# A Method of Malicious Application Detection

Xiao Cheng[1,a] , Yan Hui Guo[2,b], Qi Li[3,c]

[1]Xiao Cheng, Beijing Univ Posts & Telecommun, Sch Comp Sci, Beijing 100876, Peoples R China

[2]Yan Hui Guo, Beijing Univ Posts & Telecommun, Sch Comp Sci, Beijing 100876, Peoples R China

[3]Qi Li, Beijing Univ Posts & Telecommun, Sch Comp Sci, Beijing 100876, Peoples R China

[a]chengxiaomlxq@163.com, [b]yhguo@bupt.edu.cn, [c]liqi2001@bupt.edu.cn

**Keywords:** Malware detection; Data analysis optimization; Path optimization; Symbolic computation; Stream tracing

**Abstract.** With the rapid development of broadband wireless access technology and mobile terminal, the mobile internet developed quickly in recent years. However, malicious applications have become one of the key factors threatening the development of mobile internet. In order to protect vital interest of mobile terminal users, mobile malicious applications should be effectively prevented and controlled. This paper analyzes the existence limitation of current mobile application detection technology and uses symbolic execution on the base of stream tracing. Malicious code writers usually hide malicious code execution path, and in some special circumstances to trigger some malicious behaviors. We constraint solve execution routes with sensitive calls, and ultimately solve the specific backstage behaviors and trigger conditions. We did some experiments to evaluate the performance of the proposed method. The experimental results show that our method can work well.

## Introduction

Mobile applications static analysis is a method for feature extraction and behavior analysis through the binary samples. By analyzing and detecting malicious features and sensitive behaviors, security analysts can grasp the malicious samples of harmful way and make the corresponding means to deal with these security risks.

For malicious code detection in feature, an accurate description of the characteristics decides the detection capabilities and efficiency of the methods. In the description of the instruction features, most of the work is carried out for assembly instructions on the x86 platform. For example, Song et al. raised BitBlaze [1] tool on the x86 platform. They proposed a x86 assembly formal description language, Vine Intermediate Language (VIL), in the literature [2], which has played a good role in the program defect analysis. Literature [3] proposed an improved intermediate language according to the characteristics of malicious code, focusing for binary malware behavior analysis on the x86 platform. Shabtai et al. [4] also try to extract and analyze application characteristics on the x86 environment using n-gram model in recent years. But there is rarely targeted characterization work for Dalvik or Java instruction. Felt et al. [5] analyze privileges in Android system. They put privilege as a feature and determine whether malware by Android application privileges. Androguard [6] characterize the function and introduce the entropy and the maximum compression distance NCD [7] to compute the similarity of different functions. But the extracted feature does not delve into the similarity between the instructions.

In the formal description of the code, the existing analysis tools are mostly use Java bytecode or sequence of function calls to describe the behavior of the sample. Literature [8] first proposed a formal description of Java bytecode in the Java Card Virtual Machine, for solving the problem of application defect analysis; On this basis, literature [9, 10] proposed static analysis methods for Dalvik instructions to tracking data streams during program execution. But description method aims at defect detection still can not effectively express the behavior of malicious code.

In malicious code program behavior analysis in the mobile platform, existing tools do not use a valid abstraction intermediate language to describe the behavior of the application. Due to the lack of such a description, making malicious code analysis requires a lot of analysis of personnel involved. The accuracy of manual analysis to a great extent depends on the ability and experience of the analyst. In the environment of uneven capacity and experience of analysis, some samples of malicious behavior tend to be unconsciously ignored.

## System design

**challenge analysis and framework overview.** The main challenge of the mobile application static analysis is two points. The main problem is the executable file static analysis and source code static analysis vary greatly. Analytical work requires complex machine instructions. And program analysis requires accurate and effective formal representation of complex instruction set. Typically, a common instruction set contains hundreds of instructions, such as Java byte code contains 200 instructions, and Dalvik bytecode contains 218 instructions. In each instruction there are many complex syntax, which is a tremendous impact on the analytical work. Secondly, the presence of mobile applications easily decompiled and packaging. General anti-virus software cannot detect some malicious applications after a heavy pack because of the feature library. This difference needs to be considered during feature analysis of the program. Mobile applications feature extraction in the instruction level can avoid malicious application using a simple means to bypass.

