Checking Information Misuse in DroidRacer Traces using RWFM Model

A R&D Report

Submitted in partial fulfillment of requirements for degree of

Masters of Technology

by

Asif Ali
Roll No : 143059009

under the guidance of

Prof. R.K. Shyamasundar

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

May, 2016

Contents

Abstract

Smartphone are become integral part of our life.This is possible because the smart-phone operating system allows third party application to use system API for completing its job. the major player in the smart-phone segment is ANDROID.which cover over 80 % of total smart-phone market share and got millions of application in its play store .However it also has its Dark side. There are thousands of malicious application in play store which can steal your credential and exploit the weakness of security policies in the ANDROID O.S. So in order to solve this we are using RWFM model to assure the android security .

Chapter 1

Introduction

Most of the android developers are using third party libraries in their applications. For example 50% of the free apps embed third party libraries (like advertisement libraries etc.). They may used by developers to provide various features at significant reduced time and cost. For example in app purchases, PDF view, cloud computing etc.

Many third party libraries access privacy-sensitive information even without notification to users or application developers. For example over the analysis of 1,00,000 android apps reveals that third party libraries covertly utilize Andrid APIs to access privacy-sensitive resources such as GET_ACCOUNTS, READ_PHONE_STATE or READ_CALENDAR without mentioning them properly in their Developer's Guide. The current Android platform provides coarse-grained controls for regulating whether third party libraries access private information, allowing them to operate with the same permission as their host apps.

In these days most of our daily work has been handled by applications. These applications handled different types of data. Thse data can be of differ sensitivity(belong to different security levels). The security mechainism working today are insufficient to overcome differentiate between these sensitivity. RWFM (Reader Writer Flow Model) offers a simple, intutive and systematic approach to track down the flow of information amongst the subjects in a system variety basis. We present an script which work on RWFM to track down readers and writers of information very explicitly. Our script tracks down flow of information using explictly labelling each entiy in a communication(usually in form of subjects and objects).

Chapter 2

Reader Writers Flow Model

In this section, we introduce RWFM model,which is novel model for information flow control. RWFM is obtained by recasting the Denning's label model.

Structure of Label RWFM can be definded as five tuples $(S, O, S \times 2^{\mathrm{s}} \times 2^{\mathrm{s}}, (*, \cap, \cup), (*, \subseteq, \supseteq))$, where S and O denote the set of subjects and objects in the information system respectively.

RWFM describes following operations under which operation is as follows:

READ RULE

Subject s with label (s_1, R_1, W_1) requests read access to an object o with label (s_2, R_2, W_2).

$if(s \in R_2)then$

change the label of s to $(s_1, R_1 \cap R_2, W_1 \cup W_2)$

ALLOW

ELSE

DENY

WRITE RULE

Subject s with label (s_1, R_1, W_1) requests write access to an object o with label (s_2, R_2, W_2).

$if(s \in W_2 \wedge R_1 \supseteq R_2 \wedge W_1 \subseteq W_2)then$

ALLOW

ELSE

DENY

DOWNGRADE RULE

Subject s with label (s_1, R_1, W_1) requests to dwongrade an object o with label(s_2, R_2, W_2) to (s_3, R_3, W_3)

$if(s \in R_2 \wedge s_1 = s_2 = s_3 \wedge R_1 = R_2 \wedge W_1 = W_2 = W_3 \wedge R_1 \subseteq R_3 \vee (W_1 = s_1 \vee (R_3 - R_2 \subseteq W_2)))then$

ALLOW

ELSE

DENY

CREATE RULE

Subjcect s with label (s_1, R_1, W_1) requests to create an object O. Create a new object O, label it as (s_1, R_1, W_1) and add it to the set of objects O.

Chapter 3

Message Sequence Chart

A message sequence chart (or MSC) is an interaction diagram from the SDL family standardized by the International Telecommunication Union.

