© 2015 IJEDR | Volume 3, Issue 4 | ISSN: 2321-9939

Various Types of Tool for Instrumentation of Android Application

[1]Bhusadiya Pravin, [2]Prof. B.V.Bhuddhadev
[1]M.E Student, [2]Professor
Department of Information Technology,
Shantilal Shah Engineering College, Bhavnagar, Gujarat, India

***Abstract* - There are various types of tool for instrumenting the android application. PIN is a dynamic binary instrumentation engine or framework. It can be used for several purposes, but mostly for program analysis (memory allocation analysis, error detection, performance profiling, etc...) and for architectural study (processor and cache simulation, trace collection, etc…). There are three type of tools (1) Pintool (2) Soot (3) Drozer. Pintool is mostly use for Program analysis. Drozer tool is used for security assessment and soot tool are used for optimization. This paper analyze these tool and pin tool is best for instrumenting the android application**

***Keyword* - Instrumentation; Pin; Pintool; Virtual Machine; Runtime; Soot; Drozer**

I. INTRODUCTION

Instrumentation is a simple technique for inserting any extra line of code in to an application to observe its behavior. It can be performed at various stages – inside the source code, at compile time, post link time, or even at run time.

Source Code Instrumentation is a way to instrument source programs and **Binary Instrumentation** is to instrument binary executables directly.

Static binary instrumentation (SBI) occurs before the program is run phase, a phase in which we can rewrite executable code or object code. Static binary instrumentation was first used by ATOM], and then followed by others such as Etch, EEL, and Morph. whereas dynamic binary instrumentation (DBI) is done at run time. The analysis code can be injected by a program into the client program or process, or by any external process. If the client program uses any dynamically linked code, then the analysis code must be added after the dynamic linker has finished its job.

Pin is a framework system that performs run-time binary instrumentation of Windows, Linux or Android applications. Dynamic binary instrumentation has following two distinct advantages. Firstly, we do not require the client program to be prepared in any particular or fixed way, which makes it very convenient and easy for users. And second, it naturally covers all code, instrumenting all code statically can be difficult if code and data are mixed or different modules are used, and also it is Impossible if the client uses dynamically generated code. This ability to instrument all code is crucial for accurate and proper handling of libraries because we may need some libraries in our program to be attached. Thus above mentioned advantages of Dynamic binary instrumentation makes it the best technique for many dynamic analysis tools to analyze programs.

II. PIN FRAMEWORK

Figure 1 illustrates Pin's software architecture[1]. At the highest level, Pin consists of a machine (VM), a code cache, and an instrumentation API invoked by Pintools. The VM consists of a just-in time compiler (JIT), an emulator, and a dispatcher. After Pin gains control of the application, the VM coordinates its components to execute the application. The JIT compiles and instruments application code, which is then launched by the dispatcher. The compiled code is stored in the code cache. Entering/leaving the VM from/to the code cache involves saving and restoring the application register state. The emulator interprets instructions that cannot be executed directly. It is used for system calls which require special handling from the VM. Since Pin sits above the operating system, it can only capture user-level code.

As Figure 1 shows, there are three binary programs present when an instrumented program is running: the application, Pin, and the Pintool. Pin is the engine that jits and instruments the application.

The Pintool contains the instrumentation and analysis routines and is linked with a library that allows it to communicate with Pin. While they share the same address space, they do not share any libraries and so there are typically three copies of glibc. By making all of the libraries private, we avoid unwanted interaction between Pin, the Pintool, and the application. One example of a problematic interaction is when the application executes a glibc function that is not reentrant. If the application starts executing the function and then tries to execute some code that triggers further compilation, it will enter the JIT. If the JIT executes the same glibc function, it will enter the same procedure a second time while the application is still executing it, causing an error. Since we have separate copies of glibc for each component, Pin and the application do not share any data and cannot have a re-entrance problem. The same problem can occur when we JIT the analysis code in the Pintool that calls glibc (jilting the analysis routine allows us to greatly reduce the overhead of simple instrumentation on Itanium).

