Poster: An Efficient Solution for Detecting UI-Mimicking Android Applications

Jian Mao*, Hanjun Ma *†, Yaoqi Jia†, Zhenkai Liang†, and Xuxian Jiang‡
*School of Electronic and Information Engineering, Beihang University
†School of Computing, National University of Singapore
‡Qihu 360 Technology Co. Ltd.

I. Motivation

In the Android system, apps are managed by centralized markets, such as Google Play. To eliminate malicious apps, the markets actively check apps using scanners, which perform program analysis to detect malicious logic in apps. Many solutions have been developed to further enhance the accuracy of detection [4], [6], [10].

Meanwhile, malicious apps often deceive users via faked UIs. For example, phishing apps [5] mimic UIs of their target apps, such as banking apps, to lure private information from users. As another example, UI-hijacking apps [3] detect internal states of target apps, and replace the fore-ground UI, such as the payment interface of Google Pay, with their faked ones to intercept user inputs. Therefore, UI similarity is an important metric in detecting this type of malicious apps, which is missed by most of the scanners used in app markets.

Several techniques have been developed to detect malicious Android apps based on their resources. For example, ViewDroid [9] detects similarity in Android apps based on the relationship among apps' activities. DroidEagle [8] and ResDroid [7] detect similar UIs in Android apps based on the syntax features of layout files. Though such solutions offer basic techniques in detecting similar UIs, they can be evaded when attackers make simple changes to the content of the resources, without significantly changing the UI's appearance.

To reliably detect similar UIs among Android apps, an effective solution needs to detect UI similarity based on the actual presence of the UI. Specifically, it needs to identify UI features that represent their visual effects. In addition, it needs to efficiently quantify UI similarity based on such features.

II. Solution

In this work, we present an efficient solution to detect Android apps with mimicking UIs. The core component of our approach is the technique to measure the similarity of the UIs. It extracts visual features of an Android apps' UIs, and scores UI similarity using such features.

Android represents UIs using XML-based layout files in apps' APK packages. Our approach first extracts the layout files to compare using `apktool`, which unpacks APK files and extracts the layout files to compare from the `res` directory.

Taking two Android layout files as inputs, our approach evaluates UIs' similarity in two modules: *Visual Feature Extraction* and *Similarity Detection*.

- **Visual Feature Extraction.** This module first converts the layout files into tree structures based on the XML structure of the layout files. It traverses the UI layout trees, propagates the visual features through all the leaf nodes in the layout trees, and extracts the visual features that represent the layout's visual appearance. The features are represented as a set of UI components.
  In our feature set, each UI component has a list of properties. We currently include three main properties: *type*, *position*, *size*. The type property describes the UI component's type as defined by Android, such as *TextView*, *EditText*, etc. The position property describes the absolute position of the component on a standard screen. The size property describes the size of the UI component. The outputs of this module are two UI feature sets, F_1 and F_2.
- **UI Similarity Detection.** This module evaluates the UI similarity rate of two applications. It takes as inputs two UI feature sets and outputs the similarity score of the two applications' UIs.
  The inputs F_1 and F_2 are two sets of UI component feature set. Intuitively, we need to make pair-wise components similarity comparison between two UI feature sets and select the component pairs that is the best match.
  We first measure the similarity of two UI components based on its properties. Given the type, position, and size, we quantify the similarity score of the two components based on their normalized similarity.
  With the pair-wise similarity scores of components in F_1 and F_2, we need to determine the best match of the components in F_1 and F_2, so that the sum of the similarity scores under the component match is maximized. Finding such an optimized map can be solved as an assignment optimization problem [1]. We adopt the Munkres algorithm (K-M algorithm) [2] to solve this problem.
  Our approach computes the UIs' similarity rate by aggregating the selected components' similarity score under the optimized mapping.

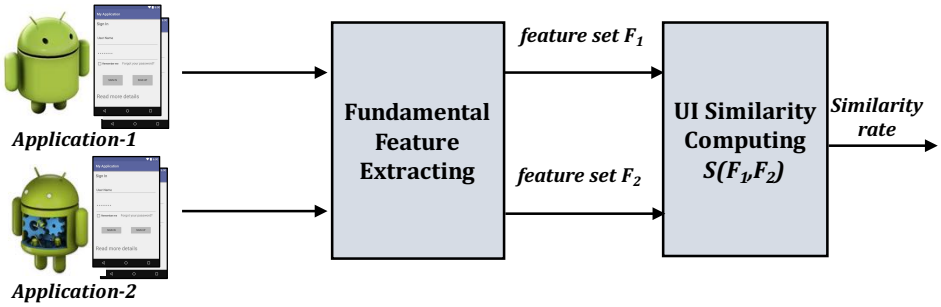

Fig. 1: The core technique of our approach.

