Instrumenting Android Application to Enforce Policies Using Pintools

[1]Umesh P.Lakhtariya, [2]Prof. B. V. Buddhadev

[1]ME student, [2]Professor
Department of Information Technology, SSEC, Bhavnagar, Gujarat,India

***Abstract*- Instrumentation can be defined as the application of instruments, In the form of systems or devices, to accomplish some specific objective in terms of measurement or control, or both. Program instrumentation is a widely used mechanism in different software engineering areas. It can be used for creating profilers and debuggers for detecting programming errors at runtime, or for securing programs through inline reference monitoring. Instrumenting is also possible with using soot, drozer and pin tools.Soot is useful for creating a jimplecode, grimple code and shimple code and baf code. This paper presents on instrumenting Android applications using Pin Tools for exploring ways for improving instrumentation system. Here, we also provide monitoring and profiling of such relevant application like short message services to enforce policies using pintools.**

***Index Terms*—Android, Pintools, Security, Policies, Dynamic Analysis, Profiling, Monitoring.**

I. Introduction

Smart phones have been a vulnerable target for malware since June 2004. The number of infected applications steadily increased until certain security measures like application signing and validation of developers was introduced. Android phones are one such smart phones that were and continue to be a prime target for hackers.

Android is one of the most widely used mobile operating system in the world representing over 72% of the market share [1]. More than 500,000 Android apps available in dozens of markets applications can be installed end-users. In the official Google market (Google Play, Android Market above), more than 10 000 new applications are available every month.

For the end user, downloading an application on your smart phone is similar to the choice an apple on an apple tree: only see the surface and no evidence that there is no worm in it. Unfortunately there are many different types of worms waiting to infect smart phones as malware leaking private data and adware call premium numbers. With Android phones being ubiquitous, they become a worthwhile target for security and privacy violations. Attacks range from broad data collection for the purpose of targeted advertisement, to targeted attacks, such as the case of industrial espionage. Attacks are most likely to be motivated primarily by a social element: a significant number of mobile-phone owners use their device both for private and work-related communication [2].

An efficient and easily applicable means to enhance the privacy of Android applications is to conduct monitoring at runtime and the interception of the application interact with the Android stack for implementing bytecode application directly on your smart phone.

Why performing Instrumentation?

There are at least two ways to perform runtime monitoring and interception: modification of the Android software stack or bytecode instrumentation. Modification of the software execution stack consists in altering the operating system or the core libraries to intercept the required information. On Android, it means changing the underlying kernel, the Dalvik virtual machine or the Android framework. Unless convincing the Android consortium, this is rather limited in deployment since normal end-users have neither the rights (jailed phones) nor the ability to do so. Also, this solution would require users to change their firmware which is a non-trivial task, further complicated by the so called fragmentation problem of the Android system as there is not a single Android system but many different Android systems each customized to run on a specific device (tablet, smartphone . . .). If the operating system is modified, one would need to create a custom instrumented version for every possible Android version which is not easily doable in practice. Bytecode instrumentation however, is one of the lightest way to perform runtime monitoring on top of execution platform that can not be modified. In the context of a fine-grained policy enforcement for improving privacy, we are able – thanks to bytecode instrumentation – to enforce a fine-grained permission model of already deployed applications on Android smartphones without any modification of the Android software stack.

II. Android

The android OS can be conceptually broken down into four layers. The lowest layer is the Linux kernel, which provides basic operating system services such as memory management, process separation,and device drivers. The next layer is the Android runtime, system daemons and support libraries. The android runtime is implemented using the Dalvik and virtual machine. The Dalvik VM as well as the system daemons and support libraries are executed as native code. The last two layers are the Android application framework and the actual applications. Both are implemented in managed code and are executed by the Android runtime using the Dalvik VM.

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet

computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it also has been used in game consoles, digital cameras, regular PCs (e.g. the HP Slate 21) and other electronics.
Android using following tools:
(1)Eclipse: Eclipse is a multi-language software development environment compromising an integrated development (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop application in Java and ,by means of various plug-ins,other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, R, Ruby(including Ruby on Rails framework), Scala, Clojure, Groovy and scheme. It can also be used to develop packages for the software Mathemetica.
(2)ADT Plugins: Android Development Tools(ADT) is a plugin in for the Eclipse IDE that is designed to give you a powerful, integrated environment in which to build Android applications. ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug your applications using the Android SDK tools, and even expert signed(or unsigned) .apk files in order to distribute your application.
(3)SDK toolkit: SDK Tools is a downloadable component for the Android SDK. It includes the complete set of development and debugging tools for the Android SDK. It contains tools for debugging and testing , plus other utilities that are required to develop an application.