The purpose of recommending MSC (Message Sequence Chart) is to provide a trace language for the specification and description of the communication behaviour of system components and their environment by means of message interchange. Since in MSCs the communication behaviour is presented in a very intuitive and transparent manner, particularly in the graphical representation, the MSC language is easy to learn, use and interpret. In connection with other languages it can be used to support methodologies for system specification, design, simulation, testing, and documentation.
Below is an example of MSC

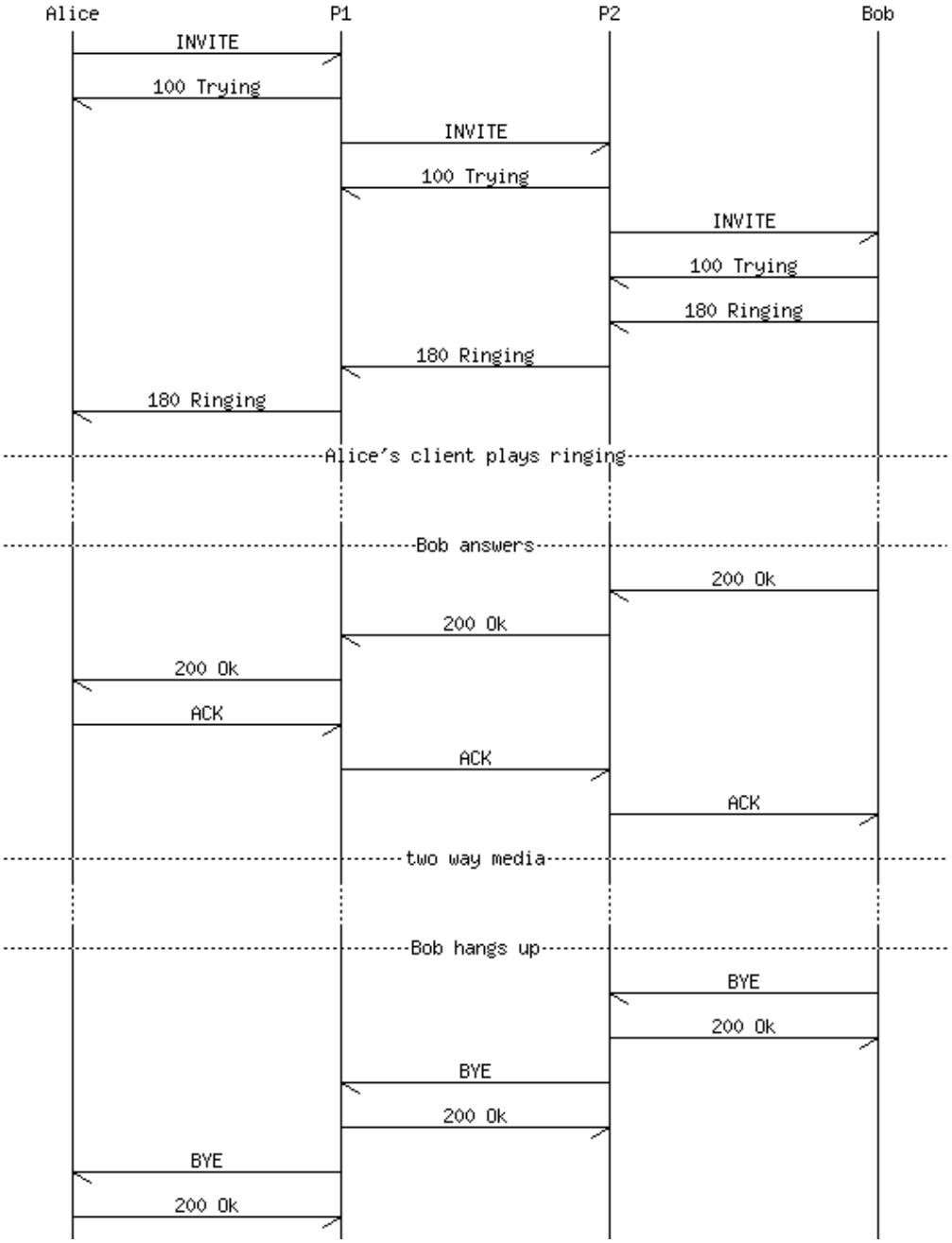

Figure 3.1: Figure showing Message Sequence Chart

Chapter 4

DroidRacer

DROIDRACER provides a framework that generates UI events to systematically test an Android application.It runs unmodified binaries on an instrumented Dalvik VM and in-strumented Android libraries. A run of the application produces an execution trace, which is analyzed offline for data races by computing the happens-before relation. The control flow between different procedures of an Android application is managed to a large extent by the Android runtime through callbacks. DROIDRACER uses a model of the Android runtime environment to reduce false positives that would be reported otherwise. Further, DROIDRACER assists in debugging the data races by classifying them based on criteria such as whether one involves multiple threads posting to the same thread or two co-enabled events executing in an interleaved manner.

