BertRLFuzzer: A BERT and Reinforcement Learning Based Fuzzer (Student Abstract)

Piyush Jha[1], Joseph Scott[2], Jaya Sriram Ganeshna[2], Mudit Singh[2], and Vijay Ganesh[1]

[1]Georgia Institute of Technology, USA
[2]University of Waterloo, Canada
{piyush.jha, vganesh45}@gatech.edu, {joseph.scott, jsganesh, m286sing}@uwaterloo.ca

Abstract

We present a novel tool BERTRLFUZZER, a BERT and Reinforcement Learning (RL) based fuzzer aimed at finding security vulnerabilities for Web applications. BERTRLFUZZER works as follows: given a set of seed inputs, the fuzzer performs grammar-adhering and attack-provoking mutation operations on them to generate candidate attack vectors. The key insight in the design of BERTRLFUZZER is the use of RL, along with a BERT model as an agent, to guide the fuzzer to efficiently learn grammar-adhering and attack-provoking mutation operators. In order to establish the efficacy of BERTRLFUZZER, we compare it against a total of 13 black box and white box fuzzers over a benchmark of 9 victim websites with over 16K LOC. We observed a significant improvement relative to the nearest competing tool in terms of time to first attack (54% less), new vulnerabilities found (17 new vulnerabilities), and attack rate (4.4% more attack vectors generated).

Introduction

In recent decades, we have witnessed a dramatic increase in the number and complexity of Web applications and a concomitant rise in security vulnerabilities (OWASP 2023). A common problem faced by developers is that the kinds of fuzzers they use to find bugs in Web applications are often required to be grammar-adhering, especially when fuzzing applications with sophisticated input grammar. Developing grammar-adhering fuzzers is well-known to be a time-consuming and error-prone process. Moreover, if newer classes of security vulnerabilities are discovered, the fuzzer may need to be modified accordingly.

The problem we address in this paper is to create a fuzzer that takes as input a victim application and a set of seed inputs, and outputs a mutation operator that transforms these inputs into attack vectors for the given victim application in an extensible, (likely) grammar-adherent, and completely automatic manner.

It is well known that Large Language Models (LLMs) based on BERT (Devlin et al. 2018) have a surprising capacity to learn the grammar of programming languages with just a few fragments of code and without requiring one to specify grammar. Based on this observation, a key insight of our work is that an LLM-augmented fuzzer can automatically learn the (relevant fragment of) grammar of victim applications and attack vectors via data (i.e., a set of grammar-adherent attack vectors or seed inputs).

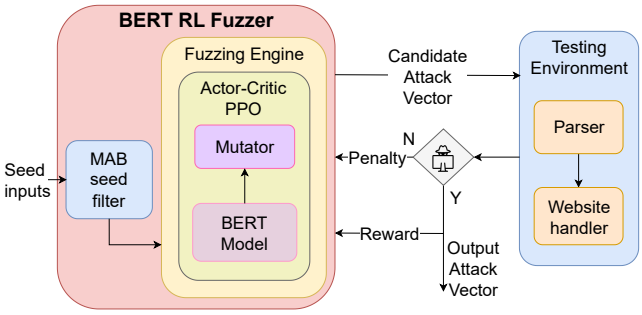

Figure 1: Architecture Diagram of BERTRLFUZZER. Please refer to Jha et al. (2023) for more details.

In BERTRLFUZZER, a pre-trained BERT model (pre-trained on seed inputs of a victim application, thus enabling it to learn the grammar of said application as well as attack patterns) acts as the agent in an RL loop which interacts with the victim application. The RL loop, in turn, enables BERTRLFUZZER to zero in on those mutations that are highly likely to expose security vulnerabilities in the victim application. The RL loop then fine-tunes this pre-trained BERT model by leveraging the victim application, the mutation operators, and the reward mechanism to search through a space of inputs in order to generate a new class of attack vectors for the given application.

BERTRLFUZZER: A BERT-based Fuzzer

Given a victim application A and a set S of seed inputs, BERTRLFUZZER outputs a grammar-adhering (when we say "grammar-adhering", we mean likely-grammar-adhering with high probability) mutation operator and a concomitant attack vector (i.e., the mutation operator mutates a seed input into an attack vector) for the given victim application A. It is possible that the seed input may be mutated multiple times by different mutation operators output by BERTRLFUZZER before it is deemed to be an attack vector. The user provides a list of grammar-adhering seed in-

Models	TFA	UF	VF	PP
BERT RL Fuzzer	**102**	**52**	**61**	5%
DeepFix	224	39	44	39%
DeepXSS	267	35	40	45%
DeepFuzz	254	28	29	41%
DQN fuzzer	110	9	10	**0%**
Baseline Grammar Mutator	NA	0	0	**0%**
Baseline Random Mutator	NA	0	0	100%
Baseline Random Fuzzer	NA	0	0	100%