Feature comparison algorithm based on instruction sequence can efficiently find the known malicious applications. However, the pace of development of mobile malicious applications far exceeds the signatures base update rate. In the routine analysis, we use instruction characteristics analysis to filter the applications simply and quickly. And on this basis, this paper further studied the semantic analysis of related instructions, which studies the behavior of the sample, thus to discover unknown malicious applications.

Based on the above analysis, this paper studies behavioral analysis methods of mobile malicious applications. And we develop an analytical tool for detecting mobile platforms malicious applications.

In this paper, we use symbolic computation. The main purpose is to calculate the trigger conditions of characteristics sensitive behavior in mobile applications. Thus we do not need to traverse the entire control flow. So before the symbolic computation, control flow graph inside function can be preprocessed, greatly reducing the number of branches in symbolic computation and avoiding the path explosion problem in symbolic calculation process.

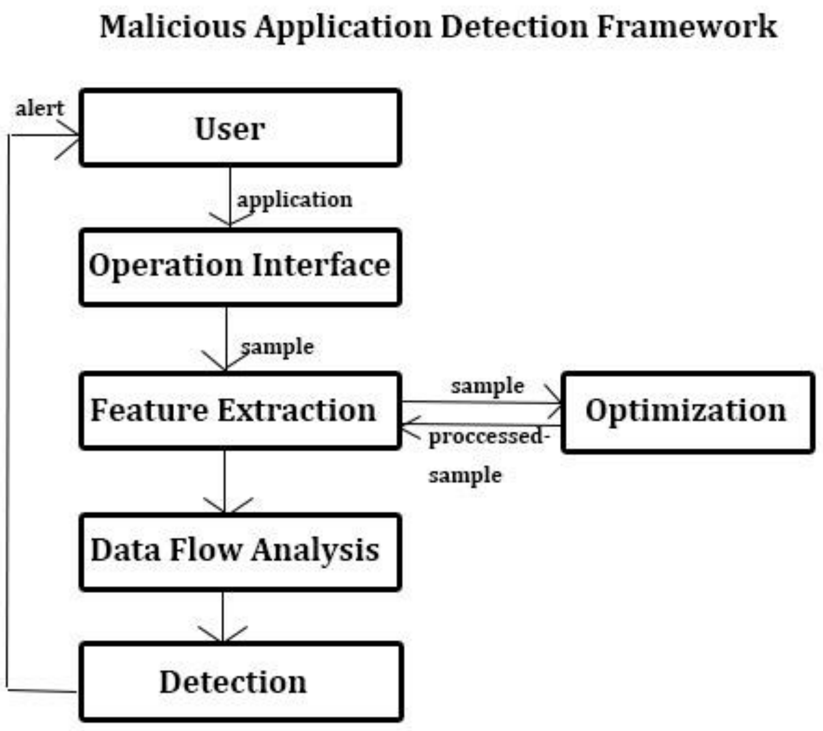

Figure. 1 System Framework

**Path optimization based on data flow analysis.** Path extracting is unable to clear the sensitive call parameters and path trigger condition. To do this, on the basis, we need to trace the data flow and control flow of the path, thinking symbolic computation, to constraint solve path trigger condition.

In this paper, the main idea of the optimization for the control flow inside a function is a combination of data flow analysis and calculation of unrelated branches of sensitive data to achieve optimizes computation references to Reaching Definitions.

Reaching Definitions is a method of data flow analysis, which requires sloving equations composed by in[Bn], out[Bn], gen[Bn], and kill[Bn]. The purpose is to find the relationship between variables and statements in a control flow graph.

