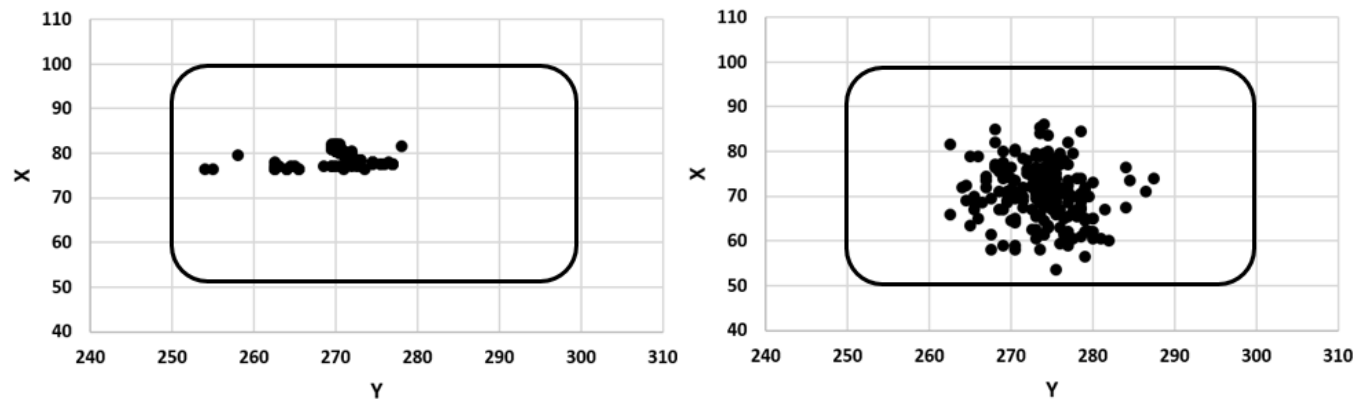
In order to carefully observe the distributions of the clicking and sliding points, we carry out some experiments on a simple demo application, in which we can easily collect the desired data. The demo application has one button for clicking and one rocker for sliding, mimicking the common operation interface of real-world mobile games. Each clicking or sliding event is captured by the application and the point coordinates are recorded.

3.1 Detecting the CBG Clicking

A clicking event on the smart phone screen refers to the procedure of pressing somewhere on the screen and then releasing without movement. It can be triggered by the human fingers, as well as by the CBG. We aim to differentiate the two types of clicking events in mobile applications. As analyzed in Section 2.2, the clicking points produced by the CBG are more concentrated around the target button widget

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers



(a) CBG (b) Fingers

Fig. 6: Distribution of clicking points. The rounded rectangle indicates the effective area that can activate the target button.

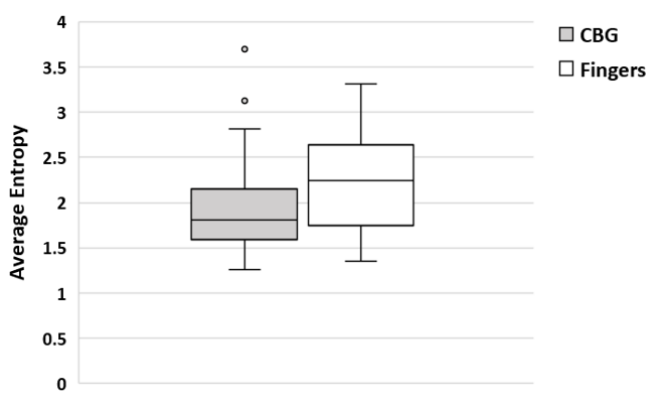


Fig. 7: Entropy of sliding points.

(especially along the vertical direction, i.e., the X-axis), and the finger clicking points tend to scatter more randomly. Namely, We have reason to suppose that the points of the two clicking events possess different location distributions.

To validate the hypothesis, we use the demo application to record 200 clicking events for the CBG and the human fingers respectively. Figure 6 illustrates the distributions of the points. It is clear that the CBG clicking points are more concentrated along the X-axis than the finger clicking points.

Based on the observation, we propose an entropy-based approach to differentiating the two types of clicking events. Entropy [16] is often used to measure the uncertainty in a series of discrete numbers or the degree of ordering of a system. The more orderly a system is, the lower its entropy is. On the opposite, the more chaotic a system is, the higher its entropy is. Given a discrete data sequence $X = \{x_1, x_2 ... x_n\}$, and the frequency distribution of X denoted as $P = \{p_1, p_2 ... p_m\}$, the entropy of X is defined as Equation 1.

$$H(X) = \sum_{i=1}^{m} p_i log \frac{1}{p_i} \tag{1}$$

For the two types of clicking points, the uncertainties of their X-coordinates are obviously different. Naturally, the entropy values of X-coordinates can be immediately leveraged to detect the CBG clicking, whose entropy should be lower than that of the finger clicking.

In the real-world scenarios, the numbers of the clicking points may be different among different rounds of plays. In order to calculate the entropy values under a uniform scale, we divide the collected clicking points of a play round into multiple groups of the same size. The entropy is calculated for the groups individually, and then the average of these entropy values is used to measure the uncertainty of the clicking points in a round of play. To facilitate the discussion, we denote the group size as N, which is a predefined and adjustable value. For the i-th round of play, we denote the number of the collected points as S_i and the number of groups as k_i, thus we have $k_i = S_i/N$. We further denote the vertical coordinate sequence of the i-th round as $\{X_i^j\}$, where j ranges from 1 to k_i. The average entropy of the i-th round (denoted as $\overline{H_i}$) is computed as Equation 2.

$$\overline{H_i} = \frac{\sum_{j=1}^{k_i} H(X_i^j)}{k} \tag{2}$$

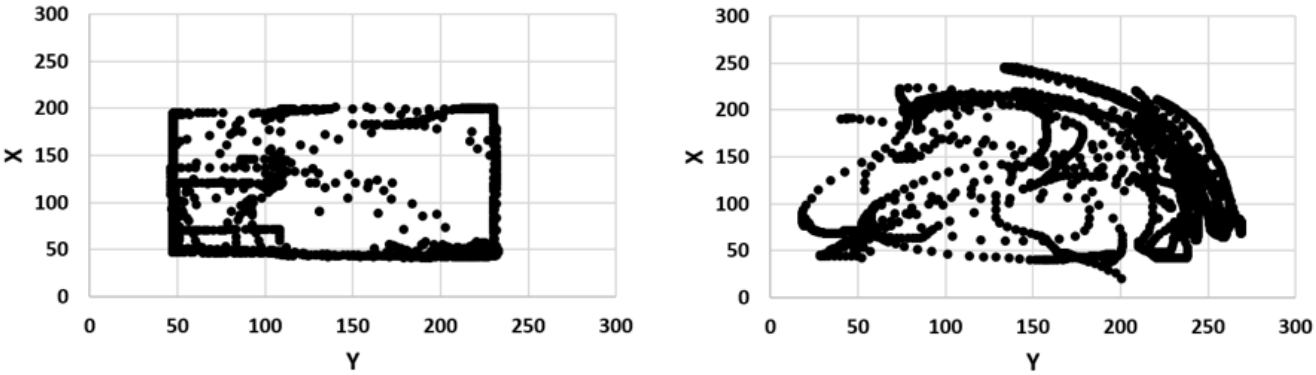
In order to distinguish between the finger clickings and the CGB clickings, we borrow the idea of the support vector machine (SVM) [21] to determine the detection threshold. For a linearly separable two-class data set, there are infinitely many separating hyperplanes. In SVM classifiers, the *best* hyperplane is learned to separate two classes of data, which can maximize the distance to the closest data points (i.e., the margin) from both classes. The data points closest to the separating hyperplane are called as *support vectors*. Theoretically, maximizing the margin means maximizing the generalization performance.