Android architecture:

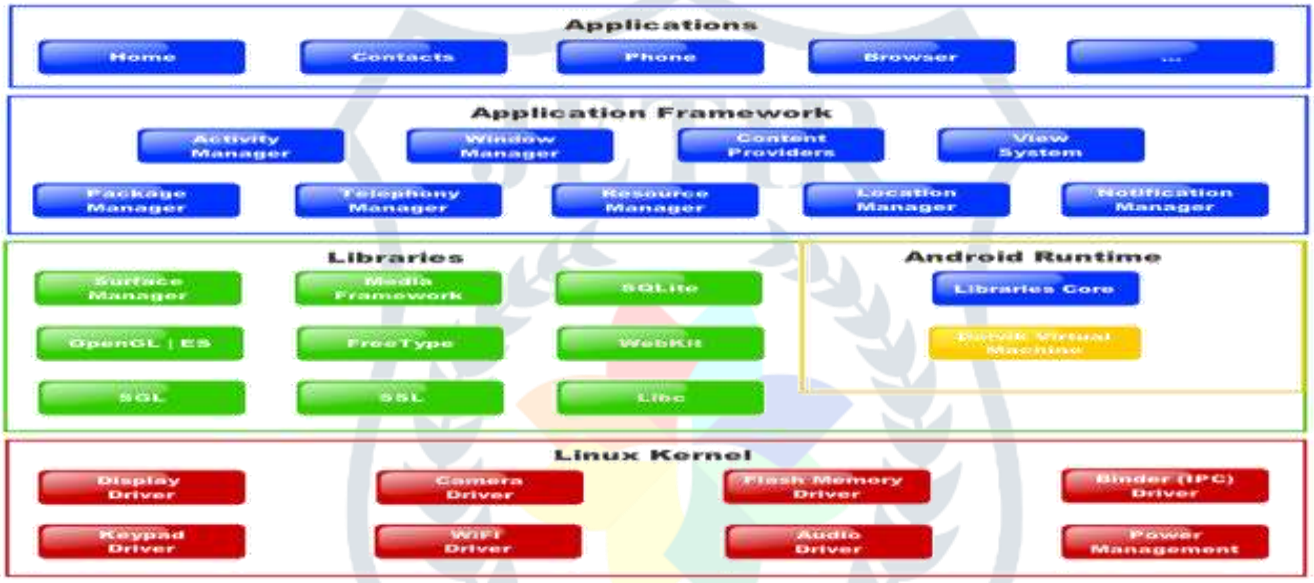

Fig 1, Android Architecture

Linux kernel: Positioned at the bottom of the Android software stack, the Linux Kernel provides a level of abstraction between the device hardware and the upper layers of the Android software stack. Based on Linux version 2.6, the kernel provides preemptive multitasking, low-level core system services such as memory, process and power management in addition to providing a network stack and device drivers for hardware such as the device display, Wi-Fi and audio.
Android Runtime: As previously noted, the Linux kernel provides a multitasking execution environment allowing multiple processes to execute concurrently. It would be easy to assume, therefore, that each Android application simply runs as a process directly on the Linux kernel. In fact, each application running on an Android device does so within its own instance of the Dalvik virtual machine (VM).

Libraries: This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access.
Application Framework: The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed. This framework implements the concept that Android applications are constructed from reusable, interchangeable and replaceable components. This concept is taken a step further in that an application is also able to publish its capabilities along with any corresponding data so that they can be found and reused by other applications. It includes services like: Activity Manager, Content Providers, Resource Manager, Notification Manager, View System, Package Manager, Telephony Manager and Location Manager.
Applications: Located at the top of the Android software stack are the applications. These comprise both the native applications provided with the particular Android implementation (for example web browser and email applications) and the third party applications installed by the user after purchasing the device.

