Pin for Android (Pindroid)

Tutorial

- Make sure to check http://pintool.org and get the latest Android Kit and the latest copy of this tutorial.

Revision History (newest on top)			
0.6	7-5-2014	Nitzan Mor-Sarid	Updated copyright notice.
0.5	8-4-2014	Nitzan Mor-Sarid	Fixed a reference to NDK r7 that should have been r9.
0.4	26-11-2013	Nitzan Mor-Sarid	Changed tar extraction command. Removed request to copy libgnustl_shared from NDK. It already exists in the runtime tar file. Updated NDK version. Added information about attaching as root. Added information about running an app from scratch.
0.3	19-8-2013	Michal Nir-Gross Nitzan Mor-Sarid	Added information about attaching Pin to a running Android application.
0.2	13-4-2013	Michal Nir-Gross Nitzan Mor-Sarid	Kit 58423.
0.1	20-12-2012	Michal Nir-Gross Nitzan Mor-Sarid	Created

Table of Contents

1. Introduction

1.1 Purpose of this Document

This document describes how to use Pin on Android applications.

1.2 What is 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.

Pin includes the source code for a large number of example instrumentation tools like basic block profilers, cache simulators, instruction trace generators, etc.

More information about Pin can be found at: http://www.pintool.org/

1.3 What are Pin tools?

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.

The Pin kit includes many tools (they can be found at: pin-w-x-y-android/source/tools). The tools are provided as source files, so it is possible to extend and change them according to the user needs.
You can also write your own tools. More information about Pin Tools and about the Pin tools API can be found at: http://www.pintool.org . The API for Pin for Android tools is the same as the Linux API.

1.4 Disclaimer and Legal Information

The information in this manual is subject to change without notice and Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. The information in this document is provided in connection with Intel products and should not be construed as a commitment by Intel Corporation.

EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Android is a trademark of Google Inc.

Other names and brands may be claimed as the property of others.

2. Building and running Pin on Android - A step-by-step tutorial

***** Make sure to check http://pintool.org and get the latest Android kit and the latest copy of this tutorial *****

In the following tutorial we will install and run Pin on an Android device. We will run Pin from command line with a Pin tool called "inscount2" – a simple Pin tool that counts the number of dynamic instructions that were executed.
Afterwards we will attach Pin to a running Android application.

2.1 Setting up the environment

2.1.1 Prerequisites

To run Pin on Android you will need the following:

1. A host machine with Linux. The recommended Linux version is Ubuntu 10.4
2. An Android device.
3. Busybox installed on the device, or any other way to run "tar" command on the Android device.
   - Installing busybox:
     a. Download a suitable binary from http://www.busybox.net (e.g. http://www.busybox.net/downloads/binaries/latest/busybox-i686)
     b. Push the busybox binary to your Android device.
     c. Make sure that you are able to run busybox.
        On the Android device run:
        `./busybox --help`

2.1.2 Configuring your host machine

On your host machine:

1. Download the Android SDK. http://developer.android.com/sdk/index.html
2. Download the Android NDK version r9 and extract it. http://developer.android.com/tools/sdk/ndk/index.html (look for the link to download NDK version r9).
3. Download the Pin kit for Android (you can get it at: http://www.pintool.org/).
4. Add the SDK command line tools to your path (adb specifically).
   ```
   PATH="$PATH:$<sdk-root>/tools"
   PATH="$PATH:$<sdk-root>/platform-tools"
   ```
5. Create a standalone toolchain for Android in /usr/android using the following command:
   ```
   build/tools/make-standalone-toolchain.sh --toolchain=x86-4.6 --install-dir=/usr/android
   ```
   - You can create the standalone toolchain at any other directory, however our instructions assume that it was created at /usr/android.

6. In /usr/android/bin create symbolic links to the long names for these two utils:

```
link i686-android-linux-gcc gcc
link i686-android-linux-g++ g++
link i686-android-linux-ar ar
link i686-android-linux-ranlib ranlib
```

2.1.3 Installing Pin on the device

- Pin-W-X-Y-android- refers to the Pin kit root (w-x-y – stands for the kit version number).