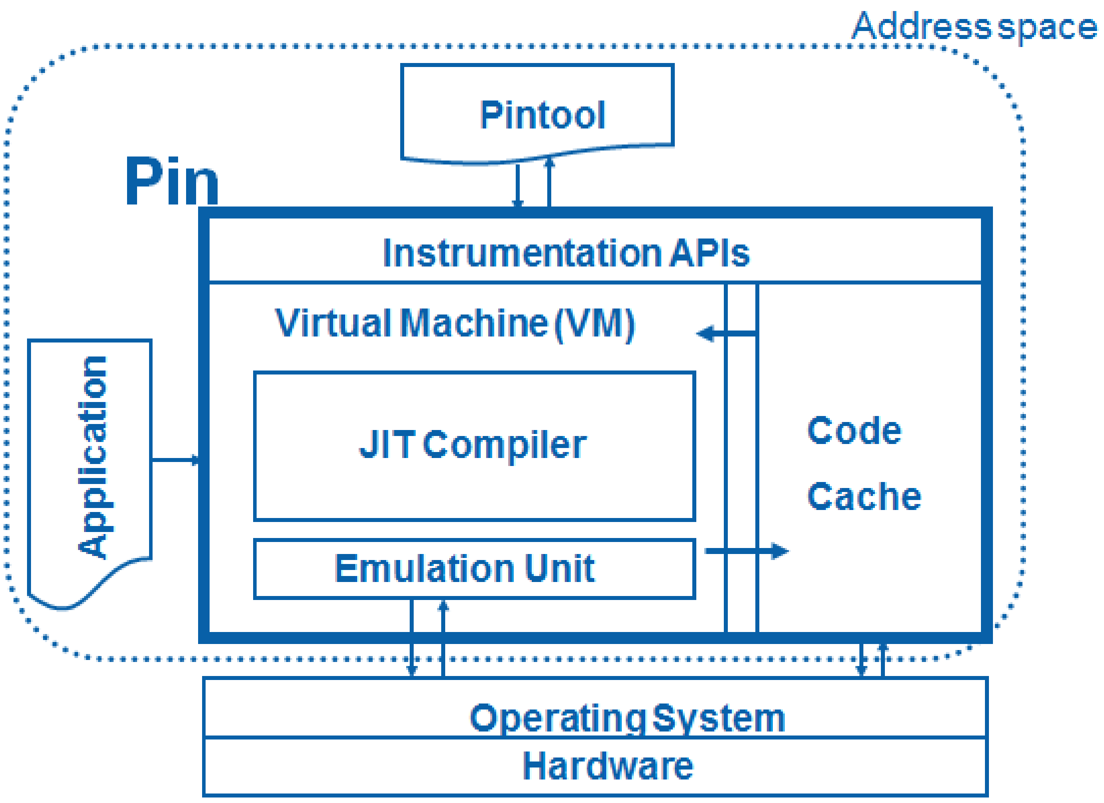

Fig : Pin Architecture[1]

III. PROGRAM ANALYSIS

In the current age of IT revolution we observe that both hardware and software systems are going increasingly complex, therefore the programmers now need much more help than ever. Now they need some tools that can be used to improve program quality, especially correctness and speed. Many such tools use some kind of analysis of programs to determine interesting information about programs.

A. STATIC ANALYSIS AND DYNAMIC ANALYSIS

In the current age of IT revolution we observe that both hardware and software systems are going increasingly complex, therefore the programmers now need much more help than ever. Now they need some tools that can be used to improve program quality, especially correctness and speed. Many such tools use some kind of analysis of programs to determine interesting information about programs. Program analysis can be categorized into following two types according to when the analysis occurs.

Static analysis is the process of analyzing the source code or machine code of the program without need of running it. There are many tools available for static analysis, in particular compilers, the examples of static analysis that are done by compilers are the analysis for correctness, such as type checking and other one is analysis for optimization, which identify the valid performance improving transformations. Also, there are some static analysis tools can identify bugs. The tools that perform static analysis needs to read a program only once in order to analyze it.

Dynamic analysis is the process of analyzing program as it executes or at the runtime. There are many tools available that perform dynamic analysis. Some common examples are checkers, profilers, and execution visualizers. The tools that perform dynamic analysis will instrument the given program with analysis code. The analysis code can be inserted entirely inline, and it may also include external routines that can be called from the analysis code. The analysis code will normally run as part of the program’s normal execution and it will not disturb the current execution and it will do some extra work `on the side`.

B. SOURCE ANALYSIS AND BINARY ANALYSIS

Program analysis can be categorized into another two following types, according to what type of code that is being analyzed.

Source analysis is the process of analyzing programs at the level of source code. There are many tools available that perform source analysis; The most typical example is that of compilers. In this category, the analysis performed on program representations that are directly taken from the source code, such as control flow graphs. Source analysis is generally done for the points of programming language constructs such as expressions, statements, functions, and variables.

Binary analysis is the process of analyzing programs at the level of machine code, that stored either as object code (pre-linking) or executable code (post-linking). In this category, we have analysis that is performed at the level of executable intermediate representations, such as byte-codes, that runs on a particular virtual machine. Binary analysis is generally done for the points machine entities, such as registers, memory locations, procedures, and instructions.

IV. INSTRUMENTATION

Instrumentation is a simple technique for inserting any extra line of code in to an application to observe its behavior. It can be performed at various stages – inside the source code, at compile time, post link time, or even at run time.

Source Code Instrumentation is a way to instrument source programs and **Binary Instrumentation** is to instrument binary executables directly.
Static binary instrumentation (SBI) occurs before the program is run phase, a phase in which we can rewrite executable code or object code. Static binary instrumentation was first used by ATOM, and then followed by others such as Etch and Morph. whereas dynamic binary instrumentation (DBI) is done at run time. The analysis code can be injected by a program into the client program or process, or by any external process. If the client program uses any dynamically linked code, then the analysis code must be added after the dynamic linker has finished its job.
Pin is a framework system that performs run-time binary instrumentation of Windows, Linux or Android applications. Dynamic binary instrumentation has allowing two distinct advantages. Firstly, we do not require the client program to be prepared in any particular or fixed way, which makes it very convenient and easy for users. And second, it naturally covers all code, instrumenting all code statically can be difficult if code and data are mixed or different modules are used, and also it is impossible if the client uses dynamically generated code. This ability to instrument all code is crucial for accurate and proper handling of libraries because we may need some libraries in our program to be attached. Thus above mentioned advantages of dynamic binary instrumentation make it the best technique for many dynamic analysis tools to analyze programs.