4.1 Modelling the runtime environment

While we do not explicitly model ActivityManagerService (running in the system process) in the execution trace, we capture its effects through the enable operations on callback procedures. This helps us identify the ordering constraints for lifecycle callbacks made by the Android environment.Through enable operations,we also capture the ordering between operations in the trace and UI callbacks.The user action results in the activity being removed from the screen (but not garbage-collected), and ActivityManagerService posts a call to the onDestroy callback. This callback executes next writes into the field isActivityDestroyed. Due to the happens-before ordering between enable and post and post and begin there is a happens-before edge between the write operations 7 and 21 and they do not constitute a data race. Without the enable operation, which specifies the environment restriction that onDestroy can only be called after LAUNCH_ACTIVITY finishes, we could not have derived the required happens-before ordering between operations 7 and 21 , resulting in a false positive.

4.2 Analysis for data races

Again, we consider the trace in Figure 4.1 The callback onDestroy is enabled while thread t2 is running and if fired, it executes on thread t1. Thus, the read and write operations 12 and 21 may happen in parallel giving rise to a potential data race.

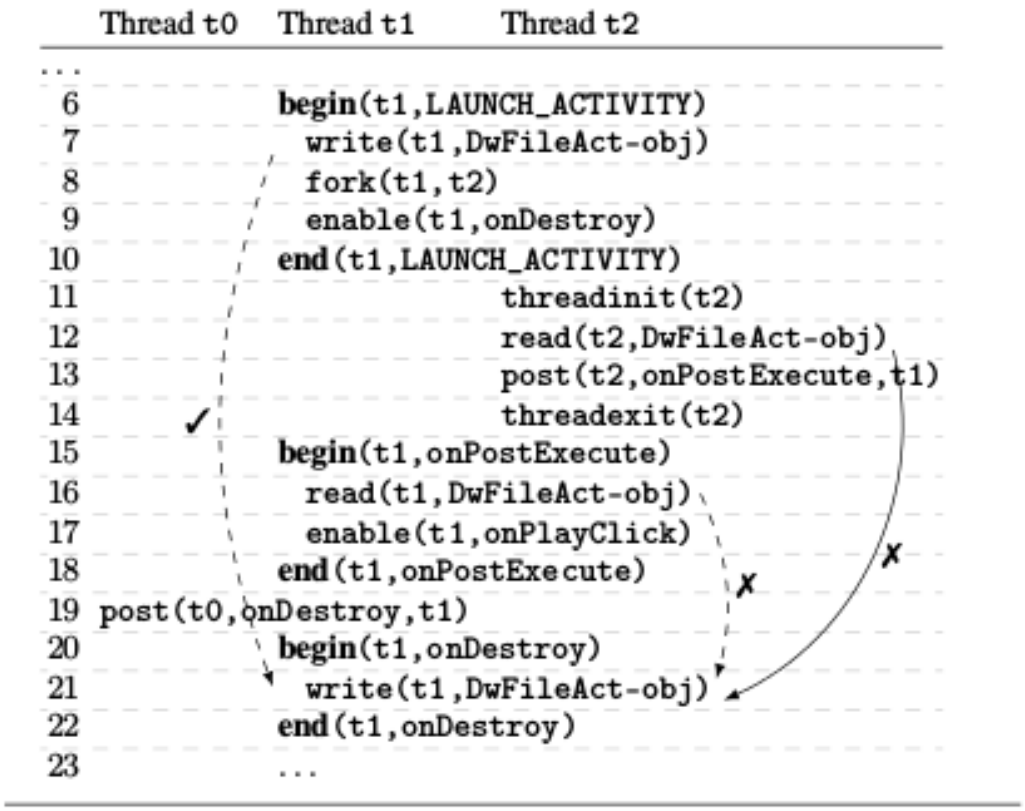

Figure 4.1: Partial exection trace for the scenario in which user clicks the BACK button instead of PLAY button

