# Timing-Based Clone Detection on Android Markets

Yingjun Zhang[1](✉), Kezhen Huang[1], Yuling Liu[1], Kai Chen[2], Liang Huang[1], and Yifeng Lian[1]

[1] Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, People's Republic of China
yjzhang@tca.iscas.ac.cn

[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, People's Republic of China

**Abstract.** With the growth of smartphone users, mobile phone applications increase exponentially. But a lot of apps are cloned. We design a timing-based clone detection method. By choosing several lists of inputs, we can get the corresponding CPU time usage, which composes a CPU time usage tuple. After comparing these tuples, we can find the clone apps. At last, we do some experiments to verify our methods.

**Keywords:** Clone detection · CPU time usage · Smartphone security

## 1 Introduction

With the growth of smartphone users, mobile phone applications increase exponentially. However, according to [1], they find 44,268 cloned apps from 265,359 free Android apps in 17 Android markets. Moreover, malicious users use these cloned apps to gain economic benefits by adding some advertisement or malicious code. So clone detection is important for users and legitimate developers.

Clone [2] means large-scale computer program is duplicated code. Current clone detection techniques mostly analyze the program execution, including control flow and/or data flow. They are mostly based on graph [3, 4], AST [1], token [5] and so on [6]. These techniques are not robust to code obfuscation [7]. In addition, some work focus on analyze binary code instead of source code [8]. Clone detection methods are mostly inefficient.

Birthmarks [9, 10], is an effective way to identify programs and prove ownership, which is a characteristic of an app for clone detection. However, static birthmark can be easily identified by attackers and removed. Researchers designed some methods [11], especially dynamic birthmarks [12], to protect the birthmarks. However, some kinds of the birthmark are overwhelming and easily changed, which make the cloned apps difficult to be detected. Our approach is a kind of dynamic birthmark. Different from previous birthmarks, our birthmark is not running statuses of program variables. In this way, attackers cannot change the birthmarks by obfuscating the exact variables that we use as birthmarks.

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2015
J. Tian et al. (Eds.): SecureComm 2014, Part II, LNICST 153, pp. 375–381, 2015.
DOI: 10.1007/978-3-319-23802-9_29

We designed a new kind of clone detection method based on CPU time usage. After giving each app several lists of inputs, we could get the CPU time usage tuple. Then we compare the tuples. If they are similar, the two apps may be clones at high possibility. At last, we do some experiments to verify it.

In sum, we made several contributions as follows.

- We use CPU time usage as a kind of dynamic birthmark on Android apps, which could be used to detect app clones on Android markets.
- We made several evaluations to verity the effectiveness of this timing-based birthmark. The results show that this kind of watermark is good to detect similar apps.

The rest paper is organized as follows. In Sect. 2, we introduce the motivation and overview of our system. Next, we will introduce our approach and implementation in detail. In Sect. 4, we give some evaluations. Then, we will discuss some problems further. The last section is our conclusion.

# 2 Motivation and Overview

## 2.1 Motivation

As we known, each application (app for short) has its own functionalities. Different apps have different functionalities. Different functionalities will use different numbers of CPU circles with high possibilities. CPU circles are represented as CPU time usages. Thus, we could use the CPU time usage to stand for an app and compare different apps.

With this idea, we made two experiments. We feed the same inputs to two cloned apps (Fig. 1) and two different apps (Fig. 2). The results are:

(1) For the cloned apps, the CPU time usages are almost the same if the inputs are the same (Fig. 1).
(2) For different apps, the CPU time usages are different with high possibility (Fig. 2).

For cloned apps, attackers would not like to change the original functionalities much. They want the cloned to run as stable as possible for long-term revenue. For example, based on previous work [13, 14], attackers only replace some variable names or change the order of statements. So the cloned apps have almost the same functionalities as the original one. That is to say the cloned apps may have almost the same CPU time usage if the inputs are same. Then we want to use CPU status to do our clone detection.