In fact, the detection threshold can be considered as a hyperplane in the one-dimensional space. Generally, the CBG clicking has a lower average entropy than the finger clicking. In other words, the two kinds of events are separable. We can use any value between the maximum average entropy of CBG clicking training events and the minimum one of the finger training events as the threshold.

Naturally, we also want to seek the threshold with the maximum margin to get a robust detector. In this study, the

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers



(a) CBG (b) Fingers

Fig. 8: Distribution of sliding points.

TABLE 1: Average entropy of the demo data.

Type	Points	N=20	N=50	N=100
CBG	200	1.45	1.62	1.71
Finger	200	2.45	2.83	3.01

support vectors can be easily identified from the CBG and finger clicking events by directly checking their the average entropy rather than learned with an optimization algorithm (i.e., the sequential minimal optimization algorithm [29]). The finger clicking event with the minimal average entropy (denoted as Min_{FC}) and the CBG clicking event with the maximal average entropy (denoted as Max_{CC}) are identified as the two support vectors, The mean value of them is taken as the detection threshold to maximize the margin.

Equations 3, 4 and 5 show how Min_{FC}, Max_{CC} and $Threshold\_C$ are calculated, where $\overline{H}_i^F$ and $\overline{H}_i^C$ indicate the average entropy values of the generated points in the i-th play round with the finger and the CBG, respectively.

$$Min_{FC} = Min(\{\overline{H}_i^F\}) \tag{3}$$

$$Max_{CC} = Max(\{\overline{H}_i^C\}) \tag{4}$$

$$Threshold\_C = (Min_{FC} + Max_{CC})/2 \tag{5}$$

We exemplify our approach with the observed data shown in Figure 6. The average entropy values in the different group sizes N (20, 50 and 100) are presented in Table 1. We can see that although the entropy increases as N increases, the CBG clicking always has a much smaller average entropy value than the finger clicking. In other words, using the average entropy can spot the CBG clicking effectively. We further evaluate our approach with a larger set of data for both Android and iOS in Section 4.2.

3.2 Detecting the CBG Sliding

A sliding event describes the process of pressing somewhere on the screen, moving and then releasing. Unlike the clicking, the points of a sliding event often scatter in a much larger screen area, even larger than half of the whole screen. Regardless of whether a sliding event is simulated or not, the distributions of sliding points have a high uncertainty by nature. In other words, it is infeasible to differentiate the finger sliding and the CBG sliding by measuring their uncertainty with entropy.

To illustrate the issue, we record 50 rounds of finger and CBG sliding in PUBG and compute their entropy respectively. From Figure 7, we can see the entropy value ranges of the two type of sliding points overlap heavily. It is clear that directly using the entropy to detect the CBG sliding will lead to unacceptable false positives and false negatives.

Fortunately, due to the physical limitation of the CBG rocker, the simulated sliding is actually confined within a potential boundary, and the CBG sliding is likely to touch the physical boundary frequently. On the contrary, there is not a fixed boundary to restrict the finger sliding. Intuitively, given a group of sliding events, if the majority of their extremum points are located near a potential boundary, we can believe that these events are very possibly produced by a CBG rocker.

To study the distribution of sliding points, we use the demo application to record the sliding points for a few minutes performed by the CBG and the finger respectively. The coordinates of the collected points are illustrated in Figure 8. Obviously, there is an potential boundary with a rectangle shape for the CBG sliding (Figure 8a), and many points gather around the boundary. Compared with the CBG sliding, the points of a finger sliding scatter more randomly in the screen. There is not an obvious boundary around them (Figure 8b).

The key of detecting the CBG sliding is to effectively identify the boundary according to the coordinates of sliding points. Inspired by the study in computational geometry, we leverage the convex hull concept [13] to identify the potential boundary of the sliding events. For a given set of points, their convex hull is the smallest convex set that contains all points. On a two-dimensional plane, the convex hull of a point set is the smallest convex polygon that joins the outermost points. For example, the polygon represented by the red line segments in Figure 9 is the convex hull of the point set $Q = \{q_1, q_2, ..., q_{14}\}$.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

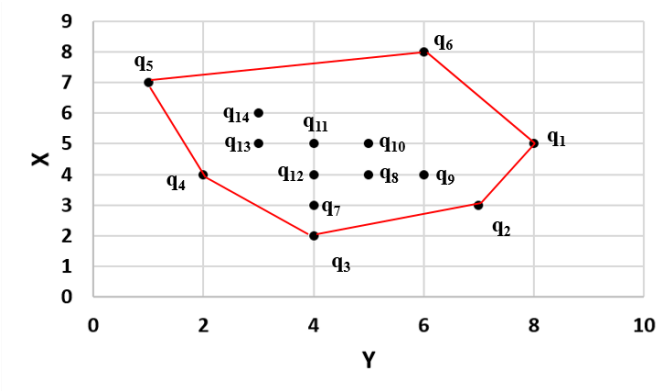


Fig. 9: Example of the convex hull.

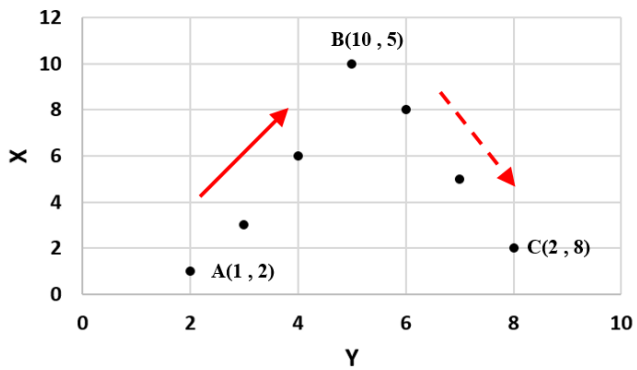


Fig. 10: Example of the inflection point.

Based on the above discussion, we design a technique to detect the CBG sliding via examining the distribution of the extremum points of the sliding point sequence. Given a sliding event, we first find all the extremum points, including inflection points and termination points, from the event points, and then use them to identify the convex hull, i.e., the boundary. Eventually, all the extremum points are checked whether they are located near the convex hull one by one. If it is true for most extremum points, the sliding event will be labelled as a CBG sliding.

