Independent Studies Report

Tatheer Zahra
Advisor: Dr. Gary Tan, Dr. Saeid Tizpaz

November 9, 2023

1 Problem Statement

Fuzzing has been an important technique for finding errors and vulnerabilities in software projects. Despite advanced techniques, they lack efficiency in finding new errors. Recently, some architectures have started incorporating neural networks as an intermediate step during fuzzing. Judging the accuracy and time efficiency of neural networks could be a proof of whether neural networks would change the fuzzing game altogether or not.

2 Objective

Neural Networks are mostly used for learning the input and output mapping and maximizing edge coverage. The neural networks could either use the edge or path to identify good inputs and inputs that result in unique branches. These all demonstrate that NNs can be a good addition to fuzzing frameworks; however, it is important to deeply understand how machine learning acts in favor of fuzzing. It could be done by using metrics like accuracy, time complexity, and computational complexity.
The report is divided into multiple sections, where each section would address a particular concern regarding use of machine learning, specifically neural networks.

3 Machine Learning for fuzzing

Before we discuss Neural Networks, we need to set the basis for pursuing machine learning for fuzzing. There are multiple steps that are involved in using a machine learning algorithm such as pre-processing of data, testing, training, etc. All of this could be an overkill for current fuzzing frameworks. However, we need to acknowledge that fuzzing in hindsight is perfect for directly applying a simple model such as a neural network. The reason being that a fuzzer like AFL could be used to create massive samples. These samples are divided into testing and training. AFL server also helps in generating labels for this data. Once we have labeled data, it could easily be converted into vectors as most of the fuzzer generated data is textual. Then, a simple model could be applied to generate more mutations.

4 Fuzzing applications of Machine Learning

Machine learning can easily be applied to all stages of fuzzing. It could be used for generating a seed file, generating test cases, generating mutations, and analysing exploitability. For the scope of this study, we only focused on seed file generation. In seed file generation, we try to understand the relationship between input seed file and its execution path. It helps us in understanding the grammar of the program in a better way. Here, machine learning algorithms map this relationship.

5 Progress and Directions

Out of everything we did this semester, sharing the ones with some potential for future work:

5.1 Literature Review

A big chunk of the semester was spent understanding the background information and doing a thorough literature review. I have kept a log of all papers I read during the semester, these would be shared later as well. One key aspect that we discovered this semester was that only 4 to 5 studies out of the 44 main studies demonstrate on the performance of the machine learning algorithms as well. All baseline studies focus on the performance of the fuzzer, while not stating the direct accuracy of the machine learning algorithm chosen. Therefore, having enough experiments to back up the performance of ML based techniques is a promising direction as all primary studies lack it.

5.2 LSTM

One interesting direction, which could be pursued later, is the usage of LSTM for mapping the input to path taken or edges of the program. LSTM is a sequential model, hence, it could be great for modelling or detecting execution paths that contain vulnerabilities if we feed known vulnerabilities to it. Having a deeper understanding of LSTM could help in a direct comparison of NEUZZ with LSTM. Having said that, the current studies that use LSTM are limited to understanding vulnerable code coverage. Therefore, applying LSTM to understand normal code coverage could be a difficult task to achieve.

5.3 NEUZZ

NEUZZ is one of the benchmarks of current fuzzing techniques. It belongs to the "mutation based testcase generation" class of fuzzers. Even though it belongs to this class, it still helps us in understanding the relationship between input seeds to execution paths/edges.
NEUZZ focuses on when to mutate input samples and how to mutate them. It strategically finds locations in the previous test samples to generate mutations. Here, they use a gradient guided technique, NN, for smoothing program discontinuities, ridges, etc.
The gradient-guided technique results in a higher code coverage. The goal behind using NN is to increase coverage by mutating seed input file. The mutations are then evaluated to determine which mutations would hit more crashes or vulnerabilities. Hence, more edges/paths are explored.
Proposition for NUEZZ:

- The neural network is a surrogate function, and it is used for creating initial mutations, and then retrained into generating more mutations. I proposed breaking the retraining loop. Instead, we train the neural network in one go and then compare the results against the AFL generated samples to measure accuracy.
- The current NEUZZ repository lacks splitting data into training and testing phases. I have already added the code for testing. We can use the test data for measuring accuracy after adding the retraining loop back into the program. This will let us highlight the impact of retraining.
- The code base of NEUZZ also doesn't show the impact of choosing hyper-parameters for each type of input program. They use 10 benchmark programs, but they don't highlight how the neural network is fine-tuned to give accurate results and dodge over-fitting.

6 On-going

Since this is an ongoing research, therefore, it is important to highlight some of the things that I have been working on. Here are those:

- Creating a document that explains everything in the original NEUZZ repository for anyone who works on it later.
- Already submitted detailed steps for setting up the environment without making any changes to the original code.

- I have edited the main NEUZZ neural network to be compatible with recent python version. However, if you try running the original code, it'd crash. Therefore, best bet is going with a virtual environment. It is also important to note that the code supports python 2.7, which has reached its end. Thus, it isn't viable to keep testing the original code.
- I have broken down the training loops and added functions for testing and training separately. Need to be a little more refined as it is giving me high accuracies, which I believe is over-fitting.

7 Conclusion and Future Work

In conclusion, it can easily be said that evaluating the true performance of NNs is a promising direction for fuzzing techniques. We already have a definition of which metrics we would be focusing on. However, we need sufficient experiments to backup our proposition. Even if we prove that Neural Networks are highly accurate, we would be putting forward a step-by-step process of judging the performance of a neural network in fuzzing environments. We can further claim why machine learning is more efficient for finding vulnerabilities in software. This could easily be extended to multiple applications.