2024 3rd International Conference on Big Data, Information and Computer Network (BDICN) | 979-8-3503-6382-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/BDICN62775.2024.00025

Based on Generative Adversarial Networks seed generation method for fuzzing

Zekai Cheng*, Yongkang Guo, Xiaoke Li, Jiahao Hu
School of Computer Science and Technology, Anhui University of Technolgy, Ma'anshan, Anhui, China
*Corresponding author: chengzk@ahut.edu.cn

***Abstract*—Fuzzing, as one of the effective methods for vulnerability detection, has shown significant impact in discovering potential software vulnerabilities. However, the quality of seed inputs remains a critical factor influencing the effectiveness of fuzzing. and although in recent years, machine-learning based seed generation methods have achieved some results, there are still some problems leading to the failure of test cases generated by fuzzer variants to trigger more security vulnerabilities. This paper employs Base64 encoding and decoding technology to flexibly expand the types of generated seeds. Using the RelGAN model for seed generation and subsequently modifying the generated files with the hot-spot stitching algorithm, we enhance the quality and diversity of the seeds. This significantly improves the performance of fuzzing on target programs with various input formats. The experimental results show that the method in this paper finds a higher number of unique crashes compared to AFL++ using the original seeds.**

Keywords-Vulnerability detection, Fuzzing, Generating Adversarial Networks

I. INTRODUCTION

Fuzzing, requiring minimal knowledge about the target program and minimal manual testing while offering strong scalability, has become one of the most successful techniques for vulnerability detection[1]. Leading companies such as Google, Microsoft, and Cisco regard fuzzing as a crucial component in secure development, thereby enhancing software security. In fuzzing, carefully crafted seed files are input into the target program for automated or semi-automated testing. During the test, information is dynamically collected to guide the mutation of seeds. If a crash is triggered, the fuzzer records the current input file for subsequent analysis to identify potential defects in the target program. Therefore, the seed file, as the initial input in fuzzing, directly influences whether vulnerabilities will be triggered. The quality of seed files is a key determinant of the effectiveness and efficiency of fuzzing, where high-quality initial inputs not only increase the success rate of vulnerability detection but also accelerate the testing process. Consequently, the careful selection and optimization of seed files are vital strategies to ensure optimal outcomes in fuzzing.

Traditional seed selection strategies often consume considerable time to acquire an appropriate set of seeds, and in many cases, they do not exhibit significant advantages over random seed selection[2]. With the ongoing development of neural networks and machine learning technologies, these advanced techniques have started to be widely applied in directly generating seed files for fuzzing. Learn&Fuzz [3] attempts to intelligently guide seed generation by learning PDF syntax rules through a Seq2Seq model. Experimental results show some progress in coverage, but the model merely transfers generated PDF objects to original file objects with most content unchanged, thus limiting improvements in fuzzing effectiveness. Faster Fuzzing [4] first proposed using Generative Adversarial Networks (GAN) [5] instead of RNNs to learn data distribution features and directly generate new samples. The results indicate that GANs are indeed more effective, but experiments were conducted only on a small program, Ethkey, with a limited scale of test cases, casting doubts on the proven effectiveness. Additionally, GAN models face issues of lower quality generated data samples and difficulty in convergence.

Addressing the issues mentioned above, this paper introduces a novel approach for generating fuzzing seed files. Firstly, our method resolves issues with various data formats, demonstrating the scalability of GAN-based fuzzing in handling complex data. Secondly, by employing the RelGAN [6], we generate seed files and address the imbalance between loss values and gradient updates in GAN's task of processing discrete data. Lastly, integrating a hot-spot stitching algorithm, we further enhance the quality of the generated seeds. This improves the efficiency of AFL++ [7] in detecting crashes. Our contributions are as follows:

- We designed and implemented a fuzzing seed generation framework based on GAN and Base64 encoding/decoding rules, enabling it to handle various data types and enhancing its scalability;
- The RelGAN model, tailored for discrete data, was employed, and its output was combined with the hot-spot stitching algorithm to improve the quality of the generated seeds;
- Experiments based on AFL++ (one of the well-known grey-box fuzzers) and testing on common software indicated that our method significantly enhances the efficiency of fuzzing and achieves a higher crash rate compared to the original seeds.