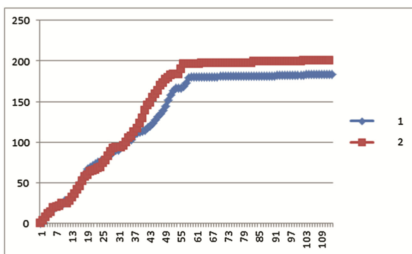


**Fig. 1.** The same app using the same inputs

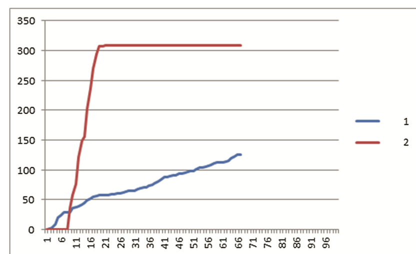


**Fig. 2.** Two apps using the same inputs

One problem in using CPU time usage is that different apps (especially the apps with simple functionalities) may have very similar CPU time usage when feed with the same inputs. To solve this problem, we do not use a single list of inputs. Instead, we use several lists of inputs. Each list includes several inputs. We have the following overview in design.

## 2.2 Overview

We design a system that is based on CPU time to do clone detection. It consists of three main steps as Fig. 3.

- Step 1: For app A:
  First, we use one list of inputs, like Inputs 1, and get the CPU time usage CPU1. In order to monitor the changes, we choose appropriate time interval and record the status of CPU time usage. Then, we choose the appropriate part to analyze. We express it as a vector CPU1 = <c1, c2, …, cn> , the element "ci" ($1 \le i \le n$) is the status of the i-th CPU time usage status.
  Second, we use several other lists of inputs, like Inputs 2…Inputs n, to get the corresponding CPU time usage. As above, we choose the appropriate parts of CPU status for each lists of inputs, which avoid unchangeable CPU status, and express them as a tuple UC = <CPU1, CPU2, …, CPU n>.
- Step 2: For app B or other apps:
  For each app to be analyzed, we do the same thing as those in "For app A". Note that the lists of inputs should be the same as "For app A", and the initial state of the apps should be the same for each testing.
- Step 3: Compare the lists for the two apps.
  After getting the CPU time usage of several apps, we just compare the UC tuples to do clone detection. We mainly do similarity comparison between two UC tuples. By designing a judge algorithm, we get the distance, and then use a threshold to judge whether two apps are cloned or not.

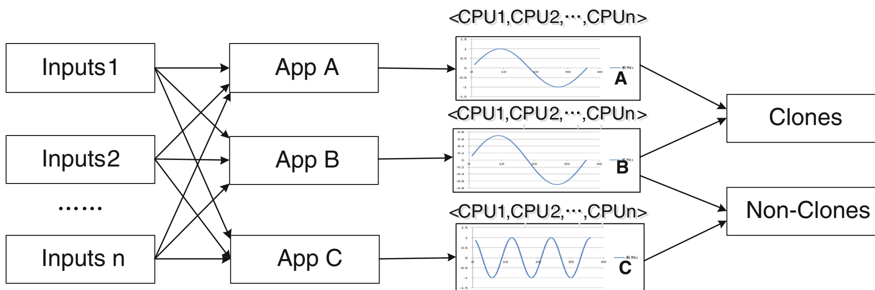


**Fig. 3.** The overview of our approach

We will talk about these steps in detail in the next section.

# 3 Our Approach and Implementation

Based on the overview, we give our detailed design and implementation.

## 3.1 Get the CPU Time Usage

In order to get the correct CPU usage, we have to make sure some pre-conditions as follows are satisfied.