Chapter 5

Idea and Approach

We are using the droidracer traces to analyze information misuse in android apps. We try to convert traces obtained from DroidRacer into message sequence chart. For that we follows the following steps:

- **Separation** We separate out stack traces on the basis of tid's or thread ids and rename as like parent_child.txt file. We found out different events from log files an dfound out there are number of events (such as: ADD_IDLE_HANDLER, ENABLE_LIFECTCLE,ENABLE_WINDOW-FOCUS,INSTANCE-INTENT,QUEUE_IDLE, REMOVE_IDLE_HANDLER etc.) which are not of our corns beacuse they doesn't cause any misuse of objects as given in figure Sheet1 in excel sheet.

- **Removal of Unnecessary Events** We remove all these non concerned events from our event set and map it to the operations given in droid racer paper. After that we label different objects and thread under these operations (Sheet2 in excel sheets).

- **Parsing** Parsed each file starting from 1.txt because thread 1 is the main thread in the following ways:
  READ Read instructions in the trace file:

  $$rwId : 1READtid : 1obj : 0x411253f8class : Ljava/lang/StringBuilder; field : 16$$

  can be changed into

  $$READ(1)$$

  given that we map obj:0x411253f8 field:16 into some integer id.
  WRITE Write instructions in trace file:

  $$rwId : 5WRITEtid : 1obj : 0x411253f8class : Ljava/lang/StringBuilder; field : 16$$

  can be changed into

  $$WRITE(1)$$

  given that we map obj:0x411253f8 field:16 into some integer id.
  FORK Fork instructions in trace file:

  $$30FORKpar - tid : 1child - tid : 6$$

  can be change into

  $$FORK-->THREADINIT$$

Operations found in Log file	Operations giv	REMARKS
ACCESS	read/write	Group continous read-writes operations without any intermediate concurrency operation into a single ACCESS operati
ADD_IDLE_HANDLER		
ATTACH-Q	attachQ(t)	
CALL	begin(t,p)	Method Call
ENABLE-EVENT	enable(t,p)	
ENABLE-LIFECYCLE		To enable component lifecycle
ENABLE-WINDOW-FOCUS		To enable window focus
ENTRY		
EXIT		
FORK	fork(t,t')	
INSTANCE-INTENT		
LOCK	acquire(t,l)	
LOOP	looponQ(t)	
NATIVE-ENTRY	threadinit(t)	serves as delimiters whenever a non-tracked thread(eg., binder thread,natively created thread) performs POST operati
NATIVE-EXIT	threadexit(t)	same as above
NOTIFY		Notify other thread
POST	post(t,p,t')	
QUEUE_IDLE		
READ	read(t,m)	
READ-STATIC	read(t,m)	
REMOVE_IDLE_HANDLER		
RET	end(t,p)	Method Return
THREADEXIT	threadexit(t)	
THREADINIT	threadinit(t)	
TRIGGER-BROADCAST		used to model events/intents and callbacks from environment.
TRIGGER-EVENT		used to model events/intents and callbacks from environment.
TRIGGER-LIFECYCLE		used to model events/intents and callbacks from environment.
TRIGGER-SERVICE		used to model events/intents and callbacks from environment.
TRIGGER-WINDOW-FOCUS		used to model events/intents and callbacks from environment.
UNLOCK	release(t,l)	
WAIT		waiting of thread
WRITE	write(t,m)	
WRITE-STATIC	write(t,m)	

Figure 5.1: All event found in trace file

Operations found in Log file	Operations given in pape	Label tracking notes
ACCESS	read/write	
ATTACH-Q	attachQ(t)	
CALL	begin(t,p)	label of t = label of t + label of message
ENABLE-EVENT	enable(t,p)	
FORK	fork(t,t')	label of t' = label of t, owner = t', add t' to readers and writers
LOCK	acquire(t,l)	label of obj = label of t = label of t + label of obj
LOOP	looponQ(t)	
NATIVE-ENTRY	threadinit(t)	no impact on label
NATIVE-EXIT	threadexit(t)	
NOTIFY	notify(t,t')	label of t' = label of t + label of t'
POST	post(t,p,t')	label of p = label of t, add t' to readers
READ	read(t,m)	label of t = label of t + label of obj
READ-STATIC	read(t,m)	label of t = label of t + label of obj
RET	end(t,p)	
THREADEXIT	threadexit(t)	
THREADINIT	threadinit(t)	no impact on label
UNLOCK	release(t,l)	
WAIT	semaphore wait	no impact on label
WRITE	write(t,m)	label of obj = label of t + label of obj
WRITE-STATIC	write(t,m)	label of obj = label of t + label of obj