The rest of this paper is organized as follows. Section two introduces related research on fuzzing. Section three details the proposed method. Section four provides experiments and result analysis, and section five concludes this work.

II. RELATED WORK

Fuzzing, a method for automatically detecting vulnerabilities in applications, is widely used across various software and computer systems. Initially introduced in 1988 to ensure the stability of UNIX programs, it has gained new prospects with the widespread application of machine learning in fields like image recognition and natural language processing. Numerous researchers have enhanced testing effectiveness by integrating fuzzing techniques with machine learning methods.

979-8-3503-6382-1/24/$31.00 ©2024 IEEE
DOI 10.1109/BDICN62775.2024.00025

In the test case filtering phase, Gong [8] and colleagues trained a deep learning model on samples generated by AFL [9] . This model predicts whether a newly generated sample will change the program state, thus not executing samples that fail to produce a new state. Augmented-AFL [10] utilized various neural network architectures to learn and predict the expected code coverage of given input modifications, determining the priority of mutations and selecting the most suitable locations for mutation.

In the seed file generation phase, SmartSeed [11] reads input files and converts them into a uniform type matrix in binary format. It then automatically learns features that trigger unique crashes or paths using WGAN and MLP from a collected dataset. The trained model can generate seed files more likely to cause crashes and unique paths. GANFuzz [12] learns protocol syntax by training a model in generative adversarial networks to estimate the underlying distribution function of industrial network protocol messages, enabling the generation of well-formed test cases based on this model, but it focuses solely on public or private industrial protocols.

In summary, this section of the document discusses the application of machine learning in fuzzing, particularly in the generation of seed files. While GAN technology has been progressively applied and improved, it still faces challenges such as difficulty in training convergence and limited scalability. The document proposes using the RelGAN model, specifically designed for discrete data, combined with a hot-spot stitching algorithm to enhance the quality and diversity of generated seeds. Moreover, the Base64 encoding mechanism is utilized for sample transformation, providing greater model extensibility.

III. METHOD

This section introduces a new method for generating seeds for fuzzing, implemented through Generative Adversarial Networks. We will present the overall workflow and the detailed methodologies for each specific step.

A. Overview

Our methodology, as depicted in Figure 1, comprises five stages: data collection, data processing, data generation using the RelGAN model, data reverse transformation, and hot-spot splicing algorithm. The resultant new seed set is then utilized by the fuzzer to conduct fuzzing.

B. Data collection

To enable the model to generate valuable seeds, a high-quality training set is essential. We fed the initial seed set into AFL++ for several hours, resulting in thousands of mutated seed files. These files not only retained the basic structure of the original files but also encompassed specific mutation information. We selected those variants that could induce unique crashes and enhance coverage. These were then streamlined in terms of file number and size to extract high-quality test cases for model training.

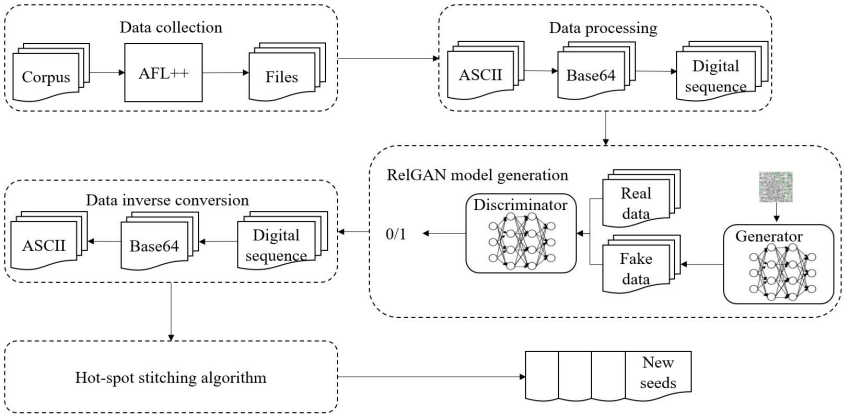

Figure 1. Approach Framework

C. Data processing

In order to meet the model's requirements on training data format and improve the scalability of the model, it is necessary to ensure that highly structured files of various types have the same metrics when input into the training set. We employed the Base64 mechanism for data transformation, dividing the encoded data into granules, ultimately converting it into numerical sequences. The specific process, depicted in Figure 2.

