Dynodroid: An Input Generation System for Android Apps

Aravind Machiry

December 19, 2013

Part I

Preface

Chapter 1

Organization of this Guide

Dynodroid is a system for automatically generating relevant inputs to Android apps. It is capable of generating both UI inputs (e.g., touchscreen taps and gestures) and system inputs (e.g., simulating incoming SMS messages). It also allows interleaving inputs from machine and human. The system is both *stand-alone*, in that it provides few testing strategies for users to test an app, and *extensible*, in that it allows users to write and run their own strategies. As a result, Dynodroid has two kinds of users: *end-users*, who only wish to use the system, and *developers*, who additionally wish to develop their own testing strategies.

For convenience, this user guide consists of two parts: a guide for end-users and a guide for developers. Unlike end-users, developers need to understand Dynodroid's source code and API, and code written by them is executed as part of a Dynodroid run. Hence, the guide for end-users concerns how to run Dynodroid, and the guide for developers concerns how to extend and debug.

Chapter 2

Acknowledgments

Dynodroid would not be possible without the following open-source software:

- Android SDK, Android SDK
- Android SDK tools, development and debugging tools for the Android SDK.

Dynodroid relies on the following open-source tools and libraries:

- Emma, a Java code coverage measuring tool
- Apk tool, a Tool for reverse engineering android apk files

Dynodroid was supported in part by DARPA (contract FA8750-12-2-0020) and gifts from Google and Microsoft.

Part II

Guide for End-Users

Chapter 3

What is Dynodroid?

Dynodroid is an automated input generation system for android apps which uses technique based on a novel observe-select-execute principle. Dynodroid works by viewing a mobile application as an event-driven program that interacts with its environment by means of a sequence of events, it considers both input events(specific to the app) and system events(common to all apps) to determine which events are relevant to the application in the current state. A selected event from this pool of possible events may then be executed to yield a new state, iteratively, It also provides various log monitoring clients which monitor the behaviour of app as it is being tested. It has the following key features:

- It works on apps both with sources and without sources (apks). For apps with sources, it additionally generates code coverage reports using Emma (cite this).
- It allows user to specify different selection strategies, which are the techniques used to pick an event from the set of available events.
- It observes relevant events for the app, uses the configured selection strategy to select a event and executes the event. This continues till it executes the provided number of events.
- It is multithreaded and can be used to test several apps with various combinations of selection strategies and event counts in a single run. As Dynodroid can take long time to run, it provides a notification feature which is an email that will be sent when it finishes the execution.

Dynodroid is intended to work on atleast Linux and MacOS. It is open-source software distributed under the New BSD License. Improvements by users are welcome and encouraged. The project website is `http://code.google.com/p/dyno-droid/`.

Chapter 4

Getting Started

This chapter describes how to download, deploy, and run Dynodroid. Section 4.1 describes how to download the source code of Dynodroid and Section 4.2 explains how to deploy it. Finally, Section 4.3 describes how to run Dynodroid.

4.1 Downloading Source Code

To obtain a stable version of dynodroid source code from the SVN repository run the following command:

```
svn checkout https://dyno-droid.googlecode.com/svn/branches/dynodroid.0.1 dyno-droid.0.1
```

Also, you can obtain the latest development snapshot from the SVN repository by running the following command:

```
svn checkout http://dyno-droid.googlecode.com/svn/trunk/ dyno-droid
```

4.2 Deploying

Deploying Dynodroid requires the following software:

- Python 2.7 or higher.
- Android SDK 2.3.x (API 10)
- Android SDK tools
- Android SDK Platform-tools

On ubuntu 10 or higher, you can follow the below instructions to setup the required software:

- **Installing Python:**

```
sudo apt-get install python
```

- **Setting up Android:**

```
Browse to the directory where you want to install sdk. For ex: ~/sdk_install
cd ~/sdk_install
wget http://dl.google.com/android/android-sdk_r22.3-linux.tgz -O sdk_tools.tgz
tar -xvzf sdk_tools.tgz
./android-sdk-linux/tools/android
A windows pops up, select Android SDK platform-tools
and Android SDK 2.3.x (API 10) and press install.
```

- **Setup environment variables:** After setting up android, you need to make changes to the environment variables by appending following lines to .bashrc:

```
export SDK_INSTALL=<absolute path of ~/sdk_install/android-sdk-linux from the above step>
export PATH=$SDK_INSTALL/tools:$SDK_INSTALL/platform-tools
```

To deploy Dynodroid, run command "`python dynodroidsetup.py . <directory_to_deploy>`" in the SVN directory. This will set up the provided deployment directory by copying the required sources, libraries and tools from SVN directory. It will also emit preliminary instructions on next steps to run Dynodroid.

4.3 Running Dynodroid

Running Dynodroid requires a JVM supporting Java 5 or higher.

