Android Fuzzing Tool
CSE509 Final Project Report

Haolong Ning
Yubin Lai
Ruonan Hao

December 17, 2013

Date Performed: November 20, 2013
Instructor: Professor Rob Johnson

1 Objective

To determine the atomic weight of magnesium via its reaction with oxygen and to study the stoichiometry of the reaction (as defined in **??**):

$$2\,\mathrm{Mg} + \mathrm{O_2} \longrightarrow 2\,\mathrm{MgO}$$

1.1 Definitions

Stoichiometry The relationship between the relative quantities of substances taking part in a reaction or forming a compound, typically a ratio of whole integers.

Atomic mass The mass of an atom of a chemical element expressed in atomic mass units. It is approximately equivalent to the number of protons and neutrons in the atom (the mass number) or to the average number allowing for the relative abundances of different isotopes.

2 Experimental Data

Mass of empty crucible	7.28 g
Mass of crucible and magnesium before heating	8.59 g
Mass of crucible and magnesium oxide after heating	9.46 g
Balance used	#4
Magnesium from sample bottle	#1

3 Sample Calculation

Mass of magnesium metal	= 8.59 g - 7.28 g
	= 1.31 g
Mass of magnesium oxide	= 9.46 g - 7.28 g
	= 2.18 g
Mass of oxygen	= 2.18 g - 1.31
	= 0.87 g

Because of this reaction, the required ratio is the atomic weight of magnesium: 16.00 g of oxygen as experimental mass of Mg: experimental mass of oxygen or $\frac{x}{1.31} = \frac{16}{0.87}$ from which, $M_{\mathrm{Mg}} = 16.00 \times \frac{1.31}{0.87} = 24.1 = 24$ g/mol (to two significant figures).

4 Design

we create a multi-thread program that recursively track build inner object type by Java reflection technique. The design included following three parts
:

4.1 Part I

Program will start from here (Part I), it will generate multiple thread to execute Part II and Part III. On our program, we use set up the maxium number of thread to be 20, as the speed of Android emulator tested in our environment is limited. Of course, android fuzzer is initially design for much number (2000 thread or more) as long as the hardware resource is powerful enough.

4.2 Part II

After program generate a thread, it will goes into second part (Part II). Each thread, will repeat Part II and Part III exactly same. The more thread we generate, the more randomlazion we have, and more possible to find out bugs among API. For each thread will fall into following stages:

4.2.1 Stage I

Get a package name (API) from user. We designed in this way that it's user's resonpsiblity to provide the correct API. Sepecifically, we chose **`Java.io.*`** as our testing package. We maintained a table for all the classes inside this package:

`Java.io.BufferedInputStream`	**`Java.io.BufferedOutputStream`**
`Java.io.BufferedReader`	**`Java.io.BufferedWriter`**
`Java.io.ByteArrayInputStream`	**`Java.io.ByteArrayOutputStream`**
`Java.io.CharArrayReader`	**`Java.io.CharArrayWriter`**
`Java.io.Console`	**`Java.io.DataInputStream`**
`Java.io.DataOoutputStream`	**`Java.io.File`**
`Java.io.File`	**`Java.io.FileDescriptor`**
`Java.io.FileInputStream`	**`Java.io.FileOutputStream`**
`Java.io.FilePermission`	**`Java.io.FileReader`**
`Java.io.FileWriter`	**`Java.io.InputStream`**
`Java.io.InputStreamReader`	**`Java.io.LineNumberInputStream`**
`Java.io.LineNumberReader`	**`Java.io.ObjectInputStream`**
`Java.io.ObjectInputStream.GetField`	**`Java.io.ObjectOutputStream`**
`Java.io.ObjectOutputStream`	**`Java.io.ObjectOutputStream.putField`**
`Java.io.ObjectStreamClass`	**`Java.io.ObjectStreamField`**
`Java.io.OutputStream`	**`Java.io.OutputStreamWriter`**
`Java.io.PipedInputStream`	**`Java.io.PipedOutputStream`**
`Java.io.PipedReader`	**`Java.io.PipedWriter`**
`Java.io.PushBackInputStream`	**`Java.io.PushBackReader`**
`Java.io.RandomAccessFile`	**`Java.io.Reader`**
`Java.io.SequenceInputStream`	**`Java.io.SeriallzeablePermission`**
`Java.io.StreamTokenizer`	**`Java.io.StringBufferInputStream`**
`Java.io.StringReader`	**`Java.io.StringWriter`**
`Java.io.Writer`	