We first read the file in binary format and then converted the characters according to Base64 encoding rules. Post-encoding, we segment the dataset into granules, create a data dictionary, and determine the length of the numerical sequences to be generated based on the size of the largest file in each test set. Subsequently, we streamline the dictionary based on the frequency of each vocabulary, forming a moderately sized data dictionary. Ultimately, by mapping each vocabulary to its index in the dictionary, we acquire the numerical sequence dataset required for the model.

D. RelGAN model generates data

Generating text sequences with GANs is challenging as discrete data generation is non-differentiable, rendering standard gradient descent methods unusable. To overcome these challenges, GANs have incorporated reinforcement learning approaches, yielding better outcomes than maximum likelihood estimation. However, issues persist, including sensitivity to

initial parameters and hyperparameters, difficulty in training, and a tendency towards generating data with limited diversity.

To overcome previous challenges, RelGAN introduces a novel architecture composed of several modules. In the generator, relational memory replaces LSTM, enhancing expressiveness and model performance for longer texts. Its structure, illustrated in Figure 3, shows memory units where each word vector x_t incorporates a self-attention module similar to the Transformer model. Through linear transformation, it generates query, key, and value vectors. The multiplication of query and key vectors, followed by softmax, yields weights for the value, updating the memory module output for the layer. All layers' memory modules are concatenated column-wise, producing an updated memory module output for time t+1. Finally, skip connections link the updated memory module $\widetilde{M}_{t+1}$ with the memory module at that moment, generating the next moment's memory module output through a multilayer neural network with parameter $\theta_1 : M_{t+1} = f_{\theta_1}(\widetilde{M}_{t+1}, M_t)$ and the generator's output with parameter θ_2's network:$o_t = f_{\theta_2}(\widetilde{M}_{t+1}, M_t)$.

On discrete data, the gumbel-softmax relaxation model is utilized instead of reinforcement learning heuristics. The model is simplified, and a multi-layer word vector representation is utilized for the discriminator, allowing the generator to be updated in a more versatile way.