III. Pintool
Before the introduction of pintools there is a need of introduce ourselves with pin.

A. Introduction of Pin

Pin is a dynamic binary instrumentation engine. Instrumentation is a technique that inserts code into a program to collect run-time information. It can be used for several purposes, mostly for program analysis (performance profiling, error detection, memory allocation analysis, etc...) and for architectural study (processor and cache simulation, trace collection, etc…) Pin is used for the instrumentation of programs. It supports Linux*,Windows*, macOS and Android* executables for IA-32, and Intel(R) 64. Pin allows a tool to insert arbitrary code (written in C or C++) in arbitrary places in the executable. The code is added dynamically while the executable is running. Pin provides a rich API that abstracts away the underlying instruction set idiosyncracies and allows context information such as register contents to be passed to the injected code as parameters. Pin automatically saves and restores the registers that are overwritten by the injected code so the application continues to work. After the introduction of pin we have to introduce ourselves with pintool.

- **PIN Framework**

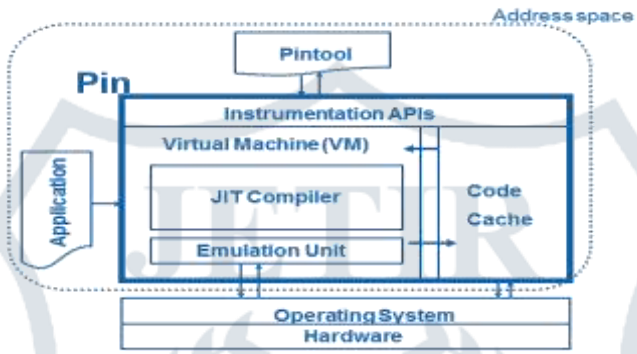

Fig 2 Pin Software architecture[2]

Figure 2, illustrates Pin's software architecture. At the highest level, Pin consists of a virtual machine (VM), a code cache, and an instru- mentation API invoked by Pintools. The VM consists of a just-intime 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 2 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

B. Introduction of Pintools

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 codes to be generated. This instrumentation callback routines represents the implementation component.It inspects code to be generated ,investigates its static properties, and decides if and where to inject calls to analysis function.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 argument to be passed to the functions.
The pintool can also register notification callback routines for event such as thread creation or forking. These callbacks are generally used to get data tool initialization or cleanup.

C. Instrumentation with Pin

Instrumentation is a technique for inserting extra code in to an application to observe its behavior Instrumentation can be performed at various stages: in the source code, at compile time, post link time, or at run time. Source Code Instrumentation is

away to instrument source programs and Binary Instrumentation is instrument binary executable directly. Pin is a software system that performs run-time binary instrumentation of Linux applications. Pin is a dynamic binary instrumentation engine. It can be used for several purposes, mostly for program analysis (performance profiling, error detection, memory allocation analysis, etc…) and for architectural study (processor and cache simulation, trace collection, etc…) Pin is used for the instrumentation of programs. It supports Linux,Windows, macOS and Android* executables for IA-32, and Intel(R) 64[6].Pin allows a tool to insert arbitrary code (written in C or C++) in arbitrary places in the executable. The code is added dynamically while the executable is running. The best way to think about Pin is as a "just in time" (JIT) compiler (a.k.a dynamic compiler). The input to this compiler is not bytecode, but a regular executable. Pin dynamically re-compiles the application during execution. Dynamically compiled code is cached and reused for boosted performance. Pin provides a rich API[7] that abstracts away the underlying instruction set idiosyncracies and allows context information such as register contents to be passed to the injected code as parameters. Pin automatically saves and restores the registers that are overwritten by the injected code so the application continues to work.

III. Applying a Pintool to an application

An application and a tool are invoked as follows:
pin [pin-option]... -t [toolname] [tool-options]... -- [application] [application-option]
The following Pin-options are frequently used:

- *-t toolname:* specifies the Pintool to use.
- *-pause_tool n:* is a useful Pin-option which prints out the process id and pauses Pin for n seconds to permit attaching with gdb.
- *-pid pid:* attach Pin and the Pintool to an already running executable with the given process id.

The *tool-options* follow immediately after the tool specification and depend on the tool used.
Everything following the -- is the command line for the application.

