# Replication Package Documentation
CLOTHO: Saving Programs from Malformed Strings and Incorrect String-handling

Aritra Dhar[1], Rahul Purandare[2], Mohan Dhawan[3] and Suresh R[2]

[1]Xerox Research Center India
[2]IIIT Delhi
[3]IBM Research and IBM Research

June 7, 2015

## 1 Clotho Repository

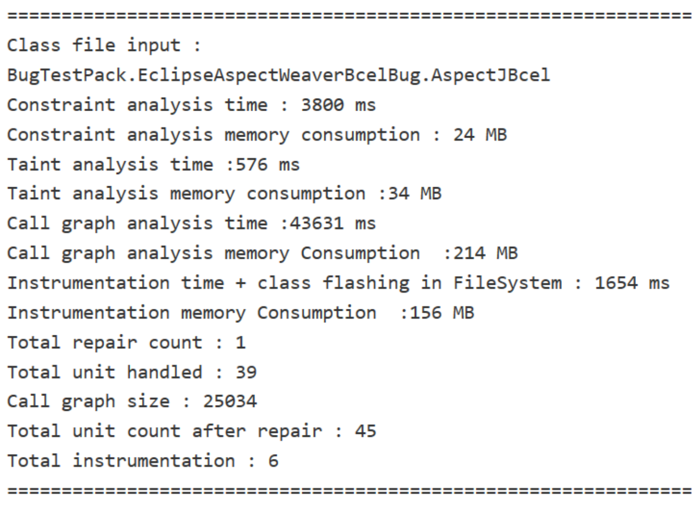
The source code of Clotho is located two github public repository : `https://github.com/aritradhar/CLOTHO` and `https://github.com/aritradhar/CLOTHO-TaintAnalysis`. The first one contains the source code of core Clotho framework and the later one contains the source code of the taint analysis framework used by CLOTHO. Both of these repositories contains Eclipse projects and can be import and build in Eclipse IDE easily. Dependency jar files are located at `https://github.com/aritradhar/CLOTHO-TaintAnalysis/tree/master/BuildPathClass` and should be added to eclipse build path. Project CLOTHO-Taint analysis (`https://github.com/aritradhar/CLOTHO-TaintAnalysis`) must be also included in the project path to built it properly.

## 2 Log File Description

Detailed result can be found in the file located at `https://github.com/aritradhar/CLOTHO/blob/master/log.txt`. One example log is given in the figure 1.

Each log entry contains following fields describing the patching information and performance.

1. **Class file input** : Input class file/path to the project which is to be repaired.
2. **Constraint analysis time** : Total time taken to perform static constraint collection and analysis. This value also includes the overhead of `Soot`.



```
================================================================
Class file input :
BugTestPack.EclipseAspectWeaverBcelBug.AspectJBcel
Constraint analysis time : 3800 ms
Constraint analysis memory consumption : 24 MB
Taint analysis time :576 ms
Taint analysis memory consumption :34 MB
Call graph analysis time :43631 ms
Call graph analysis memory Consumption  :214 MB
Instrumentation time + class flashing in FileSystem : 1654 ms
Instrumentation memory Consumption  :156 MB
Total repair count : 1
Total unit handled : 39
Call graph size : 25034
Total unit count after repair : 45
Total instrumentation : 6
================================================================
```

Figure 1: Example of Clotho output

3. **Constraint analysis memory consumption** : Total memory consumption of the static constraint collection and analysis.

4. **Taint analysis time** : Total time taken by the taint analysis phase. This also includes the overhead by the `Soot InfoFlow` tool.

5. **Taint analysis memory consumption** : Total memory consumption by the taint analysis phase.

6. **Call graph analysis time** : Time taken to produce and analyze the call graph. The time is dependent on the type of algorithm used in `Soot`'s call graph analysis phase. We have used `CHA` which is less precise but faster algorithm compared to `SPARK` which provides most accurate.

7. **Call graph analysis memory Consumption** : Memory consumption of the call graph generation and analysis phase.

8. **Instrumentation time + class flashing in FileSystem** : Total time taken by the instrumentation phase i.e. application of the path in the byte code. This phase is also responsible to flush the instrumented class file in the file system.

9. **Instrumentation memory Consumption** : memory consumption by the instrumentation phase.