1. Choose a working directory on the Android device (It should be a directory that you have permissions to execute from, e.g. "data".)
   - If you are using an unrooted phone and you can't find a directory on the device that you can execute from, try to run:
     `adb install <any apk file>`
     Check which directory it copies to apk file to – you should have execution privileges from this directory even on unrooted phones.

2. **On the <u>host</u> machine**:
   a. Unzip the Pin Kit.
   b. Copy pin-W-X-Y-android/android-install.tar.gz to your working directory on the Android device.
      `adb –s <device> push pin-W-X-Y-android/android-install.tar.gz <Your working directory on the Android device>`
3. **Connect to your <u>Android device</u>** (run: `adb shell`):
   Unzip install.tar.gz on the Android device into your working directory.
   (e.g. at your working directory `./busybox tar -zpxvf android-install.tar.gz`)

To test your installation:

Connect to your <u>Android device</u> (run: `adb shell`):
`cd <Your working directory on the Android device>`
Run:
`./pin -version`
and
`./pin -- /system/bin/ls`

(continued on the next page)

Both commands should run without errors. e.g:

```
>>># ./pin -version
Pin 2.13
Copyright (c) 2003-2013, Intel Corporation. All rights
reserved.
@CHARM-VERSION: $Id: version.cpp 60992 2013-08-21 10:18:06Z
mnir1 $
@CHARM-BUILDER: BUILDER
@CHARM-COMPILER: gcc 4.4.3
@CHARM-TARGET: ia32
@CHARM-CFLAGS:  __OPTIMIZE__=1  __NO_INLINE__=__NO_INLINE__

>>>>#./pin -- /system/bin/ls
android-install.tar.gz
busybox
ia32
pin
pin.tar.jz2
source
```

2.2 Compiling the Pin tool.

To use inscount2 tool, we will have to compile it.

To compile the tool:

1. **On the <u>host</u> machine**, from the unpacked kit root directory, cd to the tool's directory: `cd source/tools/ManualExamples`.
   As you can see, the ManualExamples directory contains many Pin tools. One of them is inscount2.cpp.
2. **On the <u>host</u> machine** we will compile the tool by using make : `make HOST_ARCH=ia32 TARGET_OS=android CC=/usr/android/bin/gcc CXX=/usr/android/bin/g++ obj-ia32/inscount2.so`

   If you created the standalone toolchain in another directory, change CC and CXX according to that directory and also add `LPATHS=-L<kit home>/runtime/stl/libgnustl_shared.so`

   The make command should complete without errors. If there are any errors – make sure that you defined the toolchain correctly.

   The compiled tool (inscount2.so) will be at `source/tools/ManualExamples/obj-ia32` (on the host machine).

2.3 Run Pin with the tool on the Android device from command line

First, we will copy the tool to the Android device.

1. **On the <u>Android device</u> run**:
   `mkdir <Your working directory on the device>/obj-ia32`
2. **From the <u>host</u> machine**: Copy the tool to <Your working directory on the device>/obj-ia32

```
adb –s <device> push /source/tools/ManualExamples/obj-
ia32/inscount2.so <Your working directory on the
device>/obj-ia32
```

Before we'll try to run Pin we will have to choose which application we would like to instrument. For the tutorial we will use /system/bin/ls.

Run Pin with the tool on /system/bin/ls:

From the <u>Android device:</u>

1. Cd to <Your working directory on the device>
2. Run:

```
./pin -t obj-ia32/inscount2.so -- /system/bin/ls
```

```
>>># ./pin -t obj-ia32/inscount2.so -- /system/bin/ls
android-install.tar.gz
busybox
ia32
inscount.out
obj-ia32
pin
pin.log
pin.tar.jz2
source
```

When we look at the output – we can see that pin has successfully finished instrumenting /system/bin/ls (we can see /system/bin/ls output on the screen).
Now let's examine the tool output: If we will look at the tool's sources we will see that the tool writes its output to a file called inscount.out

```
>>># cat inscount.out
Count 571038
```

This means that the number of instructions that were executed while running /system/bin/ls on my device was: 571038.