Step 1: Identifying Extremum Points. The convex hull is computed based on the extremum points. In this study, the extremum points consist of two types of points. Besides the sliding termination points of the sliding sequences, their inflection points are also used to identify the complete hull. We design a simple but effective search algorithm to discover the inflection points as far as possible. On the two-dimensional screen plane, if the direction of a sliding changes on either X or Y axis at a point of its sliding point sequence, we will mark the point as an inflection point. As shown in Figure 10, a sliding starts from A, passes through B and reaches C. From A to B, all X- and and Y-coordinates increase but the X-coordinates decrease along B to C. Thus, the point B is regarded as an inflection point. Eventually, we get two extremum points for the example, i.e., B and C.

Step 2: Computing the Convex Hull. The convex hull is computed with the identified extremum points, which is

Algorithm 1: IdentifyConvexHull ($points$)

```
1  startPoint ← getFarthestPointInHorizontal (points);
2  lastPoint ← ∅;
3  curPoint ← ∅;
4  nextPoint ← ∅;
5  angles ← [];
6  boundary ← [];
7
8  while curPoint != startPoint do
9
10     if curPoint == ∅ then
11         curPoint ← startPoint;
12
13     boundary ← addPoint (boundary, curPoint);
14     angles ← getAngles (points, lastPoint, curPoint);
15     nextPoint ← getPointWithMinAngle (angles);
16
17     lastPoint ← curPoint;
18     curPoint ← nextPoint;
19
20 output boundary ;
```

built on the smallest convex polygon that joins the outermost points. The widely used Jarvis March algorithm is employed for the task. Algorithm 1 gives a high-level overview of it. Interested readers can refer to [20] for details. The algorithm starts from selecting the farthest point in horizontal, i.e., the one with the maximum Y-coordinate (line 1), and then performs a loop to find out the convex boundaries (lines 8 $\sim$ 18). In each iteration, it calculates for each point in $points$ the angle between the line connecting the point and the current point $curPoint$ and the line connecting $curPoint$ and the last point $lastPoint$ (line 14). The point with the minimal angle is selected as the current point (line 15) and will be added into the boundary points set (line 13) in the next iteration. Eventually, the algorithm outputs the point set of the convex hull, i.e., $boundary$.

Step 3: Detecting the CBG Sliding. The CBG sliding is detected via inspecting the distribution of the extremum points around the convex hull. A convex hull has multiple boundary edges. For a given extremum point, the Euclidean distances [15] to each boundary edge are calculated. Assume a boundary edge connects two points (x_1, y_1) and (x_2, y_2), such a connecting line can be expressed as Equation 6.

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} \tag{6}$$

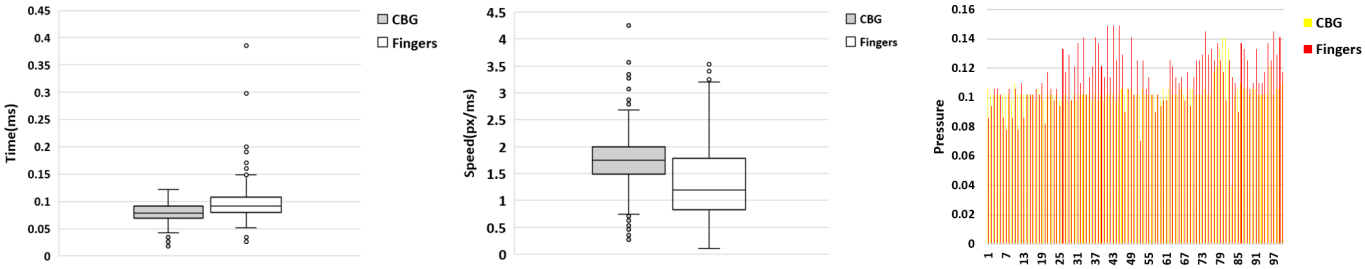
The above equation can be simplified to the form of $ax + by + c = 0$, and the distance d from a given extremum point (x_0, y_0) to the line can be calculated as Equation 7.

$$d = \left| \frac{ax_0 + by_0 + c}{\sqrt{a^2 + b^2}} \right| \tag{7}$$

Finally, the minimum of the distances of the given extremum point to all boundary edges is used as the distance to the convex hull. We set a gap range D and count how many extremum points whose distances to the convex hull are not larger than D. If there is a large enough percentage $P\%$ of extremum points falling in the gap, we will consider the sliding events are produced by a CBG rocker.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers



(a) Duration (b) Speed (c) Pressure

Fig. 11: Other features for CBG detection.

TABLE 2: The statistics of the distances to the convex hull.

Type	Points	D=5px	D=8px	D=10px
CBG	1,675	66.67%	74.60%	79.37%
Fingers	1,953	20.87%	29.57%	41.74%

In a similar way of detecting the CBG clicking, the maximum (Max_{FS}) and the minimum (Min_{CS}) percentage of extremum points within the boundary gaps are computed, and the sliding detection thresholds ($Threshold\_S$) are determined as the mid-range values of them.

We demonstrate our idea with the data in Figure 8, and the results are presented in Table 2. The last three columns summarize the percentages of extremum points whose distances to the convex hull are not larger than 5px, 8px and 10px, respectively. For example, 66.67% (in row 2 and column 3) means that 66.67% of the CBG sliding related extremum points locate within a 5-pixel gap of the convex hull. On the contrary, only 20.87% (in row 3 and column 3) of the finger sliding related extremum points are there. We will further evaluate our approach with a larger set of data on both Android and iOS in Section 4.3.

3.3 Robustness Analysis

Our detection approach exploits the physical limitations of the CBG to detect the simulated clicking and sliding events. Although these limitations are intrinsic to the design of the CBG, they could be mitigated to some extent with the upgrade of the CBG devices in the future. In this subsection, we study the robustness of our detection approach against the potential upgrades of the CBG.

Robustness of CBG Clicking Detection. Recall that the CBG clicking is detected by leveraging the concentration of the clicking points, which is caused by the one-to-one mapping between a copper sheet and a button. That is, CBG uses the same copper sheet for clicking the same button. To de-concentrate the clicking points, an upgraded CBG may try to correlate a single button with multiple copper sheets in the same physical space. Theoretically, the more copper sheets correlated to a single button, the lower the concentration of the generated clicking points.

However, the maximum number of copper sheets is bounded in practice. For example, in our experimental environment, we find that in order to effectively simulate a clicking event, the minimum width of each copper sheet should be more than 1mm. As such, a 4-mm copper sheet (the default setting for a general CBG) can be split into three copper sheets at most. In other words, the number of the copper sheets correlated to a single button can triple at most.