10. **Total repair count** : Total number of instrumentation points discovered by Clotho.

11. **Total unit handled** : Total number of `Units` i.e. `Jimple` statements produce by `Soot` analyzed by Clotho.

12. **Call graph size** : Size of the call graph generated in the CLOTHOcall graph generation and analysis phase.

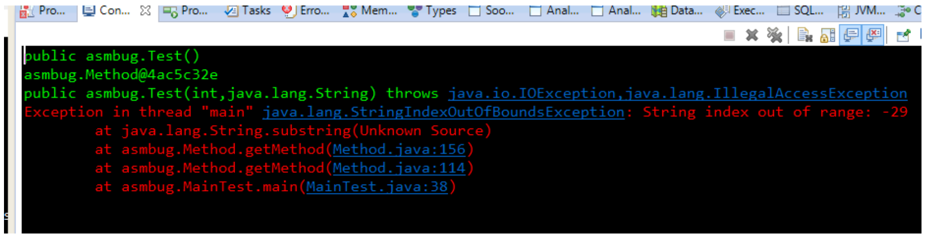
13. **Total unit count after repair** : Total number of *Unit*s in the instrumented class file.

14. **Total instrumentation** : Total number of `Units` instrumented.

## 3 Project Structure

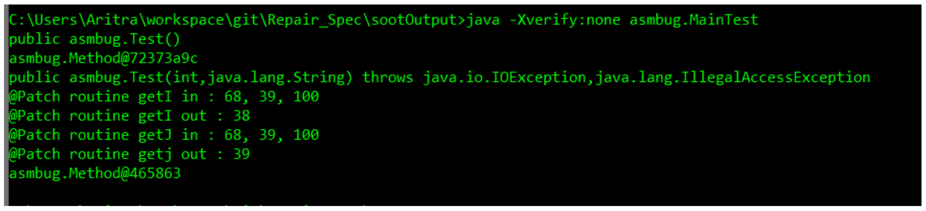
The entry point of the project is `constraintAnalysis.Driver`. The class `util.ENV` contains all the configurations to tune CLOTHO. To be able to analyze all the given test cases, `SOOT_CLASS_PATH` should be configured properly. It contains the location of required jar files. The package `BugTestPack` contains the source code of the bugged library. To replicate it easily, we have added the portion of the method which is bugged. The entire compiled library as a jar file can be passed to CLOTHObut it will take time to process. We have commented the variable `className`. The reviewers can uncomment any of the lines one at a time to patch the bugged library. After the successful execution, the patched version will be flushed in a folder named `sootOutput` in the Eclipse project directory. To check if the patch version of the class file is working or not run with with `java` command. In case the execution throws some security error, use `Xverify:none` option to skip bytecode verification check by the JVM.

One such execution screen shot is given in the images 2 and 3. The former one shows the bug in the `Apache ASM` library and the later one shows after patching execution with some diagnostic message. The `getI` and `getJ` shows the indexes passed in `subString()` methods and the patched indexes. The detailed version of the bug can be found at `https://issues.apache.org/jira/browse/ARIES-1204`


Pro... Con... Pro... Tasks Erro... Mem... Types Soo... Anal... Anal... Data... Exec... SQL... JVM...

```
public asmbug.Test()
asmbug.Method@4ac5c32e
public asmbug.Test(int,java.lang.String) throws java.io.IOException,java.lang.IllegalAccessException
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: -29
	at java.lang.String.substring(Unknown Source)
	at asmbug.Method.getMethod(Method.java:156)
	at asmbug.Method.getMethod(Method.java:114)
	at asmbug.MainTest.main(MainTest.java:38)
```

Figure 2: Befor patching Apache ASM



```
C:\Users\Aritra\workspace\git\Repair_Spec\sootOutput>java -Xverify:none asmbug.MainTest
public asmbug.Test()
asmbug.Method@72373a9c
public asmbug.Test(int,java.lang.String) throws java.io.IOException,java.lang.IllegalAccessException
@Patch routine getI in : 68, 39, 100
@Patch routine getI out : 38
@Patch routine getJ in : 68, 39, 100
@Patch routine getj out : 39
asmbug.Method@465863
```

Figure 3: Afetr patching Apache ASM