In the same way, you can compile the other tools at the kit and run them on your Android device on any application of your choice.

2.4 Attaching Pin to a running Android application

Note: there is some inherent confusion on what 'Apps' means in the context of Android and Pin. App is an abbreviation of application, which in Pin's vocabulary is the program being instrumented, be it command line or GUI. In the context of Android, an app(lication) is reserved for programs installed from .apk files that use specific APIs and (usually) Android's GUI. In the following section the words app and application refer to Android apps. With the note in mind, here are a few things to be aware of:

- Pin will run properly only on applications whose native components are x86 binaries. Make sure the application you are attaching to doesn't have a native component or has an x86 compiled native component.
- Since pin will be running with the application's permissions, pin executables and runtimes must be world readable in terms of permissions.
- Make sure pin and your pin tool only write files to places where your application has permissions to write to (INCLUDING pin's logfile and the pintool's logfile). We recommend writing files to the application's private store. This varies from device to device but for most devices it is likely in /data/data/application.package.name/. It is also possible to write the logs to an SD card if your device has one.

1. Log in to the Android device and start the app:

```
adb shell
am start –n \
com.package.name/com.package.name.ActivityName
```

- If your app is already running you can skip this step.

2. Using 'ps', figure out the app's PID. This will be used to tell pin what to attach to.
   `ps | grep com.packge.name`

3. Attach Pin to the app:

```
run-as <application-package-name> <full-path-to-pin>/pin –PID
<application PID> \
-logfile <path-to-logfile> <additional pin cmd line arguments> -t \
<full-path-to-tool>/tool.so –logfile <application-package-
name>/pintool.log
```

- If the tool also writes to an output file you must also specify the path to the file in the command line.
- For the run-as command to work properly, the application must be debuggable. This allows us to run Pin with the application's privileges. If the application is not debuggable and the device is rooted, while running as root, try to run pin without the "run-as <application-package-name>" prefix. Pin will still run with the application's privileges, so write logfiles to the

2.5 Launching Pin on an Android Application Started from Scratch

- It is also possible to launch a new process for the app with Pin instrumenting it from scratch. All application processes are actually instances of an executable called app_process. Normally, when an app is launched, it is cloned from Zygote, a pre-initialized app_process instance, to improve time and memory performance. This is obviously no good if we want Pin to instrument the process from scratch.
- Fortunately, Android supports a special mode for running apps from scratch. This mode is turned on by defining a system property (using propset) that looks like this: wrap.<app.name.space>.
- When this property is set, the content of the property will be initialized with parameters that when executed, will start the application itself. We will use that to create a script that will run Pin on the app.

1. In your device's working directory, create a file called launchpin.sh, with the following content, and directory paths replaced with real paths:
   ```
   #!/system/bin/sh
   echo Running $* with pin launcher >
   /path/to/app/launch.log
   exec /path/to/pin -xyzzy -logfile /path/to/app/pin.log -t
   /path/to/pintool -- $*
   ```
2. This file will be the wrapper that will launch pin. Make sure to always use full paths for everything, because this script will not be executed from the same environment your shell is running on. Now let the system know you want to execute the file. Change to root and execute:
   ```
   setprop wrap.application.name.space "logwrapper
   /path/to/launchpin.sh"
   ```
3. Finally, run the app:
   ```
   am start -n application.name.space/.Main
   ```
4. The application should start with Pin instrumenting it from the beginning.

3. Useful links:

- http://www.pintool.org/ - the official Pin website. Includes the use manual, examples, and the Pin kits.
- http://tech.groups.yahoo.com/group/pinheads/ - the Pin community, a great place to ask questions regarding Pin.
- Busybox: http://www.busybox.net/
- Android SDK download: http://developer.android.com/sdk/index.html
- Android NDK download: http://developer.android.com/tools/sdk/ndk/index.html
- Intel based devices are available from a variety of vendors online, and it's also possible to run Pin JB version on the JellyBean x86 Android QEmu emulator, available from the Android SDK. Make sure there's an accepted way to root the device before purchasing it