We further perform an entropy analysis to better understand the impact of multiple copper sheets on the concentration of clicking points. Without loss of generality, we assume that every copper sheet in the original CBG has n corresponding copper sheets in the upgraded CBG. The split copper sheets share the same distribution of clicking points with the original copper bench, since they use the same physical material. The frequency of a coordinate generated by each split copper sheet is $\frac{1}{n}$ times of that for the original copper sheet. As such, the entropy for each split copper sheets is $\frac{log\ n}{n}$ times of that for original copper sheet, as calculated by Equation 8. The total entropy of all the split copper sheets is $log\ n$ times of the original copper sheet, as calculated by Equation 9.

$$H_j^{'}(X) = \sum_{i=1}^{m} \frac{p_i}{n} log \frac{n}{p_i} = \frac{log\ n}{n} H(X) \tag{8}$$

$$H^{'}(X) = \sum_{j=1}^{n} H_j^{'}(X) = log\ n\ H(X) \tag{9}$$

At the worst case, where an upgraded CBG splits a copper sheet into three small copper sheets, the entropy is around 1.58 times of the original entropy. As shown in Table 1, under the sampling of 100 data points, the entropy for the upgraded CBG is 2.70 (i.e., 1.71 * 1.58) , which is still much smaller than that of the finger counterpart (i.e., 3.01). That is, the increment of entropy caused by the limited multiplication of copper sheets is not significant enough to confuse the CBG clicking with the finger clicking. In other words, our CBG clicking detection approach keeps effective.

Robustness of CBG Sliding Detection. The detection of sliding relies on the the existence of physical boundary of CBG and the sensitiveness of the rocker, i.e., the CBG sliding is more likely to reach the physical boundary than the finger sliding. To conceal the physical boundary, a CBG may try to deteriorate the sensitiveness of its rocker (so that the physical boundary is hardly reached) or set different boundaries for different instances of the same sliding intent (to vague the physical boundary). Either way would inevitably degrades the user experience, which is fundamentally contrary to the original intention of the CBG. We believe none of these ways will get adopted in practice. That is, the physical boundary is still most likely to be

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

TABLE 3: Real-world mobile games for evaluation.

Game	Downloads	Major Operations
Game for Peace	2,458,770,270	Clicking & Sliding
Ride out Heroes	1,106,880	Clicking & Sliding
Strike of Kings	7,528,803,297	Sliding
Struggle of Snake	9,692,900	Sliding

TABLE 4: Devices for evaluation.

Device	OS	Resolution
iPhone 6	iOS 11	1334 * 750
Huawei Mate 20 X	Android 9.0	2240 * 1080

retained in future upgraded CBG. Hence, our CBG sliding detection approach can keep effective.

3.4 Other Features for Detection

The above subsections demonstrate how to use the distribution of the touch points to identify the CBG clicking/sliding from the finger operations. Below we will discuss the feasibility of using other features (e.g., speed, pressure and duration, etc.) as the metrics for CBG detection.

Using Duration for Detection. The work of [32] uses the duration of keystroke as a feature to distinguish between human players and bots that use API to trigger UI events. The intuition is that API-triggered UI events have relatively shorter keystroke duration than those triggered by human. Unfortunately, such a feature is not suitable for detecting CBG. There is not obvious difference between human players using fingers and the players using CBG. Figure 11a shows the duration distributions of the CBG clicking and the finger clicking for 100 rounds of tests respectively. As we can see, the two distributions look very similar and are difficult to be distinguished.

Using Speed for Detection. Similar with duration, the speed of sliding does not have enough discrimination to be used for CBG detection. Figure 11b shows the speed distributions of the CBG sliding and the finger sliding, each containing the data for 100 tests. We can observe that the two distributions have a large overlap, which makes it difficult to distinguish one from the other.

Using Pressure for Detection. Besides duration and speed, the pressure is another feature of a clicking event. However, the vast majority of Android devices and the early-age iOS devices are not equipped with pressure sensors. They use the decay of electronic signals to approximate the pressure. Such an approximation is very inaccurate, which hinders it from being used for the CBG detection. Figure 11c shows the pressure distributions of the CBG clicking and the finger clicking for 100 tests respectively. As we can see that these two distributions are not easy to be distinguished, making the pressure not a good candidate feature for the CBG detection.

4 Evaluation

We have implemented our detection methods and applied them to four predominant mobile games, including Game for Peace [5], Ride out Heroes [6], Strike of Kings [7] and Struggle of Snake [8]. The major operations of the first two games are clicking and sliding, while the last two games primarily use sliding. Table 3 shows the detailed information of the tested mobile games. All the experiments are conducted on Huawei Mate 20 X (Android 9.0) and iPhone 6 (iOS 11). The device configurations are shown in Table 4. We choose two popular CBG devices, WASP2 [3] and FC [2], as the target gamepads. Due to space limitation, we only present the data of the evaluation on WASP2 in this paper. The evaluation on FC has a very similar result.

TABLE 5: The information of the involved players.

No.	Age	Gender
1	22	male
2	25	male
3	16	male
4	24	male
5	25	female
6	25	female
7	26	female
8	26	female
9	26	female
10	32	male

TABLE 6: Thresholds learned from the training clicking data.

	Android			iOS		
N	Max_{CC}	Min_{FC}	Threshold	Max_{CC}	Min_{FC}	Threshold
20	1.32	2.40	1.86	1.50	2.16	1.83
50	1.50	2.90	2.20	1.60	2.58	2.09
100	1.57	3.30	2.44	1.63	2.77	2.20

4.1 Experimental Setup

Ten players are involved in the experiments. We present in Table 5 their ages, genders and roles from the viewpoint of how their operation data are used. Notice that the players age from 16 to 32, which fits the fact that young people in China are the majority in the mobile game's world [11].

To facilitate the evaluation, we first develop a demo game application to collect the clicking and sliding points produced by the CBG and human fingers. The data are used to train the detection thresholds and test our approach on Android and iOS. On each platform, every player is required to normally play several rounds of clicking and sliding with their fingers and the CBG. In total, we collect 100 rounds of clicking and sliding data on each platform, 50 produced with fingers and 50 with the CBG. Each round contains 200 clicking points and a series of sliding points. Eventually, 20,000 clicking points and 326,303 sliding points are collected.

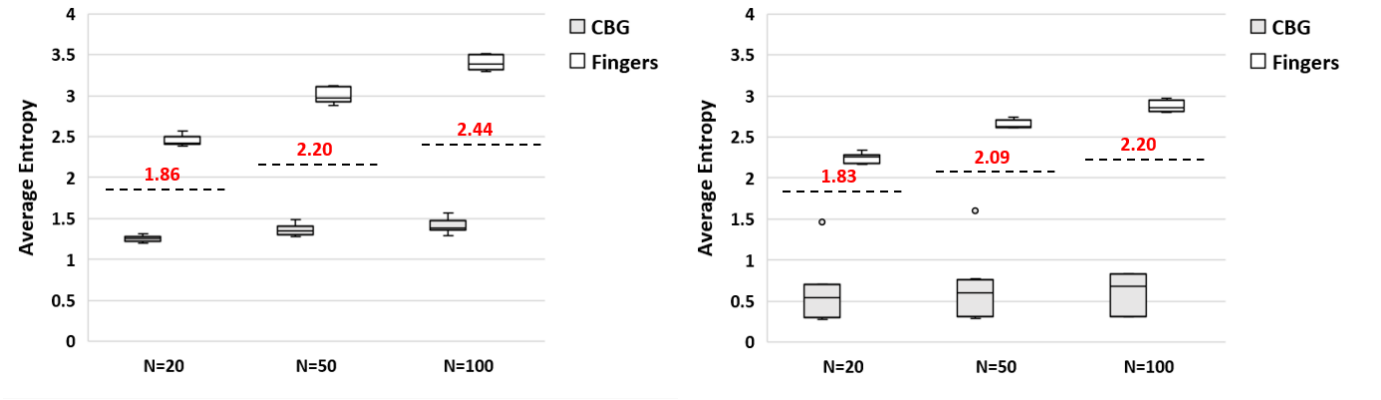
We randomly select 80% of the data produced by 10 players on the demo application as the training set to determine the detection thresholds. The rest 20% of the data are used as the testing set to evaluate our approach with the learned thresholds. They are further applied to detect the CBG operations in the four real-world mobile games.