- We need to keep the initial running status as the same for each run. For example, after doing some operations in a testing app, the status of the app is different from the status when the app starts. To make sure the initial statuses are the same, we shut down the app, and open it again. Otherwise, the CPU time usage may be impacted.
- We have to generate the same lists of inputs. Moreover, the inputs should better trigger some complex operations with various types. To generate such inputs, we use the Monkey [15], which is a tool for testing. By emulating a normal user, it generates different kinds of events such as clicks and swipes. In addition, if the seed to the Monkey is the same, the generated inputs are also the same.
- We need to choose the appropriate time interval. With regard to the CPU time usage, if the time interval is long, it may lose some details about CPU changes. If it is short, we have to compare a lot of useless data. To meet this condition, we try to generate different numbers of inputs for each app. For example, the long inputs will trigger more events and make the time interval longer. So we could have different lengths of time intervals.

After we meet the conditions, we could get the CPU time usage. To get the usage, we do not want to insert any code into the apps. If we do so, attackers could find the code and remove all the code, which will effectively undermine our approach. This will also further expose our birthmark.

So we want to get the CPU usage without changing the original apps. As we know, in Linux system, each process has a status file in the system. That is the "/proc/[pid]/stat". In the status, there is a number which indicates the usage of CPU. The number changes when the CPU usage is changed. To read the stat file, we first need to get the pid of the target app. Then we read the stat file every 100 ms.

## 3.2 How to Compare Different Lists

After getting the CPU time usage tuples, we design an algorithm to compare them. The result shows their similarity.

Suppose there are two apps App1 and App2. Using the same lists of inputs, we get the CPU time usage (i.e., birthmark) UC1 = <CPU1, CPU2, …, CPUn> for App1 and UC2 = <CPU1', CPU2', …, CPUn'>. We define the distance between the two apps using following equations.

$$\text{Distance} = \Big(\sum_{i} (|\text{CPUi} - \text{CPUi'}|) / (\text{CPUi} + \text{CPUi'})\Big)/n.$$

After observing the CPU time usage of apps, we find that it increased with time. So we use the CPU time usage after a list of inputs is fully executed by an app. We use a threshold to judge whether two apps are similar or not.

## 4 Evaluation

We do some experiments about clone detection. We first use two cloned apps (“com.gamelin.gjump” and “com.ladty.gjump”) to test. The result is as Fig. 4. We can see that the results of CPU time usage of cloned apps are almost the same. And the Distance = $((|40-41|)/(40+41)+(|19-09|)/(19+20)+(|33-31)/(33+31)+(|18-18|)/(18+18)+(|25-24|)/(25+24))/5=0.018$.

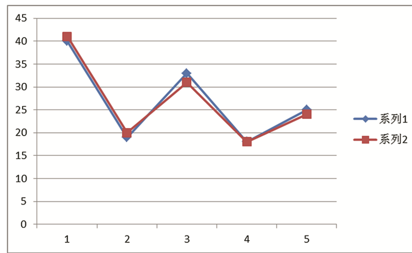


**Fig. 4.** The CPU time usage of two Cloned apps

Then we do experiments on two apps (com.android.calculator2 and com.android.browser). The result is as Fig. 5. We find that the distance is as follows:

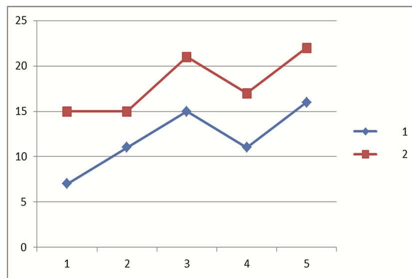


**Fig. 5.** The CPU time usage of two apps not cloned

Distance = $((|7-15|)/(7+15)+(|11-15|)/(11+15)+(|11-17|)/(11+17)+(|16-22|)/(16+22)+(|15-21|)/(15+21))/5=0.211$. So we can choose the threshold as 0.1 and find the cloned ones. In future, we will do more experiments and get a proper threshold.

# 5 Discussion

(1) **How to remove the impact of Internet?**

In android applications, some operations may be closed related with Internet, which may change the CPU time. For example, users have to submit some personal information when register most of applications. In addition, some apps inclusion of ads may also use Internet without notice. But the speed of Internet is impacted by several factors, like Internet speed, operation performance and so on. And our testing result may change a lot in different places.