- **Copy the required apps:** You need to copy the apps(either with sources or apk) that needs to be tested to the apps folder under deployment directory.

```
cp -r app1 directory_to_deploy\apps
cp app2.apk directory_to_deploy\apps
```

- **Run:**

```
cd directory_to_deploy
ant run
```

Chapter 5

Dynodroid Properties

The only way to specify inputs to Dynodroid is by means of properties. Dynodroid expects 2 command line arguments:

1. **Run mode:** Dynodroid supports 2 modes of running, standalone mode(argument : ser) where we run Dynodroid as a standalone application and Java Remote Message Invocation(RMI) mode (argument : rmi) where we run Dynodroid as RMI server to accept requests to test an app. But currently RMI mode is not well tested.

2. **Path of the properties file:** The path to the file containing all the properties required for the Dynodroid.

The ant build script generated by the deployment step takes care of these. `ant run` will run the Dynodroid in standalone mode with properties file `dynodroid.properties` present in the deployment directory. Section 5.1 explains how to set properties and Section 5.2 explains the meaning of properties recognized by Dynodroid. Notation `[<...>]` is used in this chapter to denote the value of the property named `<...>`.

5.1 How to Set Properties

There is only one way to pass properties for Dynodroid Via a user-defined *properties file* whose location is the second argument passed to Dynodroid. The default properties file is `dynodroid.properties` in the deploy directory. You can modify the `build.xml` in the deploy directory to use different properties file. Properties are provided as key value pair `<key>=<val>`, with one property per line. By default `dynodroid.properties` contain known good values, but you can change then or provide your own properties file as mentioned before by changing `build.xml`.

5.2 Recognized Properties

The following properties are recognized by Dynodroid. The separator for list-valued properties is a comma.

5.2.1 Input and Output directories

This section describes properties that control the input and output directories Dynodroid uses to read the apps and writes the results respectively.

`app_dir`

Type: location
Description: Directory containing the apps that need to be tested. Apps with sources should be in a folder, where as apks can be directly under this directory.
Default value: folder name apps under deploy directory.

`work_dir`

Type: location
Description: Directory where the test results should be stored. There will be a folder for each test variation in the format AppName_¡TestStrategy¿_¡SelectionStrategy¿_¡NoofEvents¿.
Default value: folder name workingDir under deploy directory.

5.2.2 Test Profile or variation

This section describes properties that control the different variations to be tested on each app.

`test_strategy`

Type: String list
Description: List of test strategies to run.
Available values:

1. **WidgetBasedTesting**: This is the actual Dynodroid testing strategy.
2. **RandomMonkeyTesting**: This is the android monkey testing strategy.

Default value: WidgetBasedTesting

WidgetBasedTesting Properties

This section describes properties that are used when WidgetBasedTesting is provided as one of the testing strategy.

`max_widgets`

Type: Integer list
Description: Maximum number of events that needs to be tested.
Default value: 1000

`sel_stra`

Type: String list
Description: The selection strategy that needs to be used to pick an event from the available events.
Available values:

1. **RandomBased**: An event is picked randomly from the available list.
2. **FrequencyBased**: An event is picked based on the frequency/number of times it has been triggered.
3. **RandomBiasBased**: An event is picked randomly with negative bias towards already triggered events.

For more details of these selection strategies, please refer our paper Dynodroid FSE 2013.
Default value: RandomBiasBased

RandomMonkeyTesting Properties

This section describes properties that are used when RandomMonkeyTesting is provided as one of the testing strategy.

`event_count`

Type: Integer list
Description: Maximum number of events that needs to be tested.
Default value: 100

`verbose_level`

Type: Integer
Description: Verbosity level of the logs to be displayed by monkey.
Possible Values: 1,2 or 3.
Default value: 1

`tch_pct`

Type: Integer (Maximum value 100)
Description: Integer indicating the percentage of touch events in the random event generator.
Default value: 15

`sml_pct`

Type: Integer (Maximum value 100)
Description: Integer indicating the percentage of small navigation events in the random event generator.

Default value: 25

`mjr_pct`

Type: Integer (Maximum value 100)
Description: Integer indicating the percentage of major navigation in the random event generator.
Default value: 15

`trk_pct`

Type: Integer (Maximum value 100)
Description: Integer indicating the percentage of track events in the random event generator.
Default value: 15

5.2.3 Results Notify

This section describes properties that are used to notify the completion of Dynodroid run.

`complete_notify`

Type: Email address
Description: Email-id to be notified post completion of entire run.
Default value: someone@example.com

`report_email_user`

Type: Email address
Description: **gmail** id to be used to send notification mail.
Default value: reportSourceUserName@gmail.com

`report_email_pass`

Type: string
Description: Password of the account specified by reportemailuser property.
Default value: password

All other properties can be ignored, as they configure the location of tools used by Dynodroid. The deployment script takes care of populating these with correct values.

Chapter 6

Dynodroid Test Profiles

As mentioned earlier Dynodroid is multithreaded. This chapter describes how Dynodroid interprets few properties and run various test variations for a given app. Each test variation is called **Test Profile**.

