ApakoHa - an automated tool using dynamic taint analysis for android security focusing on sensitive data

Outline and Topic Proposal

Kaiyuan Xu, Yiwei Yang, Zhe Ye, Longwen Zhang

December 22, 2019

Android security has attracted much research attention from both academy and industry recently. Dynamic Taint Analysis(DTA) is a classic analysis to detect information flow problems and it has been widely adopted to detect private data leaks in Android applications. In this project, we will automate the process of taint analysis and provide more detailed information about the dataflow on both the source code and the byte-code. Thus providing a way for Maple IR to continue to compile.

1 Scope of Work - 4 Questions

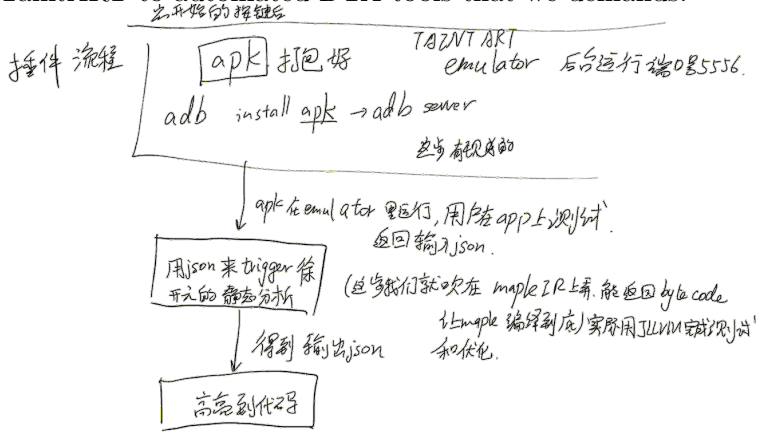
In this section the essence of the proposed work is described by answering four key questions.

What is the problem you want to address in your work? People can't perform the DTA on their apps on the ART runtime fast. Nowadays, DTA consumes time to set up environments and may receive invalid results.

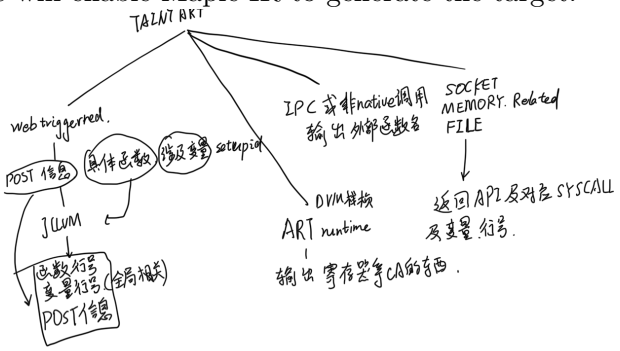
Why is it a problem? Generally, researchers detect privacy leaks and analyze related data flows. One is static analysis, that is, decompiling a given app, and performing static stain analysis or symbolic execution on the obtained code. , Such as FlowDroid, the advantage is that it can widely obtain the trend of private data, but it does not include runtime information. When app developers use technologies such as java reflection mechanism or code encryption, static analysis basically cannot handle this, and The overhead is large, and it takes tens of times more time of compiling time to complete. The second is dynamic analysis, modifying the system or system library, or instrumenting the app, so that the app at runtime

can be monitored. An important task of using dynamic stain analysis technology to detect the information flow of the Android system is TaintDroid. However, TaintDroid is based on the Dalvik virtual machine for taint propagation, and currently only supports Android 4.3 at the highest.

What is the solution you developed in your work? We proposed to adapt the TaintART to automated DTA tools that we demands.



The source code is transferred to the qemu x86 64 virtual machine. Coverted to ART Runtime executable by dex2oat implemented with llvm. In this step, TaintART performs a spot instrumentation operation. At runtime, TaintART uses the general-purpose register R5 to store the tags of the variables in the register, R1 to 3 passes the parameters between the methods, and R12 cooperates with the transfer of the internal operation of the register. A specific area is opened on the heap to store the stain information, so that the stain propagation logic can be detected To information leakage or potentially malicious behavior. After returning to the host, it will use JLLVM Pass (this step can be replaced with Maple IR phase) to perform static control flow analysis on the stain information and corresponding functions and APIs, and finally return it to the developer. The developer can modify the source code or use Pass to improve development efficiency. The final bytecode will enable Maple IR to generate the target.



As for the information of different source(like web, IPC, socket or memory and files), we adopt different methods to deal with.

Why is it a solution? The reason we use x86 64 virtual machine is to raise the speed in case of cross ISA compilation. The reason to use static analysis is to get a more precise information of the control flow graph which is the shortage of DTA. We will apply the benchmark and a demo to show that our solution is faster and more focused privacy.

2 Preliminary Table of Contents

In this section the table of contents for the proposed work is described.

1. **Introduction** How android information privacy security evolves
   a) **from DVM to ART**
   b) **from STA to DTA**
2. **The basics of ApakoHa principle** How The steps go
   a) **how DTA is realized in TaintART**
   b) **how to second-develop TaintART**
   c) **JLLVM pass**
   d) **the combination of STA and DTA**
3. **Case Study** How our tools preform in real test.
   a) **simplified TouTiao**
      i. **Benchmark compared with state of art of android privacy detection tools** comparisan with a JIT-based java STA
      ii. **Benchmark with TaintDroid** Will TaintART get the bugs as TaintDroid do
      iii. **The app developers feedback** will the developer use the plugin
4. **Division of Duty** insert brief description
   a) **Kaiyuan XU** implement the JLLVM pass.
   b) **Zhe YE** implement the VScode plugin
   c) **Longwen ZHNAG** second-develop TaintART
   d) **Yiwei YANG** Propose the Idea and do TainART realization

3 Relevant Related Work

In this section, identified related work is described.

[Sun:2016:TPM:2976749.2978343] A Practical Multi-level Information-Flow Tracking System for Android RunTime

[WCNCW.2019.8902627] Android Malware Detection Based on System Calls Analysis and CNN Classification

[SPW.2018.00031] A Dynamic Taint Analysis Tool for Android App Forensics

[NGMAST.2014.23] Android Malware Detection Using Parallel Machine Learning Classifiers

[IMIS.2014.28] PasDroid: Real-Time Security Enhancement for Android