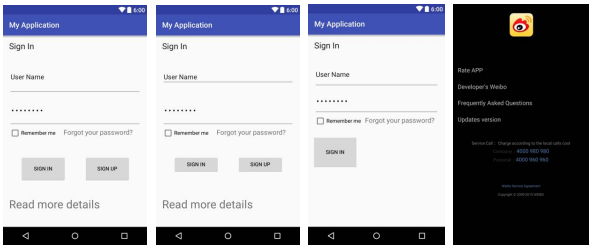

Fig. 2: Synthesized UIs for correctness evaluation. From the left, $UI1$, $UI2$, $UI3$, $UI4$.

TABLE I: Similarity Score of Examples

	$UI1$-$UI2$	$UI1$-$UI3$	$UI1$-$UI4$
Similarity	0.884	0.665	0.374

III. Preliminary Results

We have implemented a prototype of our approach using Python and Shell script.

Correctness. To validate its correctness, we use our tool to detect the similarity of UIs we synthesized. Figure 2 shows four different UIs, namely, $UI1$, $UI2$, $UI3$, $UI4$. $UI1$ is the target UI to be compared to. $UI2$ is slightly different to $UI1$; $UI3$ has more changes in the size and position of controls; $UI4$ is significantly different from $UI1$. The corresponding similarity checking results in Table I demonstrate the effectiveness of our algorithm.

Case Study. We select one pair of repackaged apps as a case study: The original app (MD5: 61b9eb788fb2071db6cf4bac2a8d1e87) is obtained from Google Play, and a repackaged app (MD5: 6024a7bdbcffd5cc9d6ca33624565408) has some mimic UIs as the original one. We tested this pair of apps using our tool and found that they have 6 pairs of mimic UIs whose similarity scores are 0.949, 0.989, 0.949, 0.996, 0.999, 0.805, respectively.

Figure 3 gives the screenshots of one UI pair that layout files are named */res/layout/game_start.xml*. The similarity testing result is 0.989. The original UI is shown in the left part of Figure 3, together with the mimicking UI on the right. They look very similar but the layout files are different. The mimic one doesn't have the *share* button on the right top and the position of the cloud and button are a little bit different too. Our approach analyzes their layouts by visual appearance and outputs a high similarity score.

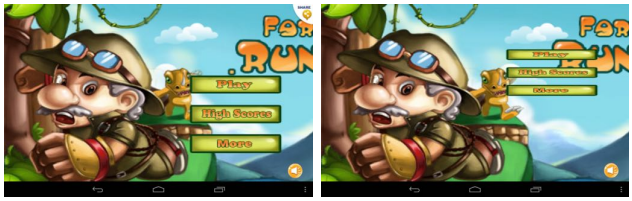

Fig. 3: Examples of mimic UIs

Acknowledgment: This work was supported in part by the National Natural Science Foundation of China (No. 61402029), the National Key Basic Research Program (NKBRP) (973 Program) (No. 2012CB315905), the National Natural Science Foundation of China (No. 61370190), Singapore Ministry of Education under NUS grant R-252-000-539-112.

References

[1] Assignment optimization problem. https://en.wikipedia.org/wiki/Assignment_problem.
[2] Hungarian algorithm. https://en.wikipedia.org/wiki/Hungarian_algorithm.
[3] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. What the app is that? deception and countermeasures in the android user interface. In *IEEE S&P*, 2015.
[4] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for Android. In *SPSM Workshop*, 2011.
[5] Adrienne Porter Felt and David Wagner. Phishing on mobile devices. In *W2SP*, 2011.
[6] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *MobiSys*, 2012.
[7] Yuru Shao, Xiapu Luo, Chenxiong Qian, Pengfei Zhu, and Lei Zhang. Towards a scalable resource-driven approach for detecting repackaged android applications. In *ACSAC*, 2014.
[8] Mingshen Sun, Mengmeng Li, and John Lui. Droideagle: seamless detection of visually similar android apps. In *WiSec*, 2015.
[9] Fangfang Zhang, Heqing Huang, Sencun Zhu, Dinghao Wu, and Peng Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In *WiSec*, 2014.
[10] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *NDSS*, 2012.