Testing Infer Quandary as a security tool via confusion matrix perturbation analysis

ICT Risk Assessment final project

Giovanni Bartolomeo and Laura Bussi

1 Introduction

When dealing with security issues, we are provided a pletora of tools for the analysis of possible vulnerabilities in software. Many of them analyse the behaviour of the software at run time, but static analysis can also be performed: several tools can analyse source code and find possible flaws before the program is running.
Of course, both this kind of approaches cannot be 100% accurate. Most likely they will provide as result a set of possible vulnerabilities which intersect the set of the actual ones, i.e. for each pointed out vulnerability we will have four possible cases:

- **True Positive** Tool correctly identifies a real vulnerability
- **False Negative** Tool fails to identify a real vulnerability
- **True Negative** Tool correctly ignores a false alarm
- **False Positive** Tool fails to ignore a false alarm

From this classification, we obtain a 2×2 matrix, namely a confusion matrix. Confusion matrix can be a useful tool to benchmark a security analysis tool capabilities.

1.1 OWASP Benchmark

The OWASP benchmark project is an executable web application, provided as a Maven project, written in Java using the javax framework. It contains several Java files (slightly less than 3000): each one of this files is a Java servlet wich can contain either a true vulnerability or a false positive. As the project is provided as a public repository on GitHub, one can run both dynamic application security testing tool or static vulnerability analysis tools against it.
Of course, we are also provided with a csv containing the expected results for each test case, meaning that for each test we have:

- The name of the test case;
- The vulnerability area;
- A boolean flag indicating if the vulnerability is a true or a false positive;
- The CWE number of the vulnerability.

Once the analysis is complete, we are provided with a score, called the Benchmark Accuracy Score, in the range 0-100. This score is a Youden index, computed as:

$$J = sensitivity + specificity - 1$$

where in turn we have:

$$sensitivity = TP/P = TP/(TP + FN)$$

and

$$specificity = TN/N = TN/(TN + FP)$$

Thus, we have that sensitivity represents the ability of recognizing a true positive, while the specificity is the ability of correctly identifying true negative. Looking at the formulas, it is clear that a tool which label each line of code as a vulnerability has a very low sensitivity, as the number of False Negatives is very high. At the very same way, a tool which does not recognize any vulnerability has sensitivity 0, as the TP factor is nullified. A similar consideration can be done for specificity.

1.2 Infer Quandary

Facebook Infer is an open source static code analyser written in OCaml. Based on abstract interpretation, it is not primarly intended to be used to discover security issues but, more generally, possible errors in the source code. This obviously means that it can shows several limitations for this kind of purpose: however, it seems interesting to test it against the Owasp benchmark, in order to see how can it be used and improved for security.
Infer is provided with a plugin, namely Quandary, devoted to the taint analysis of the source code. Quandary can be configured by providing a JSON file, where one can define:

- a list of sources for the input data;
- a list of sink procedures, which may use tainted inputs;
- a list of sanitizer procedures;
- a list of endpoints.

As the OWASP benchmark is a web application, tainted inputs come from servlet procedures, which are not included in the Quandary configuration by default. This means that, at a first run, Quandary got a score 0: in the next sections we are providing some configurations which can improve Infer Quandary's performances in a web environment.

2 Project structure

This project is composed of a set of Python scripts that runs the benchmark, analyses the results and creates the confusion matrix together with some statistics. The generation of the final results can be accomplished in 2 steps with the following scripts:

1. `run_test.py` Runs the Maven compilation of the benchmark together with the Infer quandary tool and exports the results.
2. `confusion_builder.py` For each exported result from the previous step, prints statistics and the confusion matrix.

2.1 Custom Infer configurations

As mentioned before, is possible to customise the analysis of the vulnerabilities with a configuration file where is possible to setup some custom sinks, endpoints, sources and sanitisers. In the case of this project, the configuration file is `benchmark/.inferconfig`. In order to run the test with a new configuration is only necessary to edit this configuration file and run again the tool using the script mentioned at step 1.

2.2 Compare the results

For each time the `run_test.py` is run, a `csv/actual/actual.csv` file is generated or overwritten. In order to compare different results from different configurations of the Quandary tool and assess the results, is possible to rename this file and run again the benchmark with the new configuration. A new `csv/actual/actual.csv` file will be generated and the old one won't be overwritten. Now, running again the script mentioned at step 2, is possible to obtain separate statistics according to all the `.csv` files placed inside the `csv/actual` folder.

3 Benchmark results

As said, we can set up Quandary in such a way that it can perform several tests on tainted input, and check wether they are sanitized or not, by defining a JSON configuration file. We run Quandary on the benchmark several times: as the first run didn't show any issue, thus scoring 0, the very last configuration obtained a score of 29.4.

This value is still not satisfactory in absolute terms, hence it deserves to be analyzed deeper, in order to see if such a bad result is due to inherent limitations of Quandary or if we can refine the configuration furhter.

Looking at the benchmark results, we can get the list of the vulnerabilites which appear to be reported as false negative or false positive. Let us take as a main example secure cookie: this appear to be in the top three both as a false positive and as a false negative. Indeed, as we set up the sanitizers, we can only specify the name of some classes which will be accepted as sanitizers. What Quandary checks is if the tainted input is treated somewhere by one of these classes.

However, there are two cases which can be misleading:

- The cookie pass through a sanitizer, but the algorithm used is not secure. In this case, Quandary will not recognize a vulnerability (false negative);

- The cookie does not pass through a sanitizer, but it does not contain sensible data. Quandary will report this as a vulnerability, hence giving a false positive.

This is due to a limitation in the Quandary configuration: as said, we can only specify a sanitizer class, whithout checking any of its parameters or properties. This obviously limits the possibility of checking that the sanitizing procedure is effective, then affecting the results.

4 Conclusions

With our configuration, Quandary obtained a good sensitivity (0.8) with a low specificity. This is a precise choice, as it is preferable to deal with false positive rather than false negatives. At the best of our knowledge, then, Quandary seems not to be a widely usable tool for web applications security at the moment being.
However, the obtained results are still promising, as they came without any refinement of the Quandary code, but only providing a configuration file. Providing the users with the possibility to define a more fine grained configuration may lead to better results, probably competing with those of best in class static analysis tools.