V. LITERATURE SURVEY ON INSTRUMENTATION TOOL

A. PINTOOl [4][5]

Conceptually, instrumentation consists of two components:
1.) A mechanism that decides where and what code is inserted
2.) The code to execute at insertion points
These two components are instrumentation and analysis code. Both components live in a single executable, a Pintool. Pintools can be thought of as plugins that can modify the code generation process inside Pin.
The Pintool registers instrumentation callback routines with Pin that are called from Pin whenever new code needs to be generated. This instrumentation callback routine represents the instrumentation component. It inspects the code to be generated, investigates its static properties, and decides if and where to inject calls to analysis functions. The analysis function gathers data about the application. Pin makes sure that the integer and floating point register state is saved and restored as necessary and allow arguments to be passed to the functions. The Pintool can also register notification callback routines for events such as thread creation or forking. These callbacks are generally used to gather data or tool initialization or clean up. Pin is the instrumentation engine and Pintool is the instrumentation program and thus Pin provides the framework and API, Pintools run on Pin to perform meaningful tasks. Programmers can use the classes and functions of the PIN API and write the pintools to perform meaningful task.

B. SOOT [7]

Currently, Soot can process code from the following sources:

- Java (byte code and source code up to Java 7), including other languages that compile to Java bytecode, e.g. Scala
- Android byte code
- Jimple intermediate representation (see below)
- Jasmin, a low-level intermediate representation.

Soot can produce (possibly transformed/instrumented/optimized) code in these output formats:

- Java bytecode
- Android bytecode
- Jimple
- Jasmin

Soot can go from any input format to any output format, i.e., for instance, allows the translation from Android to Java or Java to Jasmin

C. DROZER [6]

Drozer is a comprehensive security audit and attack framework for Android.With increasing pressure to support mobile working, the ingress of Android into the enterprise is gathering momentum. Have you considered the threat posed by the Android app that supports your business function, or Android devices being used as part of your BYOD strategy?
Drozer helps to provide confidence that Android apps and devices being developed by, or deployed across, your organisation do not pose an unacceptable level of risk. By allowing you to interact with the Dalvik VM, other apps' IPC endpoints and the underlying OS.

Drozer provides tools to help you use and share public exploits for Android. For remote exploits, it can generate shellcode to help you to deploy the drozer Agent as a remote administrator tool, with maximum leverage on the device.

Table-1 presents the various instrumentation tool.

TABLE-1: VARIOUS INSTRUMENTATION TOOL

Sr. No	Tool	Description	Usage
1	Pintool [4][5]	Conceptually, instrumentation consists of two components: 1.) A mechanism that decides where and what code is inserted. 2.) The code to execute at insertion points.	Program analysis (memory allocation analysis, error detection, performance profiling, etc...) and for architectural study (processor and cache simulation, trace collection, etc…).
2	Soot[7]	Soot started off as a Java optimization framework.	Soot to analyze, instrument, optimize and visualize Java and Android applications.
3	Drozer[6]	Drozer is a comprehensive security audit and attack framework for Android.	Faster Android Security Assessment.

VI. CONCLUSION AND FUTURE SCOPE

In this paper, we surveyed multiple tools for instrumenting the android application. This types of tool are used for various purpose like Profiling, System call Tracing, program analysis, Optimization,. But pin tool is instrumentation tool for program analysis that allows inserting extra line of code for analysis of application. Soot tool is used for optimization and Drozer is used for security assessment. There are basically three types of tools but among these all tools, for instrumenting the android application pin tool is the best tool among all other types of tools.

REFERENCES

[1] Kim Hazelwood, Artur Klauser,"A Dynamic Binary Instrumentation Engine the ARM Architecture" .Proceeding CASES '06 Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems Pages 261 -270 ,ISBN 1-59593-543-6.

[2] Chi-Keung Luk Robert Cohn Robert Muth Harish Patil Artur Klauser Geoff Lowney StevenWallace Vijay Janapa Reddi † Kim Hazelwood, Pin: building customized program analysis tools with dynamic instrumentation. Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation Pages 190-200,ISBN 1-59593-056-6.

[3] Vijay Janapa Reddi, Alex Settle, and Daniel A. Connors , "PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education", Proceeding WCAE '04 Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture, Article No. 22.

[4] Intel. Pin User Manual, http://rogue.colorado.edu/Pin.

[5] https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.

[6] https://www.mwrinfosecurity.com/media/press-releases/drozer---the-new-android-security-testing-tool/.

[7] http://www.sable.github.io/soot/.