Table 1 Definition in Reaching Definitions

| Symbol | Meaning |
|---|---|
| Bn | a given basic block in the control flow graph; |
| in[Bn] | set of definitions that come before Bn; |
| out[Bn] | set of definitions coming out of Bn; |
| gen[Bn] | set of new definitions generated within Bn; |
| kill[Bn] | set of definitions whose variable is killed by Bn. |

In Reaching Definitions, gen[Bn] and kill[Bn] can be obtained directly by analyzing the control flow graph of basic blocks. in[Bn] and out[Bn] can be calculated by the following formulas:

$$out[B_n] = gen[B_n] \cup (in[B_n] - kill[B_n]) \qquad (1)$$

$$in[B_n] = \bigcup_{p \in pred(n)} out[B_p] \qquad (2)$$

Bp is the set of Bn's predecessors. Considering there may be circulating in the control flow graph, in[Bn] and out[Bn] may need for iterative calculations until it reaches a fixed point, ie until steady state.

On the basis of Reaching Definitions, according to the data constraint required traced, we can distinguish the unrelated branches in control flow graph $G_c =< N, E, start, end >$, so as to achieve the purpose of optimization.

Data dependencies can be divided into the following:

Table 2 Data dependencies in Reaching Definitions

| Symbol | Detail |
|---|---|
| $S_1 \delta^f S_2$ | flow dependence |
| $S_1 \delta^a S_2$ | anti dependence |
| $S_1 \delta^o S_2$ | output dependence |
| $S_1 \delta^c S_2$ | control dependence |

Algorithm2-2：Path Preprocessing Algorithm

Input: control flow graph $G_c =< N, E, start, end >$, $gen[B_n]$, $kill[B_n]$, set of parameters passed in by current function $S_{in}$, data constraint ￡, constraint- containing node $B_￡$

Output: reachable path sequence

```
1.  for each block $B_n$ in $G_c$ do
2.          $in[B_n] = out[B_n] = \varnothing$
3.    $in[B_{start}] = S_{in}$
4.  do
5.        for each block $B_n$ in $G_c$ do
6.              $in[B_n] = \bigcup_{p \in pred(n)} out[B_p]$
7.              $out[B_n] = gen[B_n] \cup (in[B_n] - kill[B_n])$
8.  while no more changes to any of $out[B_n]$
9.
10. do
11.       for each parent block $B_\Delta$ of $B_￡$ do
12.             if $(B_\Delta !\delta^f ￡)$ && $(B_\Delta !\delta^c ￡)$
13.                   remove $B_\Delta$ from $G_c$
14.             else
15.                   for each $stmt$ in $B_\Delta$ do
16.                         if $stmt\ \delta^f$ ￡
17.                               update ￡
18.         $B_￡ = B_\Delta$
19. while $B_￡ \neq B_{start}$
```

Data flow optimization algorithm is consist of seven steps:

step 1: Initialize in[Bn], out[Bn], gen[Bn], and kill[Bn] in Reaching Definition. The initialization of in[Bn] needs to consider two cases:

$$in[B_n] = \begin{cases} \{p_1,...,p_n\} & n = start \\ \varnothing & otherwise \end{cases} \qquad (3)$$

$p_1,...,p_n$ are passed in parameters of the function, $out[B_n] = \varnothing$.

step 2: Iterative calculation to get all the basic blocks in[Bn] and out[Bn].

step 3: Determine the initial basic block $B_{\pounds}$ and data constraint $\pounds$.

step 4: Traverse the control flow graph in inverted order starting from $B_{\pounds}$. Calculate all of the data dependencies between $B_{\Delta}$ and its parent node $B_{\Delta}$. Remove the basic block if $B_{\Delta}$ and $B_{\Delta}$ are irrelevant.

step 5: If $B_{\Delta}$ and $B_{\Delta}$ are relevant, analyze data dependencies ci=exp.

step 6: If $B_{\Delta} \neq B_{start}$, $B_{\pounds} = B_{\Delta}$ and return to step 4.

step 7: Generate new control flow graph $G_c{}'$. Save the reachable path sequence.

## Experiment

As shown in Fig.2, fig(a) shows the control flow graph before using the Data flow optimization algorithm, while fig(b) shows the optimizated control flow graph after using the proposed algorithm.