4.2 CBG Clicking Detection

The entropy thresholds used to identify the CBG clicking are determined with 80 rounds (80%) of the clicking data, which

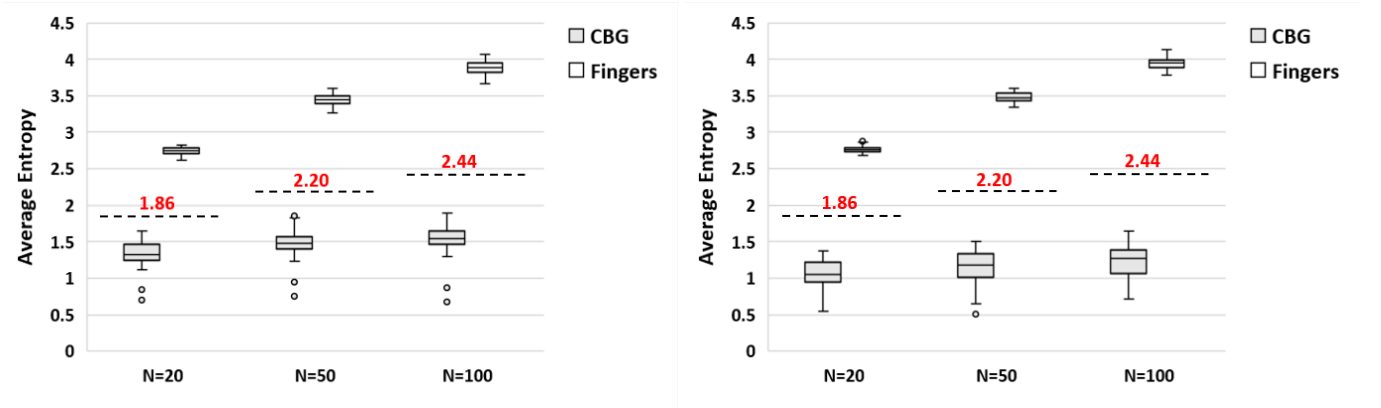
0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers



(a) Android (b) iOS

Fig. 12: Clicking detection on the demo application.



(a) Game for Peace (b) Ride out Heroes

Fig. 13: Clicking detection on the real-world mobile games.

are randomly selected from the collected data on the demo application. Table 6 presents the obtained thresholds under different group size settings (N={20, 50, 100}). Columns 2 ∼ 4 and 5 ∼ 7 show the data on Android and iOS, respectively. For each platform, the mid-ranges of the minimal average entropy of the finger clicking (Min_{FC}) and maximal average entropy of the CBG clicking (Max_{CC}) are calculated and taken as the detection thresholds in the way discussed in Section 3.1.

The learned thresholds are first applied to the rest 20 rounds (20%) of the clicking data. Figure 12 illustrates the distributions of the average entropy values ($\overline{H}_i$) under each setting, and the thresholds are marked as dashed lines. We can clearly see that all the thresholds under different group sizes work perfectly. Both on Android and iOS, we can perfectly tell the CBG clicking from the finger clicking without any false positives or false negatives.

We then examine whether our approach and the learned thresholds can effectively detect the CBG clicking in the real-world mobile games. We choose two popular mobile games, Game for Peace and Ride out Heroes, as the evaluation targets. In the two games, the majority of the player operations are clicking. As illustrated in Figure 13, it is clear that the learned thresholds can be used to effectively differentiate the CBG clicking and the finger clicking in the games.

TABLE 7: CBG clicking detection on two popular games.

N	Thres hold	Game for Peace				Ride out Heroes			
		Max_{CC}	Min_{FC}	FP (%)	FN (%)	Max_{CC}	Min_{FC}	FP (%)	FN (%)
20	1.86	1.64	2.62	0	0	1.37	2.69	0	0
50	2.20	1.85	3.27	0	0	1.50	2.82	0	0
100	2.44	1.89	3.66	0	0	1.64	3.56	0	0

TABLE 8: Thresholds learned from the training sliding data.

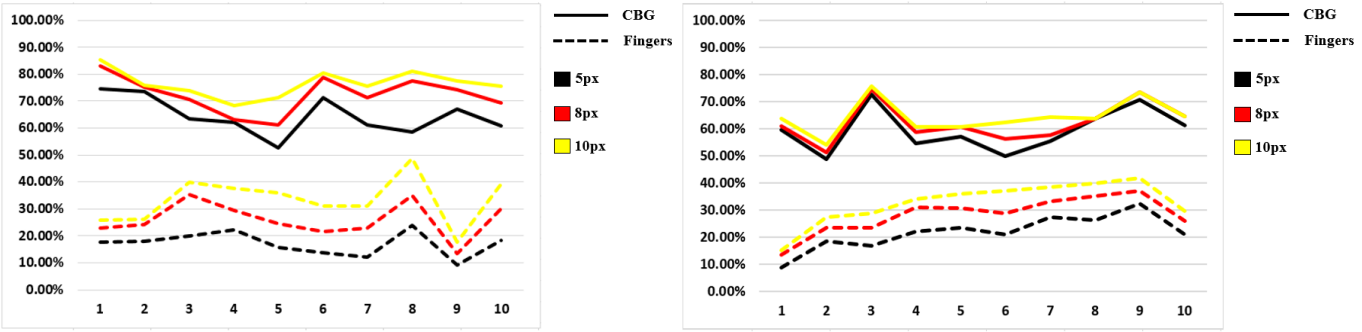
Android				iOS			
D (px)	Min_{CS} (%)	Max_{FS} (%)	Thres (%)	D (px)	Min_{CS} (%)	Max_{FS} (%)	Thres (%)
10	49.26	30.86	40.06	5	48.65	28.28	38.47
20	57.75	45.68	51.72	8	51.35	39.66	45.51
30	62.68	53.09	57.89	10	54.05	46.93	50.49

Table 7 summarizes the CBG clicking detection results. We can see that there are not any false positives and false negatives under different size settings. It is demonstrated that our approach also works perfectly for the two games. Note that the same detection thresholds can be applied to both the demo application and the real-world mobile games.

4.3 CBG Sliding Detection

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers



(a) Android (b) iOS

Fig. 14: Sliding detection on the demo application.

In the same way, the thresholds for detecting the CBG sliding are also learned with randomly selected 80% of the sliding data.