Table 1: Comparison of different ML and non-ML fuzzers. (*TFA:* Time to first attack in seconds, *UF:* Unique fields, *VF:* Vulnerabilities found, *PP:* Parser Penalties)

Models	TFA	UF	VF	PP
GRU based DQN	224	39	44	39%
GRU based PPO	197	39	45	43%
Attention based DQN	155	43	48	31%
Attention based PPO	141	43	50	28%
+ improved rewards	125	48	57	18%
+ BERT pre-training	118	**52**	**61**	**5%**
+ MAB seed filtering	**102**	**52**	**61**	**5%**

Table 2: Impact of design choices: Ablation studies (*TFA:* Time to first attack in seconds, *UF:* Unique fields, *VF:* Vulnerabilities found, *PP:* Parser Penalties)

puts that are samples from a known class of attack vectors (easily found in the public domain) for a given victim application. The seed inputs must be representative of the vocabulary and grammar of some known attack vectors (e.g., BERTRLFUZZER cannot come up with a UNION-based attack if it has not seen one before). These common attack patterns would be used as the pre-training dataset for the BERT model and seed inputs for the BERTRLFUZZER. The goal here is for the tool BERTRLFUZZER to learn a meaningful representation of the input grammar and attack patterns and then automatically generate appropriate attack vectors specialized for a given victim application. We gathered over 7K attack vectors to pre-train the BERT model from the files in public GitHub repositories focused on injection attacks.

Implementation Details of BERTRLFUZZER. Initially, BERTRLFUZZER's seed generator generates samples from a list of seed inputs (these are samples from a known class of vulnerability) that are pre-processed and tokenized. The MAB agent filters and selects one seed input from the list and feeds it to the pre-trained BERT model that acts as the Actor-Critic PPO agent. This PPO agent predicts a grammar-adhering mutation operator, that is then used to create a candidate attack vector. This candidate vector is passed on to the victim website in the testing environment and sends a reward/penalty signal back to the RL agents (i.e., BERT and MAB). Success is detected when the purported attack vector is indeed able to launch an attack on the victim application. In case of a reward (or penalty), the RL agents are trained by appropriately modifying the probability distribution to prefer (or reject) the corresponding mutation operation. The previously mutated candidate vector is now used as an input for the PPO agent to predict new mutation operations. This loop is repeated several times before discarding the current input string and choosing a new one from the MAB agent. The fuzzing process terminates when the desired timeout or epoch has been reached. (Please refer to Figure 1.)

Experimental Evaluation. We evaluate our tool against 13 black box and white box fuzzers over a benchmark of 9 victim websites (with over 16K LOC), including the benchmarks used by the authors of these tools. We find that BERTRLFUZZER outperforms all the other tools in terms of time to first attack, unique fields, and number of vulnerabilities found (Table 1). More specifically, on the time-to-first-attack metric, our tool is 54% faster than the nearest competing tool, finding 17 new vulnerabilities in 13 new unique fields (Please refer (Jha et al. 2023) for additional results). The only metric where BERTRLFUZZER doesn't outperform all other tools is parser penalties, where it has a score of 5%, and the grammar-adhering mutation fuzzer that we wrote has a parser penalty score of 0%. This is to be expected since in the case of the grammar-adhering mutation fuzzer, we explicitly specified the grammar, while BERTRLFUZZER learned a representation of the grammar of the victim application with high accuracy.

We reuse the same Web applications and perform an ablation study (Table 2) to observe how each component plays an essential role in BERTRLFUZZER. We observe that introducing a PPO agent with better reward signals in the RL loop of BERTRLFUZZER significantly increases the number of unique fields and vulnerabilities found. This result shows that using an improved reward signal helps BERTRLFUZZER to explore the search space better. We also observe clear improvements over a Deep Q Network (DQN) counterpart (See Table 2).

Moreover, using a BERT model as an RL agent led to a drastic decrease in parser penalties (-13%) relative to a vanilla RL agent. This result shows that using a BERT model enables BERTRLFUZZER to be significantly more grammar-adhering than comparable techniques.

To show that our tool can be easily extended to other categories of attacks, other than SQLi, we evaluated on one of the real-world benchmark sets with XSS vulnerabilities present. The ease of pre-training a grammar-adhering model helps our tool extend to the new attack easily. Our tool is able to find all the SQLi and XSS1 (first-order XSS) vulnerabilities reported by the authors. Please refer to Jha et al. (2023) for details of this experiment.

References

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jha, P.; Scott, J.; Ganeshna, J. S.; Singh, M.; and Ganesh, V. 2023. BertRLFuzzer: A BERT and Reinforcement Learning based fuzzer. *arXiv preprint arXiv:2305.12534*.

OWASP. 2023. Open Web Application Security Project (OWASP) Foundation.