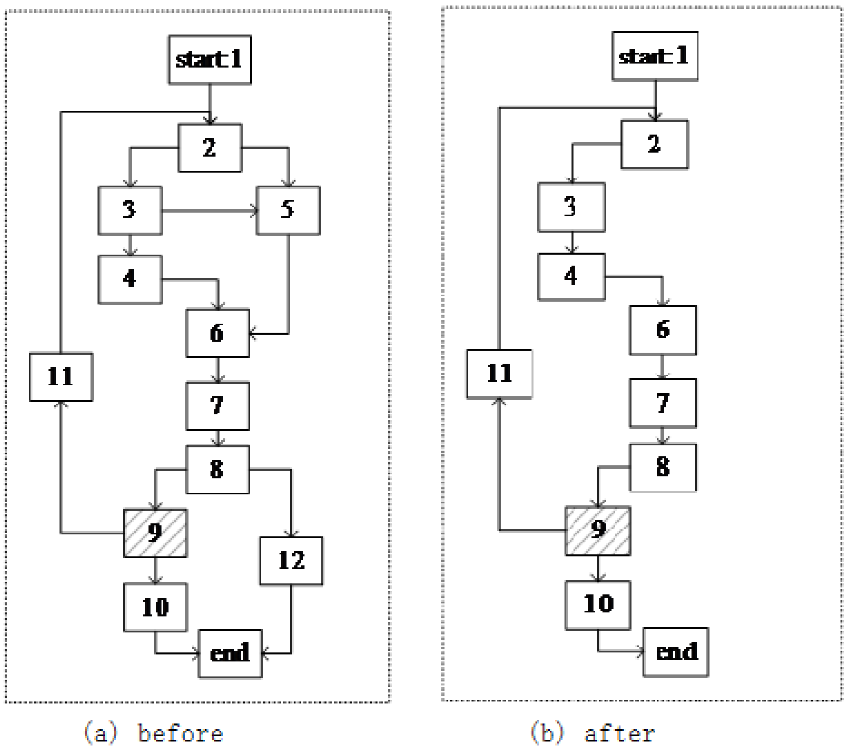

Figure. 2 Comparison of control flow before and after the proposed algorithm

Fig.3 shows the function relationships of a sample mobile application. As it shows, by using the proposed algorithm, mostly unrelated paths can be discarded, which radically reduce the amount of calculation.

