The Art of War:

Offensive Techniques in Binary Analysis

This paper presented a binary analysis framework that unified a number of techniques for the automated identification and exploitation of vulnerabilities in binaries. In the introduction section, the importance of binary analysis. First of all is that interpreted languages are mostly interpreted by binary programs, the second reason is that core OS performed critical applications that complied in binary and the third reason is the rise of IOT that their frameworks are commonly written in languages that compile to binary.

Many analysis techniques have been proposed and they are vary in term of approaches and vulnerabilities they tend to be targeted. In order to be able to compare them, there have been some efforts done which due to some reasons like implementing different frameworks and engines, the results were not actually comparable. With this regard a binary analysis framework has been introduced by this paper that integrate many of these technique in a single framework. The goal of presenting the angr, was to compare their efficiency in a single framework to get better understanding of their performance. Also the dataset that has been used in this evaluation, is the one that was prepared by DARPA cyber grand challenge due to its simple environment.

In the next session, two main techniques for binary analysis have been introduced and discussed in detail. First the static vulnerability discovery that are applied without executing the victim programs, and second the dynamic vulnerability discovery that examine a program in an actual environment.

The static techniques are also divided into two main parts, those that use graph for modeling the program properties, and those that modeling the data itself. However the static vulnerability identification techniques lack from two drawbacks, not replayable of inputs and reduction of their semantic insight due to operation on simple data domains.

On the other hand, the dynamic techniques are also split into two main categories, first the concrete execution in which generates concrete inputs such that each input takes one

path through the program execution tree, and second the symbolic execution that tries to bridge the gap between static and concrete techniques.

In the next session, an overview of the “angr” engine has been introduced. Supporting varying hardware, different operating systems and analyzing paradigm have been introduced as the goals for designing this engine. Also submodules of that such as intermediate representation, binary loading, program state representation and modification, data model and full program analysis have been presented in detail. It is also mentioned that this engine has been release in open source in order for future binary analysis.

As one of the main goals of the engine that was introduced under the name of angr, different binary analyzing techniques have been implemented in order to compare results. Analysis such as value set analysis, dynamic symbolic execution, under constrained symbolic execution, symbolic assisted fuzzing and exploit generation and hardening have been discussed. The analysis was applied with same engine and almost same codebase making the comparison more acceptable. As a result, it has been shown that dynamic symbolic execution fits for finding shallow crashes that do not require the execution of many basic blocks, however deep crashes seems to be hidden and unreachable.another important conclusion that was made is that if the path generated by fuzzing were combined into the graph, it would cover more code than the one recovered by CFGAccurate. It was also worthy to mention that the exploit found by dynamic symbolic execution engines tend to represent short path.