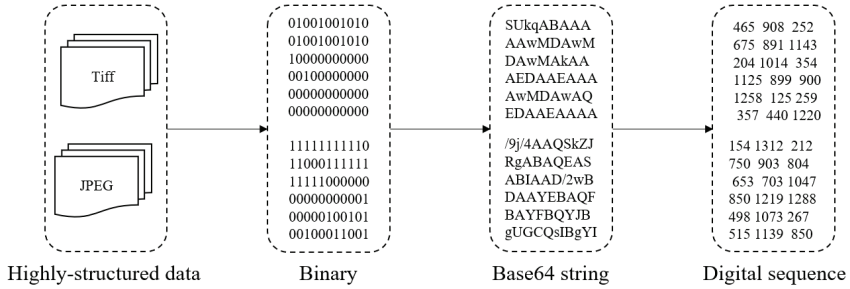

Figure 2. Workflow of conversion

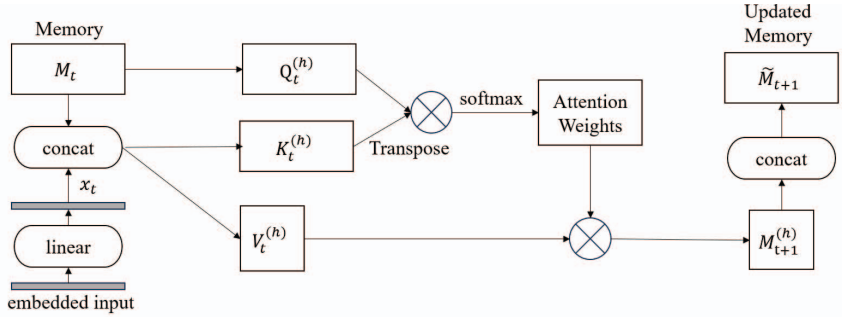

Figure 3. Structure of the Relgan generator

E. Hot-spot stitching algorithm

To better utilize the valuable information in the dataset, we proposed the hot-spot splicing algorithm, which combines the high-frequency bytes repeated in the training set with the files generated by the RelGAN model, thus further improving the quality of the seeds. Following the granularity in previous data conversions, the test set is segmented, and the frequency of byte sequences at each position is calculated. Byte sequences with higher frequencies are referred to as hot-spots. Hot-spots are selected in a certain proportion and randomly spliced with the files generated by the model, by position, to obtain new seed files. The specific algorithm flow is as follows:

Algorithm 1 Hot-spot stitching Algorithm

```
procedure STITCH_FILES(generated_files, high_freq_sequences)
    stitched_files ← []
    for each file in generated_files do
        stitched_file ← file
        for each position in file do
            k ← random(0, 1)
            if k > threshold then
                sequence ← random_choice(high_freq_sequences)
                stitched_file ← insert_sequence_at(stitched_file, position, sequence)
            end if
        end for
        stitched_files.append(stitched_file)
    end for
    return stitched_files
end procedure
```

IV. Implement and Evaluation

In this section, we present and discuss the results of our experiments and demonstrate the validity of our method.

A. Implement

The first step is to collect the training set. This paper selected two commonly used open-source programs as the target for fuzzing, as shown in Table 1. These two input formats are highly structured, making them difficult to learn through simple neural networks. Our model attempts to generate inputs that meet the format requirements of these programs. As previously mentioned, we first conducted several hours of fuzzing on the applications using AFL++ and collected valuable test cases. Subsequently, we used afl-tmin and afl-cmin to compress the file size and select the smallest subset with the same coverage as the corpus to form the training set. We then used the previously mentioned method for data conversion to meet the model's requirements. Next, the RelGAN model was used for the generation task, followed by reverse conversion of the data generated by the model. The generated files were then combined with the hot-spot splicing algorithm to complete the generation of new seeds.

The experimental environment of this paper is divided into Ubuntu 20.04LTS, the model is trained using NVIDIA RTX A5000 GPUs, and the model is constructed using python 3.8 and Pytorch 1.13 deep learning framework.

Table 1. Experiment programs.

Program	Version	Input format
LibTIFF	4.0.4	TIFF
Libexif	0.6.15	JPEG

B. Evaluation

The selection of evaluation metrics should be objective and representative. In this study, the chosen metrics are the number of unique crashes and the rate of change in unique crashes.

Number of unique crashes: Identifying more unique crashes is the primary goal of fuzzing. The higher the number of unique crashes found, the more likely it is to discover program vulnerabilities, thus contributing more to fuzzing.

Rate of change of unique crashes: mainly the change in the growth of the number of unique crashes, measuring the evolution of crashes.

The final results are presented in Table 2 and Figure 4, where Table 2 compares the number of unique crashes discovered by AFL++ in fuzzing with original seeds and new seeds generated using the method proposed in this study, within the same timeframe.

Number of unique crashes: As evident from Table 2, the method proposed in this study is capable of identifying more unique crashes within the same timeframe. There was an increase of 24 crashes in LibTIFF, amounting to an 87.51% enhancement, and in Libexif, an additional 4 crashes were found, marking an 11.76% improvement. It may be that Libexif collected a smaller dataset. Rate of Unique Crash Variation: As illustrated in Figure 4, the rate of change in unique crashes is slightly higher than before.

Table 2. Comparison of number of unique crashes.