For each Test Profile:

- A clean emulator will be used.
- A Dedicated directory will be created under the provided working directory where all the logs will be stored.
- A Dedicated Java thread handles the execution.

Following logic is used is used to compute the number of Test Profiles to be run:

```
WidgetBasedTesting_Present = 1 If test_strategy contains WidgetBasedTesting else 0.

Number_Of_Selection_Strategies = Number of Selection Strategies specified
under sel_stra property.

Number_Of_Widget_Counts = Number of widget counts specified under max_widgets property.

WidgetBasedTesting_Combinations =
WidgetBasedTesting_Present * Number_Of_Widget_Counts * Number_Of_Selection_Strategies;

RandomMonkeyTesting_Present = 1 If test_strategy contains RandomMonkeyTesting else 0.

Number_Of_Event_Counts = Number of events counts specified under event_count property.
```

```
RandomMonkeyTesting_Combinations =
RandomMonkeyTesting_Present * Number_Of_Event_Counts;

    Total Number of Test Profiles =
    Number_of_Apps * (WidgetBasedTesting_Combinations + RandomMonkeyTesting_Combinations);

    Example:
       test_strategy=WidgetBasedTesting
   sel_stra=RandomBiasBased,FrequencyBased
   max_widgets=100,1000
   event_count=100,1000
   Assume number of apps provided = 2

    WidgetBasedTesting_Present = 1
Number_Of_Selection_Strategies = 2
Number_Of_Widget_Counts = 2

RandomMonkeyTesting_Present = 0
Number_Of_Event_Counts = 2

WidgetBasedTesting_Combinations = 1 * 2 * 2 = 4
RandomMonkeyTesting_Combinations = 0 * 2 = 0

Total Number of Test Profiles = 2 * (4 + 0) = 8

There will be total 8 Test profiles created and
you will find 8 different folders under specified working directory.
```

Chapter 7

Dynodroid Output

This chapter explains output of Dynodroid and its structure. There will be one folder for each of the test profile in the format: AppName_<Test_Strategy_Name>_<Selection_Strategy>_<No_of_Events>.
Each folder will has the following structure:

- **AppHandler**: This folder contains all the log files of app handler, for an app with sources this doesn't contain much info, but for apps provided as apk, this folder contains the extracted contents.
  - **ApkExtractDir**: for apps provided as apk, this contains the extracted (using apktool.jar) contents of the apk.
- **coverageHandler**: usually empty directory
- **CompleteTestProfile.log**: High level log of the status of the test profile. Contains brief description of the result of major steps.
- **TestStrategy**: This folder contains many useful and important logs regarding app behaviour.
  - **PreviousLogs**: Kernel logs and logcat contents after emulator start and before app install.
  - **MethodTraceFiles**: usually empty directory.
  - **SelectionStrategyLog.log**: This folder contains the log of the corresponding selection strategy, the various initialization, selection made at each event. etc.
  - **ODEX_Apk.dex**: The optimized dex file that got installed on the emulator for the app.
  - **GenerelLogCatLogs.log**: File containing logcat entries which are not filtered by atleast one of the monitoring clients, could be useful in finding exceptions etc.
  - **GenerelKernelLogs.log**: File containing kernel messages which are not filtered by atleast one of the monitoring clients.
  - **ResultMonkeyScript.txt**: The script containing all the actions performed by Dynodroid on to the emulator, this provides some sort of reply mechanism.
  - **ManifestInfo.txt**: This file contains the information of the app from AndroidManifest.xml in human readable form.

- **RunStats**: This folder contains coverage report of the app, the folder Coverage1, Coverage2 etc contains intermediate reports and the folder FinalCoverageStats contains the final coverage report at the end of the testing.
- **packages.list**: This file contains the list of the packages that are present in the device after app installation.
- **MonitoringLogs**: This folder contains the logs of various monitoring clients (each monitor the app for an activity) . Each of the below log files contain log entries along with the event which got triggered, for RandomMonkeyTesting these logs doesn't contain any events. These logs could be very useful especially for malware analysis.
  - **screenshots**: This folder contains png files of the widgets exercised and the corresponding screens.
  - **KernelNetwork_monitoring.log**: This file contains log entries for each of the major network event: open, listen and connect captured at kernel level.
  - **KernelFile_monitoring.log**: This file contains logs entries for each of the file opened only by the app under test, monitored at kernel level.
  - **OutboundUrl_monitoring.log**: This contains Urls being contacted by the app under test. This may not be the complete list of URLS contacted by the app.
  - **LOADLIB_monitoring.log**: The libraries being loaded by the app under test using JNI.
  - **SMSSend_monitoring.log**: The SMS messages being sent by the app under test.
  - **MethodsTrigerred.txt**: The methods of the app executed as part of the test run. This is very useful for apk apps , as emma coverage report cannot be generated for apks.
  - **DEXLOAD_monitoring.log**: The files which are loaded by using Dexload feature of android.
  - There are various other logs which are self explanatory by their name.