Figure 5.2: Necessary Events for Analysis

such that arrow from tid:1 pointed to tid:6.Thread id's are already arranged at the start of the file. Then both the parent and child id's read their own files one by one.
WAIT/NOTIFY WAIT and NOTIFY instructions in stack trace file can be given as: $WAITtid : 1/NOTIFYtid : 6notifiedTid : 1$ So if the any file contains WAIT call then we have to wait for other file NOTIFY instruction such that it notifies the thread that is waiting on that for the later thread.Therefore, it can be changed into:

$WAIT<----------$

means some thread(n+i) notifies thread(n).
LOCK/UNLOCK LOCK/UNLOCK instructions in trace file:

$$41LOCKtid : 1lock - obj : 0x41128b90/48UNLOCKtid : 1lock - obj : 0x4105e8d0$$

Lock/Unlock instructions used for synchronisation puroposes can be written as LOCK(2) / UNLOCK(2) givent that lock-obj can be mapped to some integer.
POST/CALL POST/CALL instructions in trace file:

$$54POSTsrc : 2msg : 6dest : -1delay : 0/162CALLtid : 1msg : 3$$

So if the any file contains POST call then we have to wait for other file CALL instruction such that it notifies the thread that is waiting on that for the later thread.

$POST(13)----------$

means some thread(n) post call to some thread(n+i) giving some object which mapped to 13 integer internally.

•**Merging** Merged two consecutive same event (READ/WRITE) on the same object into one event(READ/WRITE).

•**Labelling** Labelled out the resultant file obtained after merging and check out any misuse using the rules given by RWFM above.

Chapter 6

Experimentation and Output

We chose twitter application traces for examination.Here are the screenshots of different files given below:
Trace file from DroidRacer:

Message Sequence Chart:
After Similar Call Merging:
Resultant File with labelling:
OUTPUT

There are some scenarios are there such that there is data race in DroidRacer traces and no misuse occurs by using our RWFM model.

```
1 START
2 THREADINIT tid:1
3 ENABLE-LIFECYCLE tid:1 component: id:0 state:BIND-APP
4 ATTACH-Q tid:1          queue:1090953608
5 NATIVE-ENTRY tid:2thread-name:Binder Thread #1
6 POST src:2 msg:1 dest:-1 delay:0
7 NATIVE-EXIT tid:2thread-name:Binder Thread #1
8 NATIVE-ENTRY tid:2thread-name:Binder Thread #1
9 POST src:2 msg:2 dest:-1 delay:0
10 NATIVE-EXIT tid:2thread-name:Binder Thread #1
11 NATIVE-ENTRY tid:2thread-name:Binder Thread #1
12 POST src:2 msg:3 dest:-1 delay:0
13 NATIVE-EXIT tid:2thread-name:Binder Thread #1
14 LOOP tid:1    queue:1090953608
15 CALL tid:1    msg:1
16 RET tid:1     msg:1
17 CALL tid:1    msg:2
18 TRIGGER-LIFECYCLE tid:1 component: id:0 state:BIND-APP
rwId:1 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:2 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:3 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:4 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:5 WRITE tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:6 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:7 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:8 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:9 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:10 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:11 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:12 WRITE tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:13 WRITE tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:12
rwId:14 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
rwId:15 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:16 WRITE tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:17 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:16
rwId:18 READ tid:1 obj:0x411253f8 class:Ljava/lang/StringBuilder; field:8
```