IV. Proposed Approach

Profiling and Monitoring are techniques for finding malicious activities runs into the application. Monitoring is also use for providing security into the android application. Nowadays attackers try to attack on each and every android applications. They are also trying to attack on Symbian based applications and iOS based applications. So, Detecting that types of malicious activities and enforcing policies into android applications we will use Pintool. Using Pintool we will instrument Android application and then by using profiling and monitoring technique we detect the malicious activities and provide the security.

For Detecting a broader range of malicious activities we would need to consider monitoring and profiling more relevant events such as access to telephony and short messaging services.

Our approach successfully hardens application security in two important aspects:

1) Privacy—data can only be used for its intended purpose in applications, i.e., the information flow is monitored and controlled.
2) Security—data used in the execution of code, e.g., the return address on the stack used by a RET instruction is monitored for the absence of tags, that would reveal an integrity compromise.

V. Conclusion& Future work

In this paper we have presented about to enforce policies in android application by Instrumenting android application using pintools. We have also show the method of applying pintool to an application. To detect malware from application we have to use different techniques like, Dynamic analysis, Profiling and Monitoring.

In future work we will use pintool for Instrumenting android application to provide security and save applications from malware activities and attackers.We will use applications like SMS messaging services and telephony services to enforce policies through inline reference monitoring

VI. Acknowledgement

My thankfulness is first for the my dear Father and Mother because it was Their willing to give me the opportunity, the knowledge and the understanding to join the master.
My special thanks are for Prof. B.V. Buddhdev, who was my constant support and guide the generously shared his expertise to provide me the right direction to address the issues that I face in the development of this paper.

References

[1] Heidi Pan and Krste Asanovi´c, "Controlling Program Execution through Binary Instrumentation", Massachusetts Institute of Technology, ISSN: 0163-5964,2010
[2] Vijay Janapa Reddi, Alex Settle, and Daniel A. Connors "PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education", University of Colorado, Boulder.

[3] Gang-Ryung Uh, Robert Cohn, Bharadwaj Yadavalli, Ramesh Peri, Ravi Ayyagari", Analyzing Dynamic Binary Instrumentation Overhead" ISBN: 1-59593-080-9, 2005
[4] D.L. Bruening. Efficient, Transparent, and Comprehensive Runtime Code Manipulation. PhD thesis, M.I.T.(http://www.cag.lcs.mit.edu/dynamorio), September, 2004.
[5] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Reddi, and K. Hazelwood.," Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation". In Proceedings of the SIGPLAN 2005 Conference on Programming Language Design and Implementation, June, 2005.
[6] N. Nethercote and J. Seward. Valgrind: A program supervision framework. In Proceedings of the 3rd Workshop on Runtime Verification (http://valgrind.kde.org). 2003.
[7] Mohammad Karami, Mohamed Elsabagh, Parnian Najafiborazjani, and Angelos Stavrou,"Behavioral Analysis of Android Applications Using Automated Instrumentation".Computer Science Department, George Mason University, Fairfax, VA 22030
[8] Nico Golde and Collin Mulliner,"Countering SMS Attacks: Filter Recommendations",Security in Telecommunications Technische Universit¨ at Berlin {nico,collin}@sec.t-labs.tu-berlin.de
[9] Collin Mulliner and Charlie Miller,"Injecting SMS Messages into Smart Phones for Security Analysis"Deutsche Telekom Laboratories / TU-Berlin and Independent Security Evaluators.
[10] Milind Chabbi, Xu Liu, John Mellor-Crummey,"Call Paths for Pin Tools",CGO '14, February 15–19 2014, Orlando, FL, USA,ACM 978-1-4503-2670-4/14
[11] Hendrik Schweppe, Yves Roudier,"Security and Privacy for In-Vehicle Networks",EURECOM Sophia-Antipolis, France
[12] https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
[13] https://groups.yahoo.com/neo/groups/pinheads/info
[14] Intel. Pin User Manual. https://software.intel.com/en-us/articles/pintool

WEBSITES

[15] Pin - A Dynamic Binary Instrumentation Tool, https://software.intel.com/en-us/articles/pintool
[16] CGO2013 Pin Tutorial, https://sites.google.com/site/pintutorial/home/cgo2013
[17] ASPLOS2014 Pin Tutorial, https://sites.google.com/site/pintutorial/home/asplos2014
[18] Pintool.org, https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
[19] Pin(computer program) - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Pin_%28computer_program%29