Table 8 gives the thresholds under different gap range settings. Specifically, the gap ranges (D) are set as 10px, 20px and 30px for Android, and 5px, 8px and 10px for iOS. Because our Android device (Huawei Mate 20X) has a higher resolution (2240 * 1080) than iPhone 6 (1334 * 750), we employ larger gap ranges in the experiments on Android. Columns 1 $\sim$ 4 and 5 $\sim$ 8 show the data on Android and iOS, respectively. As discussed in Section 3.2, the maximum (Max_{FS}) and the minimum (Min_{CS}) percentage of points within the boundary gaps are computed, and the sliding detection thresholds ($Threshold\_S$) are determined as the mid-range values of them.

The learned thresholds are first applied to the rest 20% of the sliding data. Figure 14 illustrates the percentages of the extremum points who fall in the boundary gap. The CBG data are presented in solid lines, while the finger data are presented in dashed lines. We can find that there is a potential watershed between the CBG sliding and the finger sliding. Table 9 summarizes the detection results. Only a few false positives occur on iOS when the gap range is set to 8px or 10px. Under other gap range settings, there are not any false positives or false negatives.

All the four real-world games are used to evaluate our sliding detection approach. Figure 15 visualizes the percentages of boundary points of the testing data. We can see that there is an obvious gap between the finger sliding and the CBG sliding. Table 10 summarizes the sliding detection results. It is shown that our approach can successfully detect the CBG sliding with very high accuracy. There are not any false positives, and the false negative rate is almost zero. Except in just one play round in Game for Peace where the gap range is set as 10px, all the CBG sliding are spotted successfully. Notice that we can completely eliminate false negatives by choosing a larger gap range (e.g., 20px), and do not introduce any false positives in the meantime.

4.4 System Overhead

We measure the introduced system overhead of our approaches on Huawei Mate 20 X with 8GB memory, Android 9.0 system and Kirin 980 CPU. We average the time cost of 100 repeated experiments and show the configurations and results in Table 11. For clicking detection, there are 200 rounds and each round contains 500 coordinates. And for sliding detection, there are 160 rounds and each round contains a few minutes sliding points.

TABLE 9: Sliding detection results on the demo application.

D (px)	Thres (%)	Android Min_{CS} (%)	Android Max_{FS} (%)	Android FP (%)	Android FN (%)	D (px)	Thres (%)	iOS Min_{CS} (%)	iOS Max_{FS} (%)	iOS FP (%)	iOS FN (%)
10	40.06	52.54	23.75	0	0	5	38.47	48.65	32.28	0	0
20	51.72	61.02	35.38	0	0	8	45.51	51.35	37.04	0	0
30	57.89	68.42	48.75	0	0	10	50.49	54.05	41.80	0	0

It can be seen that it takes 1.062 seconds for our CBG clicking detection scenario to analyze 100,000 coordinates and on average processing a round of 500 coordinates consumes only 5 milliseconds. The total time for our CBG sliding detection to analyze the 361,590 sliding points (160 rounds) is 4.747 seconds, i.e., the average overhead is 30 milliseconds, higher than the clicking detection as it spends more time to identify the convex hull and compute the distances. We argue that our detection method is practical enough to be adopted in real-world games, since its overhead is negligibly small.

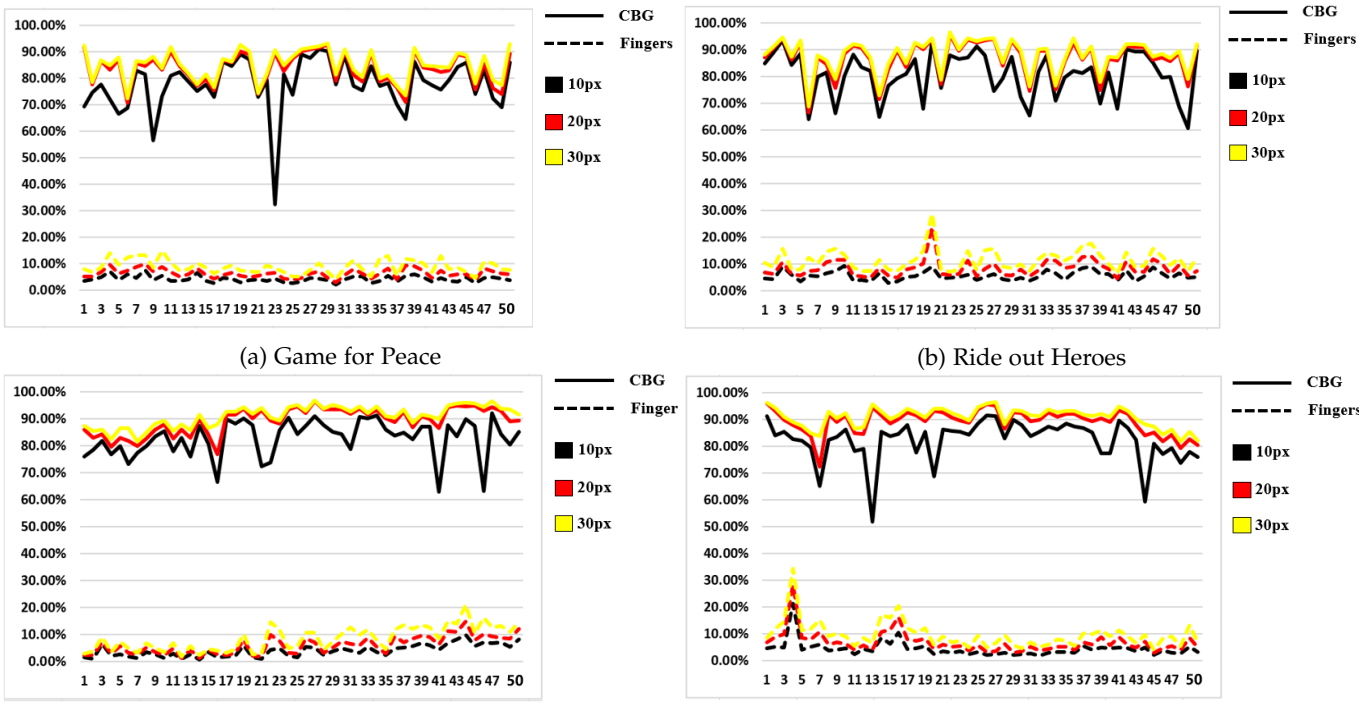
5 Discussion

While the experiments have demonstrated the effectiveness of our approach in detecting CBG, the work has some limitations.

Experimental Limitations. As a baby step towards the detection of the emerging CBG-based game cheating, our experiments are currently conducted on a demo application that simulates typical game environment and four prominent real-world mobile games. The data collection for the demo application is achieved in an in-the-box manner, since we can integrate the data collection code into the demo application. However, we cannot directly instrument the data collection code into the real-world games, since they have complicated validation on the package integrity. Therefore, we adopt an out-of-the-box manner to collect clicking and sliding data for real-world games. Specifically, on Android, we use `adb` to obtain the coordinates of the events. For iOS, it is possible to collect coordinates information of a target application at the system level by jail-breaking the device. In our current evaluation, the demo application is tested on both Android and iOS, while the real-world games are

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers



(a) Game for Peace

(b) Ride out Heroes

(c) Strike of Kings

(d) Struggle of Snake

Fig. 15: Sliding detection on the real-world mobile games.