In order to remove the impact of Internet, we try to choose some inputs, which can avoid operations using Internet. Moreover, we can add some codes to bypass or block internet connections when testing. We will do it in the future.

(2) **How to avoid the impact of obfuscation?**

Attackers often use some obfuscation methods to avoid similarity detection. They can add some useless code, change the execution sequence and so on. These modifications all impact the CPU time usage, which is the basis in our method.

In order to avoid the impact of obfuscation, our inputs lists should cover most of the functions, and avoid repetition. So if some parts are changed, the overall result will be influence little. In addition, if the attackers modify most parts of the apps, the tuple will change a lot. For example, supposing the original vector is CPU = <c1,c2,…,cn> , if attackers add some junk code everywhere in the app, the vector will be CPU' = v * CPU. That means it increase every element of CPU in a linear way. And we will pay more effort on this issue in future.

# 6 Conclusion

In this paper, we design a timing-based clone detection method. First, we have to choose some suitable lists of inputs. Secondly, by using these inputs, we get the corresponding CPU time usage as a tuple. Then, we compare tuples from different apps using the same lists of inputs. From the comparative result, we can find out the cloned apps. Finally, we do some experiments, and the results show the effectiveness of our method.

**Acknowlegements.** The authors would like to thank the anonymous reviewers for their constructive feedback. This material is based upon work supported in part by the National Natural Science Foundation of China under grant no. 61100226 and 61303248, the National High Technology Research and Development Program (863 Program) of China under grant no. SQ2013GX02D01211, and the Natural Science Foundation of Beijing under grant no. 4122085 and 4144089.

## References


1. Gibler, C., Stevens, R., Crussell, J., Chen, H., Zang, H., Choi, H.: Adrob: examining the landscape and impact of android application plagiarism. In: Proceedings of 11th International Conference on Mobile Systems, Applications and Services (2013)
2. Baxter, I., Yahin, A., Moura, L., Anna, M., Bier, L.: Clone Detection using abstract syntax trees. In: Proceedings of International conference on Software Maintenance (1998)
3. Pham, N.H., Nguyen, H.A., Nguyen, T.T.: Complete and accurate clone detection in graph-based methods. In: Proceedings of the 31st International Conference on Software Engineering, pp. 276–286 (2009)
4. Krinke, J.: Identifying similar code with program dependence graphs. In: Proceedings of Eighth Working Conference on Reverse Engineering (2001)
5. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Software Eng. **28**(7), 654–670 (2002)
6. Chen, K., Liu, P., Zhang, Y.: Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In: ICSE (2014)
7. Wang, X., Jhi, Y., Zhu, S., Liu, P.: Behavior based software theft detection. In: CCS (2009)
8. Sæbjørnsen, A., Willcock, J., Panas, T.: Detecting code clones in binary executables. In: ISSTA (2009)
9. Schuler, D., Dallmeier, V., Lindig, C.: A dynamic birthmark for java. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, pp. 274–283 (2009)
10. Wang, X., Jhi, Y.C., Zhu, S., Liu, P.: Detecting software theft via system call based birthmarks. In: ACSAC, pp. 149–158 (2009)
11. Choi, S., Park, H., Lim, H., Han, T.: A static birthmark of binary executables based on API call structure. In: ASIAN, pp. 2–16 (2007)
12. Chan, P.P.F., Hui, L.C.K., Yiu, S.M.: JSBiRTH: dynamic JavaScript birthmark based on the run-time heap. In: IEEE 35th Annual Computer Software and Applications Conference (COMPSAC), pp. 407–412 (2011)
13. Crussell, J., Gibler, C., Chen, H.: Attack of the clones: detecting cloned applications on android markets. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 37–54. Springer, Heidelberg (2012)
14. Zhou, W., Zhou, Y., Jiang, X., Ning, P.: Detecting repackaged smartphone applications in third-party android marketplaces. In: CODASPY, pp. 317–326. ACM (2012)
15. Nyman, N.: Using monkey test tools. Softw. Test. Qual. Eng. (2000)