Program	AFL++	Ours	
	Unique crashes	Unique crashes	Increase
LibTIFF	28	52	**+85.71%**
Libexif	34	38	**+11.76%**

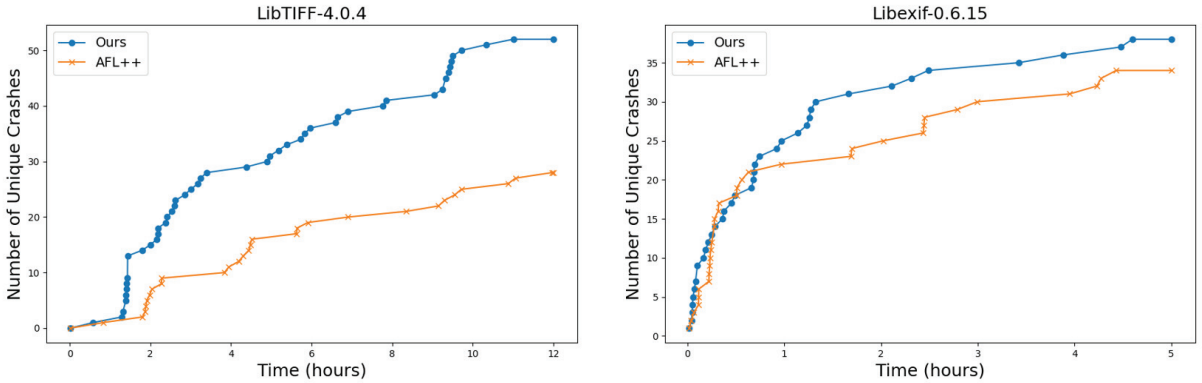

Figure 4. Trend in number of unique crashes

V. Conclusion

In this paper, we investigate the issue of generating seeds for fuzzing and propose a method that combines RelGAN with hot-spot stitching algorithm. This approach generates more valuable seed sets, accelerates the efficiency of fuzzing, and enhances its ability to discover vulnerabilities. Finally, our experimental results validate the effectiveness of this method. In the future, we aim to explore more stable and precise generative adversarial network models to further enhance the quality of the generated data.

References

[1] Zhu X, Wen S, Camtepe S, et al. (2022) Fuzzing: a survey for roadmap. ACM Computing Surveys (CSUR), 54(11s): 1-36.

[2] Zhou W, Jia Y, Peng A, et al. (2018) The effect of iot new features on security and privacy: New threats, existing solutions, and challenges yet to be solved. IEEE Internet of things Journal, 6(2): 1606-1616.

[3] Godefroid P, Peleg H, Singh R. (2017) Learn&fuzz: Machine learning for input fuzzing. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). Urbana, IL, USA . p50-59.

[4] Nichols N, Raugas M, Jasper R, et al. (2017) Faster fuzzing: Reinitialization with deep neural models. arXiv preprint arXiv. 1711.02807.

[5] Goodfellow I, Pouget-Abadie J, Mirza M, et al. (2020) Generative adversarial networks. Communications of the ACM, 63(11): 139-144.

[6] Nie W, Narodytska N, Patel A. (2018) Relgan: Relational generative adversarial networks for text generation. In: International conference on learning representations. New Orleans, Louisiana, United States.

[7] Fioraldi A, Maier D, Eißfeldt H, et al. (2020) {AFL++}: Combining incremental steps of fuzzing research. In: USENIX Workshop on Offensive Technologies (WOOT 20). Boston, MA, USA.p10-21 .

[8] Gong W, Zhang G, Zhou X. (2017) Learn to accelerate identifying new test cases in fuzzing. In: Security, Privacy, and Anonymity in Computation, Communication, and Storage. Guangzhou, China, p12-15.

[9] Zalewski M. American fuzzy lop (2023). https://lcamtuf.coredump.cx/afl/.

[10] Rajpal M, Blum W, Singh R. (2017) Not all bytes are equal: Neural byte sieve for fuzzing. arXiv preprint arXiv. 1711.04596.

[11] Lyu C, Ji S, Li Y, et al. (2018) Smartseed: Smart seed generation for efficient fuzzing. arXiv preprint arXiv. 1807.02606.

[12] Hu Z, Shi J, Huang Y H, et al. (2018) GANFuzz: a GAN-based industrial network protocol fuzzing framework. In: ACM International Conference on Computing Frontiers. Ischia Italy. p138–145.