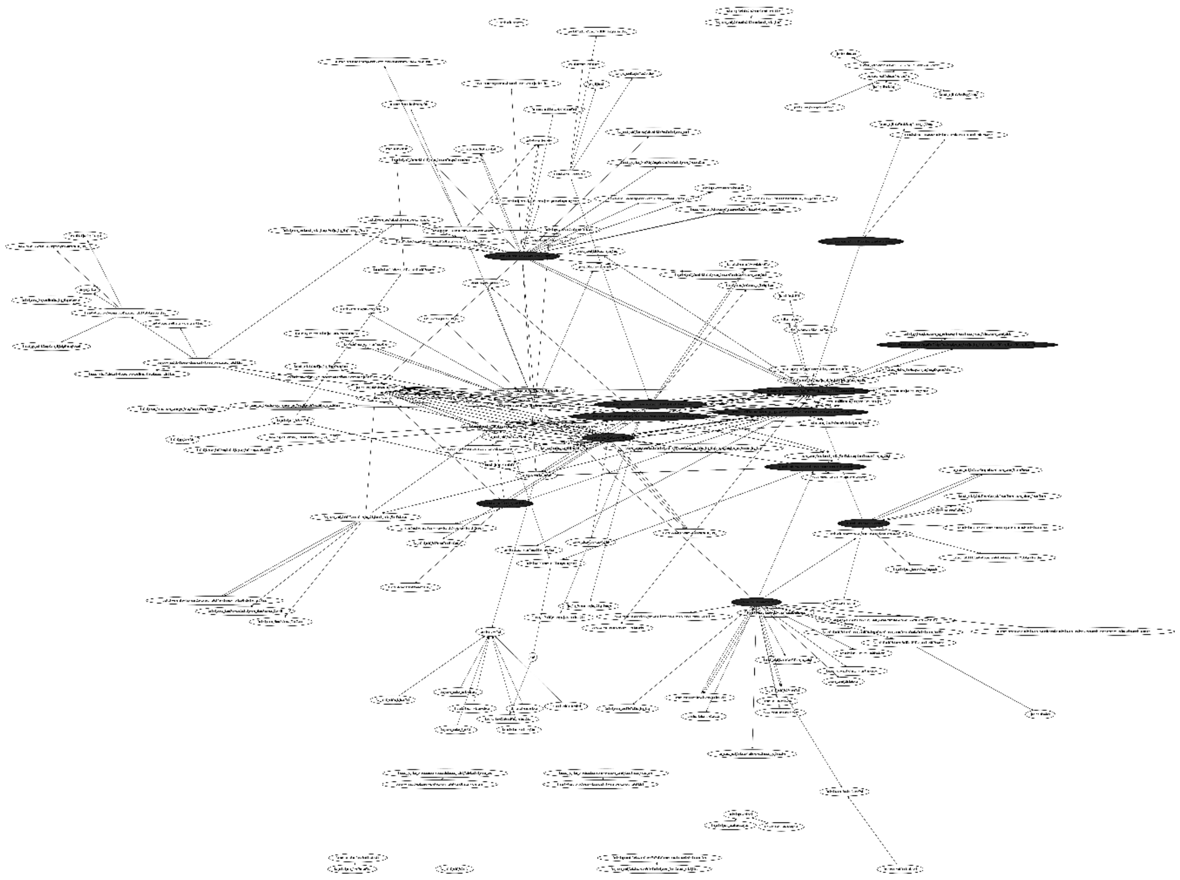

Figure. 3 function relationships of a sample mobile application

## Conclusion

This paper proposed an analytical tool for mobile platforms malicious applications. We proposed a malicious code detection method. The method extract application feature or application behavior description from a known malicious applications, calculate the path constraint condition on the basis of Reaching Definitions and extract sensitive behavior path in applications. Thus we can analyze the description characteristics and sensitive behavior of mobile platforms to detect malicious applications by characteristics and behavior. Finally, experiments show our proposed method can effectively analyze malicious application characteristics, and can effectively detect malicious behavior execution paths in unknown malicious application.

## Acknowledgements

This work is supported by National Natural Science Foundation of ChinaProject(61302087, 61401038).

## References

[1] D. Song, D. Brumley, H. Yin and et al. BitBlaze: A New Approach to Computer Security via Binary Analysis//Sekar R, Pujari A. Springer Berlin Heidelberg, 2008:1-25.

[2] D. Brumley. Analysis and defense of vulnerabilities in binary code. ProQuest, 2008.

[3] J.X. Zhong. Key technologies of malware behavior binary analysis. Beijing University of Posts and Telecommunications, 2012.(In Chinese)

[4] A. Shabtai, R. Moskovitch, C. Feher and et al. Detecting unknown malicious code by applying

classification techniques on OpCode patterns. Security Informatics, 2012,1(1):1-22.

[5] A.P. Felt, E. Chin, S. Hanna and et al. Android permissions demystified: Proc of the 18th ACM Conf on Computer and Communications Security, New York, 2011[C]. ACM.

[6] Desnos A. Androguard: Reverse engineering, malware and goodware analysis of Android applications ... and more (ninja !)[CP/OL]. http://code.google.com/p/androguard/.

[7] R. Cilibrasi and P.M.B. Vitanyi. Clustering by compression. Information Theory, IEEE Trans on, 2005,51(4):1523-1545.

[8] S.I.A. Operational semantics of the java card virtual machine. The Journal of Logic and Algebraic Programming, 2004,58(1):3-25.

[9] E.R. Wognsen and H.S. Karlsen. Study, Formalization, and Analysis of Dalvik Bytecode[G].

[10] E.R. Wognsen and H.S.N.Karlsen. Static Analysis of Dalvik Bytecode and Reection in Android[G].