However, for further work, it should be available for any package. As the number of package increased, the size of table will also increased respectfully. Since the table of classes are maintained manually and time constrained, we first target on **`Java.io.File`** sepecifically. For completely version, It will contain all classes.

4.2.2 Stage II

Randomly pick up a class that inside the selected package, take **`Java.io.File`** as example, create a the corresponding object for this class. Each class contains one or more than one constructors:

```
File(File parent, String child)
File(String pathname)
File(String parent, String child)
File(URI uri)
```

We used Java reflection to get all these constructors and randomly pick up one of them, Though this constructor, we initailize a instance (Object) for this class.

```
...
Class c = c.forName(Java.io.File);
Constructors cons = c.getConstructors();
Constructor constructor = cons[random(cons.size() - 1];
Object obj = constructor.getInstance();
...
```

After we initialized a instance of this class, we can first get all the methods associated with this class, and then randomly pick one to test:

```
...
Method[] methods = obj.getMethods();
Method method = methods[random(methods.size() - 1];
method.invoke(...);
...
```

upon this point, we finished Stage II, that pick up one path to test among mutilple choice by reflection.

4.2.3 Stage III

It is very common that to initialize one class or one method need another class (object). We recursively repeat stage I and stage II to get instance of those class in order to invoke the method successfully. For example, a method under **`Java.io.File:`**

```
int compareTo(File pathname)
```

To test **`compareTo`** method, we have to first pass a **`File`** object. Repeat Stage I and Stage II:

1) get constructors of **`File`** Class;
2) randomly pick up one among those constructors;
3) create a new instance through this constructor;
4) and finally pass this new instance to the method.

Since to get a real instance, we may have to keep repeating Stage I and Stage II mutilple time, we build method recusrively handle those steps until we hit the basic type. In order words, this recursively call won't terminate until we hit the basic type, and this will be the end of Part II.

Part III So far, the program will be able to generate a thread to randomly pick a method to test, based on a randomly chosen class, and it will recursively initailize new instance type until it hit the basic type. After one thread hit the basic type, program will goes into another path, that pick up a value from pre-defined table and feed it back to the caller. Those basic type defined as following types: **`String, int, double, float, short, byte, long, boolean, File, URI`** for testing **`Java.io.File`** class, sepecifically. For numerical type, our random value will come from 5 kind of specific case: Maxium positive value, minium positive value, 0, minium negative value, maxium negative value. Take **`double`** as example, those values will be:

```
+Double.getMaxValue()   +Double.getMinValue()   0
-Double.getMinValue()   -Double.getMaxValue()
```

Above case including: **int**, **double**, **float**, **short**, **byte**, **long**. For type **String** we handled following cases:

```
// one
str = "1----/";
strList.add(str);

str = "$PATH$:/";
strList.add(str);

str = "\64\2\67\34/";
strList.add(str);

str = "--.-%DIR%-/";
strList.add(str);

str = "----/";
strList.add(str);

str = "./.";
strList.add(str);

str = "-/";
strList.add(str);

str = "\01\01\01\01\01\01\01\01\01\01\01\01";
strList.add(str);

str = "\00\00\00\00";
strList.add(str);

str = "\177\177\177\177\177\177\177\177\177";
strList.add(str);

str = "######";
strList.add(str);

str = "\140echo␣1";
strList.add(str);

str = "\140;echo␣1";
strList.add(str);

str = "ls␣*sjgl";
strList.add(str);

str = "&df␣";
strList.add(str);

str = "./---␣/";
strList.add(str);

str = "--";
strList.add(str);

str = "%n%s%d";
strList.add(str);

str = "--10101/";
strList.add(str);

str = "ls␣-";
strList.add(str);

str = "er␣&&␣";
strList.add(str);

str = "--../--.";
strList.add(str);

str = "||";
strList.add(str);

str = "&&for␣!";
strList.add(str);

str = "0XFFFFFFFF";
strList.add(str);

str = "0X00000000";
strList.add(str);

str = "0X--␣0X";
strList.add(str);

str = "0XFF0101110";
strList.add(str);

str = "0X&er111";
strList.add(str);
```

