DroidBot

About

A robot which automatically interacts with Android apps.

DroidBot sends keyevent, gestures and simulates system events in order to exploit more app states automatically. DroidBot decides which actions to take based on static analysis result of app and dynamic device information (view hierarchy).

For more details, refer to my blog posts.

Introduction

DroidBot mainly does following two things:

1. Setting up device environments, include the contacts, SMS logs, call logs, GPS mocking, etc. The target app may have access to these resources, thus we prepare them before starting the app.

   Multiple env policies can be used for setting up environments. We support:

   - `none` policy which does not set up any environment;
   - `dummy` policy which just mocks same basic environment for all apps;
   - `static` policy which set up environment according to static information of app, for example permissions and files which the app have access to;
   - `file` policy which read environment configurations from a json file.

2. Sending events during the app is running. Events includes touch, drag gestures on screen, keyevents, and simulated broadcasts, etc.

   Similarly, we have several policies to produce events:

   - `none` policy which does not send any event;
   - `monkey` policy which make use of adb `monkey` tool, to produce randomized events;
   - `random` policy which sends randomized events to device
   - `static` policy produces a list of events based on static information of app. Eg. the intent-filters of each app.

    - `dynamic` policy. It is actually the real human-like policy. It monitors the device states, including the running activities, the foreground window, and the hierarchy of current window and sends events according to these information. It avoids going to same state too many times by comparing the window hierarchies, and it sends activity-specific intents based on static analysis of app.
    - `file` policy which generates events from a json file.

Prerequisite

1. `Python` version `2.7`
2. `Java` version `1.7`
3. `Android SDK`, make sure that `platform_tools` and `tools` added to `PATH`
4. (Optional) `DroidBox` version `4.1.1`, download from here

Installation

Clone this repo and use pip install:

```
git clone https://github.com/honeynet/droidbot.git
pip install -e droidbot
```

Usage

Basic Usage

1. Start an emulator or connect to a device using adb.
2. Start DroidBot: `droidbot -h`

Usage with DroidBox (without docker)

DroidBox print sensitive behaviours at runtime, which is useful in malware analysis. DroidBot can be used with DroidBox and is able to capture DroidBox logs.

Step 1. Start droidbox emulator:

1.1 Download DroidBox image

```
wget https://droidbox.googlecode.com/files/DroidBox411RC.tar.gz
tar xfz DroidBox411RC.tar.gz
```

1.2 Create an avd named droidbox

You can either use android avd manager or use `android create avd` command.
1.3 Start the avd with droidbox image

```
cd DroidBox411RC
sh startemu.sh droidbox
```

Step 2. Start DroidBot:

```
droidbot -a <sample.apk> -event dynamic -duration 100 -o droidbot_out
```

Usage with Docker

Prepare the environment on your host by creating a folder to be shared with the **DroidBot** Docker container. The folder will be used to load samples to be analyzed in **DroidBot**, and also to store output results from **DroidBot**analysis.

```
mkdir -p ~/mobileSamples/out
```

Now pull the ready-made Docker container (about 1.8 GB after extraction) from Honeynet Project's hub:

```
docker pull honeynet/droidbot
```

or, if you prefer, build your own from the GitHub repo:

```
git clone https://github.com/honeynet/droidbot.git
docker build -t honeynet/droidbot droidbot
```

To run the analysis, copy your sample to the folder you created above, then start the container; you will find results in the "out" subfolder.

```
cp mySample.apk ~/mobileSamples/
docker run -it --rm -v ~/mobileSamples:/samples:ro -v
~/mobileSamples/out:/samples/out honeynet/droidbot /samples/mySample.apk
ls ~/mobileSamples/out
```

Script

DroidBot supports semi-automatic testing. Users can write scripts to affect the process of testing.

The script is in json format, which contains three basic objects:

1. `View` selector, which can be used to select a view (aka. a UI component);
2. `State` selector, which can be used to select a state (such as a login Activity);
3. `Operation` object, which defines a set of events to be sent to device (such as screen-touching events).

An example of the DroidBot script is as follows:

```
{
    "views": {
        "login_email": {
            "resource_id": ".*email.*",
            "class": ".*EditText"
        },
        "login_password": {
            "resource_id": ".*password.*",
            "class": ".*EditText"
        },
        "login_button": {
            "resource_id": ".*login.*",
            "class": ".*Button"
        }
    },
    "states": {
        "login_state": {
            "activity": "LoginActivity",
            "views": ["login_email", "login_password", "login_button"]
        }
    },
    "operations": {
        "login_operation": {
            "operation_type": "custom",
            "events": [
                {
                    "event_type": "text_input",
                    "target_view": "login_email",
                    "text_content": "ylimit@honeynet.org"
                },
                {
                    "event_type": "text_input",
                    "target_view": "login_password",
                    "text_content": "ylimitpassword"
                },
                {
                    "event_type": "touch",
                    "target_view": "login_button"
                }
            ]
        }
    },
    "main": {
        "login_state": ["login_operation"]
    },
    "default_policy": "dynamic"
}
```

Explanation of the example:

- In `views`, we define the view selectors which will be used to select the views we are interested in. In this example, we define three views which are the email input view, password input view a the login button.
- In `states`, we define the states in which we want DroidBot to take different operations. In this example, we define a `login_state` which is a login screen waiting for users to input email and password. The `login_state` can be

recognized by checking the foreground activity name and the view on the screen.
- In `operations`, we define the operations which will be used in different states. In this example, we define a`login_operation` which is simply typing email, typing password and press login button.
- In `main`, we connect the states to corresponding operations. In this example, we let DroidBot to take`login_operation` in Login state, and use dynamic event policy in other states.

Evaluation

DroidBot is evaluated by comparing with DroidBot default mode (which does nothing) and adb Monkey tool. The results are in result.

Or see my visualized evaluation reports at DroidBot Posts.

Acknowledgement

1. AndroidViewClient is an amazing tool that simplifies test script creation.
2. Androguard is well-known for reverse-engineering of Android APKs.