TABLE 10: Sliding detection results.

D (px)	Threshold (%)	Game for Peace				Ride out Heroes				Strike of Kings				Struggle of Snake			
		Min_{CS} (%)	Max_{FS} (%)	FP (%)	FN (%)	Min_{CS} (%)	Max_{FS} (%)	FP (%)	FN (%)	Min_{CS} (%)	Max_{FS} (%)	FP (%)	FN (%)	Min_{CS} (%)	Max_{FS} (%)	FP (%)	FN (%)
10	40.06	32.23	7.55	0	2	60.75	9.15	0	0	62.87	9.83	0	0	51.79	21.21	0	0
20	51.72	70.67	9.80	0	0	66.53	23.62	0	0	76.93	14.98	0	0	72.48	28.28	0	0
30	57.89	72.30	14.77	0	0	68.86	28.78	0	0	81.49	21.06	0	0	81.97	34.34	0	0

TABLE 11: System overhead of our approach.

Scenario	#Rounds	#Points	Total time (s)	Avg. time (s)
Clicking	200	100,000	1.062	0.005
Sliding	160	361,590	4.747	0.030

tested on Android only. We argue that game developers, who has the access to rebuild the game application, can easily integrate our solution into the game for real-time CBG-based cheating detection. Our detection approach is application-agnostic and has negligible overhead, we believe it can work on most mobile games.

Potential Evasion. Our detection approach relies on the coordinates of points generated by clicking and sliding. The quantity and quality of the collected points will directly affect the effectiveness of our approach. In some cases, the number of the collected points may not be sufficient to support the detection, e.g., in a quickly terminated round of play. However, in the actual detection process, we do not care such short-term round of player, who are harmless losers. After all, the harmless losers will not directly affect the fairness of the game even if they use CBG. We focus on those players with high output. If such players use CBG, they will have the most direct impact on the fairness of the game. Fortunately, in the real game scenes, players with high output will always generate a sufficient number of coordinate points due to frequent operations. Our approaches can detect individual CBG events but they suffer from the cases that the players frequently switch the finger and CBG. We will investigate some advanced clustering techniques to partition the collected coordinates into different groups for detection. Furthermore, we can track the player's historical operation patterns to aid the clustering.

6 Related Work

Game cheating has attracted many researchers' attention and some defense techniques have been proposed. Unfortunately, to the best of our knowledge, there is no existing work that can be used to detect the capacitance-based gamepads. The existing techniques identify game cheating mainly by detecting application integrity, runtime environment, assistant applications and abnormal behaviors.

Detecting application integrity. A straightforward way to achieving game cheating is to instrument additional logic into the original game application. However, this will inevitably compromise the integrity of the target game application. There have been many works that detect the application integrity [23], [24], [37], [38], [39], [40], [41]. In addition, some recent works judeged whether a given two

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

apps are a repackaged pair by executing them and observing their runtime user interface traces. Yue et al. [28] proposed a new concept, i.e., layout group graph (LGG), which is built from UI trances to model the UI behaviors. The LGG can be used to identify potential repackaged application pairs. The repackaged applications are often obfuscated to bypass analysis. Glanz et al. [22] proposed a two-step approach to find obfuscated repackaged applications by leveraging fuzzy hash similarity. The try to modify or delete related code will corrupt the application itself. Although the above techniques have been proven to be effective, detecting CBG is beyond their capabilities because it can work without compromising the integrity of the target game app.

Detecting runtime environment. In order to design an effective cheating mechanism, the adversaries often analyze the target game application in a simulator or a virtual execution environment. A number of existing works have been proposed to detect the runtime environment of the target application. Vidas et al. [35] summarized some features that can be used to distinguish simulators and real devices. Brengel et al. [14] developed a low-level timing-based method to detect the hardware-virtualized systems. Luo et al. [27] proposed a lightweight defense mechanism to prevent an Android application from being launched by the host application as a plugin. Chen et al. [19] detected whether a remote target device is running in a virtual environment by analyzing the TCP packets. Jing et al. [25] proposed a framework that can automatically generate some heuristics to prevent malicious applications from bypassing emulator-based analysis. Google released SafetyNet [9] to provide a set of APIs for checking whether the Android device is in a safe state. Unfortunately, adopting a CBG does not require any special runtime environment or modifying the underlying system. This makes the above existing methods become ineffective.

Detecting assistant application. Traditional gamepads are often connected with the device via Bluetooth or an USB wire. To make the gamepads work, installing an assistant application is also required. The work in [26] proposed a method to detect game plug-ins by checking the existence of suspicious applications. However, installing an assistant app or additional drivers is unnecessary for CBG. The proposed work turns out to be ineffective.

Detecting abnormal behavior. Another way to achieve game cheating is to playing games with bots. The existing works detected game bots by observing the movement paths [17], [18], the operation frequency [33] and the features of the input data [31]. In theory, the method proposed in this paper is also a kind of abnormal behavior detection, with respect to the distribution of the operation points. However, the detection features employed in existing methods rarely appear in the behaviors of CBG operations. From the view of game behaviors, a player equipped with a CBG just act like a game master rather than a bot.

7 Conclusion

In this study, we reveal the principle of the CBG device via disassembling it and performing analysis experiments. We find that the CBG faces some physical limitations, which can be leveraged to detect it. Based on the investigation, a novel detection approach is proposed. We introduce entropy to distinguish the CBG clicking from the finger clicking via measuring the uncertainty of the click coordinates. The CBG sliding is detected by examining the possibility of its extremum points can reach the potential boundary, which is recognized with the convex hull identification algorithm. The detection experiments on iOS and Android demonstrate that the proposed approach can effectively detect the CBG clicking and sliding with very high precision. We believe that our method can be employed by vendors to ensure the fairness of mobile games.