Since we focus on **Java.io.File** as our intial testing case, we also create treat **File** as our basic type. We generate lots of file at beginning and store them in a array. We build methods for help us handle this, each method will response for one type of file:

```
this.gen_LongNameFile();
this.gen_MutipleLevelFile();
this.gen_ReadonlyFile();
this.gen_WriteonlyFile();
this.gen_ExcuteonlyFile();
this.gen_noRWXFile();
this.gen_BigFile();
this.gen_EmptyFile();
this.gen_RootFile();
this.gen_LongNameDir();
this.gen_MutipleLevelDir();
```

```
this.gen_ReadonlyDir();
this.gen_WriteonlyDir();
this.gen_ExcuteonlyDir();
this.gen_noRWXDir();
this.gen_RootDir();
this.gen_WeirdNameDir();
this.gen_WeirdNameFile();
```

For most of them is easy to understand from their name, same explain below:

gen_MutipleLevelFile()\gen_MutipleLevelDir() These two methods will create a file or directory within 15 level: **level_1\level_2\...\level _15\file** .

gen_WeiredNameFile()\gen_WeiredNameDir() These two methods will create a file or directory with weird name:

```
dir = absDir + "1----";
dir = absDir + "$PATH$:";
dir = absDir + "\64\2\67\34";
dir = absDir + "----";
dir = absDir + "--.-%DIR%-";
dir = absDir + "%DIR%-";
dir = absDir + ".";
dir = absDir + "-";
dir = absDir + "\01\01\01\01\01\01\01\01\01\01\01\01";
dir = absDir + "\00\00\00\00";
dir = absDir + "\177\177\177\177\177\177\177\177\177";
dir = absDir + "######";
dir = absDir + "\140echo␣1";
dir = absDir + "\140;echo␣1";
dir = absDir + "\140echo␣1";
```

d

5 Discussion of Experimental Uncertainty

The accepted value (periodic table) is 24.3 g/mol [**?**]. The percentage discrepancy between the accepted value and the result obtained here is 1.3%. Because only a single measurement was made, it is not possible to calculate an estimated standard deviation.

The most obvious source of experimental uncertainty is the limited precision of the balance. Other potential sources of experimental uncertainty are: the reaction might not be complete; if not enough time was allowed for total oxidation, less than complete oxidation of the magnesium might have, in part, reacted with nitrogen in the air (incorrect reaction); the magnesium oxide might have absorbed water from the air, and thus weigh "too much." Because the result obtained is close to the accepted value it is possible that some of these experimental uncertainties have fortuitously cancelled one another.

6 Answers to Definitions

a. The *atomic weight of an element* is the relative weight of one of its atoms compared to C-12 with a weight of 12.0000000. . ., hydrogen with a weight of 1.008, to oxygen with a weight of 16.00. Atomic weight is also the average weight of all the atoms of that element as they occur in nature.

b. The *units of atomic weight* are two-fold, with an identical numerical value. They are g/mole of atoms (or just g/mol) or amu/atom.

c. *Percentage discrepancy* between an accepted (literature) value and an experimental value is $\frac{|\mathrm{experimentalresult}-\mathrm{acceptedresult}|}{\mathrm{acceptedresult}}$.