Figure 6.1: Trace File from DroidRacer

```
Global Variables: 59 97 56 58 65 34 89 29 20 28 83 39 90 68 75 60 3 15 67 87 50 71 62 93 92 31 43 25 41 13 24 72 14 55 94 69 100 8 64 47 27 48 16 91 82
Lock Varaibles: 10 9 23 27 11 21 22 19 25 13 17 12 14 26 3 24 4 20 8 16 7 6 15 2 5 1 18
tid:1          tid:2          tid:5          tid:6          tid:7          tid:8          tid:9          tid:10         tid:11         tid:12         t
THREADINIT
READ(1)
READ(2)
READ(2)
READ(1)
WRITE(1)
READ(1)
READ(2)
READ(2)
READ(2)
READ(2)
READ(1)
WRITE(2)
WRITE(3)
READ(2)
READ(1)
WRITE(1)
READ(1)
READ(2)
READ(1)
READ(2)
READ(2)
READ(2)
READ(1)
WRITE(2)
WRITE(3)
READ(2)
READ(1)
WRITE(1)
READ(1)
READ(2)
READ(1)
READ(1)
READ(2)
```

Figure 6.2: Message Sequence Chart for Twitter app trace

```
Global Variables: 59 97 56 58 65 34 89 29 20 28 83 39 90 68 75 60 3 15 67 87 50 71 62 93 92 31 43 25 41 13 24 72 14 55 94 69 100 8 64 47 27 48 16 91 82
Lock Varaibles: 10 9 23 27 11 21 22 19 25 13 17 12 14 26 3 24 4 20 8 16 7 6 15 2 5 1 18
tid:1          tid:2          tid:5          tid:6          tid:7          tid:8          tid:9          tid:10         tid:11         tid:12         t
THREADINIT
READ(1)
READ(2)
READ(1)
WRITE(1)
READ(1)
READ(2)
READ(1)
WRITE(2)
WRITE(3)
READ(2)
READ(1)
WRITE(1)
READ(1)
READ(2)
READ(1)
READ(2)
READ(1)
WRITE(2)
WRITE(3)
READ(2)
READ(1)
WRITE(1)
READ(1)
READ(2)
READ(1)
READ(2)
READ(1)
READ(4)
READ(5)
READ(4)
WRITE(5)
WRITE(6)
```

Figure 6.3: Merged File

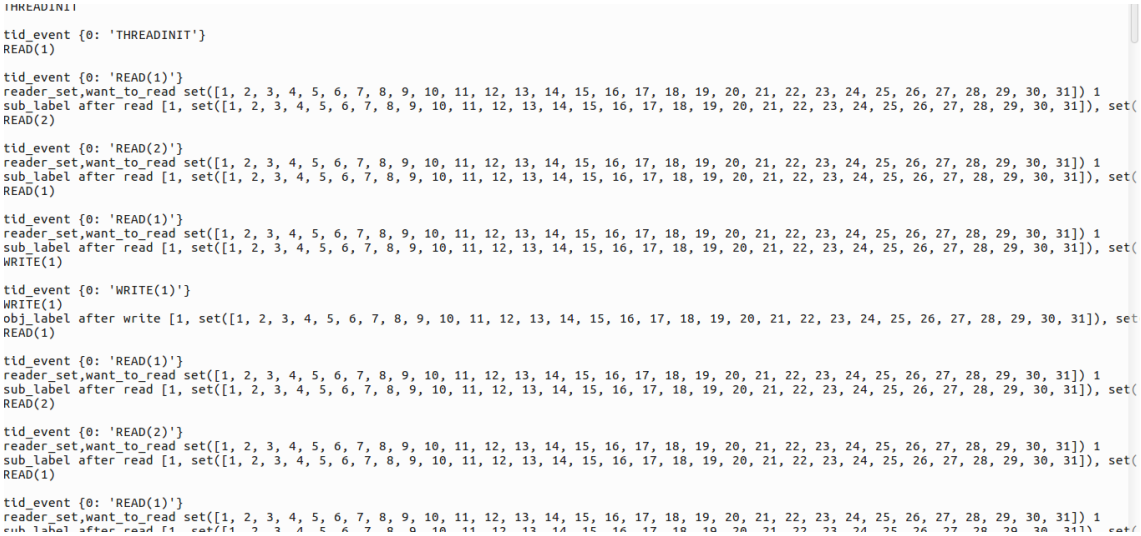

Figure 6.4: Labelled file

Chapter 7

Future Work

- Checking all the possible combination of RWFM on a parallel.
- Comparing the result of DroidRacer with RWFM misuse.