References

[1] Banned for using a controller. https://www.reddit.com/r/PUBGMobile/comments/9mq67f/banned_for_using_a_controller_read_this/.
[2] Flydigi fc gamepad. http://www.flydigi.com/fc.
[3] Flydigi wasp2 gamepad. http://www.flydigi.com/wasp2.
[4] Why is a gamepad worthwhile for games? https://www.gearbest.com/blog/new-gear/why-is-a-gamepad-worthwhile-for-games-5447.
[5] Game for peace. https://gp.qq.com, August, 2019.
[6] Ride out heroes. http://gd.163.com/index.html, August, 2019.
[7] Strike of kings. https://pvp.qq.com, August, 2019.
[8] Struggle of snake. https://sszb.qq.com/, August, 2019.
[9] Checking device compatibility with safetynet. https://developer.android.com/training/safetynet/index.html, Feburary, 2019.
[10] Priciple of mutual capacitive touch screen. https://en.wikipedia.org/wiki/Capacitive_sensing, Feburary, 2019.
[11] The state of mobile games. https://www.appannie.com/cn/insights/market-data/the-state-of-mobile-2019/, May, 2020.
[12] Tencent. https://www.tencent.com/, May, 2020.
[13] C. Bradford Barber and David P Dobkin. The quickhull algorithm for convex hulls. *Acm Transactions on Mathematical Software*, 22(4):469–483, 1996.
[14] Michael Brengel, Michael Backes, and Christian Rossow. Detecting hardware-assisted virtualization. In *International Conference on Detection of Intrusions & Malware*, 2016.
[15] Stephen Cameron. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. *Proc.ieee Int.conf.on Robotics & Automation*, 4(3):3112–3117 vol.4, 1997.
[16] Shannon Ce. The mathematical theory of communication. *M.D. computing : computers in medical practice*, 14(4):306–317, 1963.
[17] Kuan Ta Chen, Andrew Liao, Hsing Kuo Kenneth Pao, and Hao Hua Chu. Game bot detection based on avatar trajectory. In *International Conference on Entertainment Computing*, 2011.
[18] Kuan Ta Chen, Hsing Kuo Kenneth Pao, and Hong Chung Chang. Game bot identification based on manifold learning. In *Acm Sigcomm Workshop on Network & System Support for Games*, 2008.
[19] Xu Chen, Jon Andersen, Z. Morley Mao, Michael Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *IEEE/IFIP International Conference on Dependable Systems & Networks*, 2015.
[20] L. Cinque and C. Di Maggio. A bsp realisation of jarvis' algorithm. *Pattern Recognition Letters*, 22(2):147–155, 2001.
[21] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. 2000.
[22] Leonid Glanz, Sven Amann, Michael Eichberg, Michael Reif, and Mira Mezini. Codematch: obfuscation won't conceal your repackaged app. In *Joint Meeting on Foundations of Software Engineering*, 2017.
[23] Quanlong Guan, Heqing Huang, Weiqi Luo, and Sencun Zhu. *Semantics-Based Repackaging Detection for Mobile Apps*. 2016.
[24] Iakovos Gurulian, Konstantinos Markantonakis, Lorenzo Cavallaro, and Keith Mayes. You can't touch this: Consumer-centric android application repackaging detection. 2016.
[25] Yiming Jing, Ziming Zhao, Gail Joon Ahn, and Hongxin Hu. Morpheus: Automatically generating heuristics to detect android emulators. In *Computer Security Applications Conference*, 2014.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3009374, IEEE Transactions on Computers

[26] Dejian Liu, Hongzhan Chen, and Zhenhua Fang. Method and system for detecting android cheating plug-in feature, December 2015. China Patent No. 105207842.

[27] Tongbo Luo, Cong Zheng, Zhi Xu, and Xin Ouyang. Anti-plugin: Don't let your app play as an android plugin. In *Proceedings of Blackhat Asia*, 2017.

[28] Jun Ma. Repdroid: An automated tool for android application repackaging detection. In *IEEE/ACM International Conference on Program Comprehension*, 2017.

[29] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines.

[30] PUBG CORPORATION. Playerunknown's battlegrounds. https://www.pubg.com/, 2019.

[31] Travis Schluessler, Stephen Goglin, and Erik Johnson. Is a bot at the controls detecting input data attacks. In *Workshop on Network & System Support for Games*, 2007.

[32] Mengjun Xie Steven Gianvecchio, Zhenyu Wu and Haining Wang. Battle of botcraft fighting bots in online games with human observational proofs. In *Acm Conference on Computer and Communications Security*, 2009.

[33] Ruck Thawonmas, Yoshitaka Kashifuji, and Kuan Ta Chen. Detection of mmorpg bots based on behavior analysis. In *International Conference on Advances in Computer Entertainment Technology*, 2008.

[34] Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, pages 447–458, 2014.

[35] Timothy Vidas and Nicolas Christin. *Evading Android runtime analysis via sandbox detection*. 2014.

[36] Tom Wijman. Mobile revenues account for more than 50% of the global games market as it reaches $137.9 billion in 2018. https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/, 2018.

[37] Zhou Wu, Yajin Zhou, Xuxian Jiang, and Ning Peng. Detecting repackaged smartphone applications in third-party android marketplaces. In *Acm Conference on Data & Application Security & Privacy*, 2012.

[38] Shengtao Yue, Qingwei Sun, Jun Ma, Xianping Tao, Chang Xu, and Jian Lu. Regiondroid: A tool for detecting android application repackaging based on runtime ui region features. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018.

[39] Qiang Zeng, Lannan Luo, Zhiyun Qian, Xiaojiang Du, and Zhoujun Li. Resilient decentralized android application repackaging detection using logic bombs. (50-61), 2018.

[40] Fangfang Zhang, Heqing Huang, Sencun Zhu, Dinghao Wu, and Liu Peng. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. 2014.

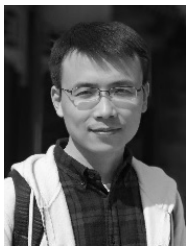
[41] Yury Zhauniarovich, Olga Gadyatskaya, Bruno Crispo, Francesco La Spina, and Ermanno Moser. Fsquadra: Fast detection of repackaged applications. In *Ifip Wg 113 Working Conference on Data & Applications Security & Privacy*, 2014.

ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation of China (NSFC) under grants U1836209 and 61802413, the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China under grant 19XNLG02. Bin Liang is the corresponding author.

Shilei Bai received the B.S. degree in Information Security from Renmin University of China. He is currently a graduate student at School of Information, Renmin University of China. His research interests focus on mobile security.

Bin Liang received the Ph.D. degree in Computer Science from Institute of Software, Chinese Academy of Sciences. He is currently a professor at School of Information, Renmin University of China. His research interests focus on program analysis, vulnerability detection, mobile security and browser security.



Jianjun Huang received the Ph.D. degree in Computer Science from Purdue University. He is currently an assistant professor at School of Information, Renmin University of China. His research interests focus on program analysis, vulnerability detection and mobile security.

Wei You received the Ph.D. degree in Computer Science from School of Information, Renmin University of China. He is currently an assoicate professor at School of Information, Renmin University of China. His research interests focus on program analysis, mobile security and browser security.

Jiachun Li received the B.S. degree in Communication Engineering from University of Science and Technology Beijing. She is currently working towards the Ph.D. degree in Information Security at School of Information, Renmin University of China. Her research interests focus on adversarial machine learning and AI security.

Yaping Li received the Ph.D. degree in Information Engineering from Beijing University of Posts and Telecommunications. She is currently an assistant professor at School of Information, Renmin University of China. Her research interests focus on mobile communication technology and mathematics.

Wenchang Shi received the Ph.D. degree in Computer Science from Institute of Software, Chinese Academy of Sciences. He is currently a professor at School of Information, Renmin University of China. His research interests focus on information security, trusted computing, cloud computing, computer forensics, and operating systems.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Renmin University. Downloaded on July 17,2020 at 13:33:44 UTC from IEEE Xplore. Restrictions apply.