


Article

A Novel Seed Generation Approach for Vulnerability Mining Based on Generative Adversarial Networks and Attention Mechanisms

Chunlai Du ¹, Guizhi Xu ¹, Yanhui Guo ^{2,*} , Zhongru Wang ^{1,3} and Weiqiang Yu ⁴

¹ School of Information Science and Technology, North China University of Technology, Beijing 100144, China; duchunlai@ncut.edu.cn (C.D.); 2021316210181@mail.ncut.edu.cn (G.X.); wangzhongru@cac.gov.cn (Z.W.)

² Department of Computer Science, University of Illinois Springfield, Springfield, IL 62703, USA

³ Chinese Academy of Cyberspace Studies, Beijing 100048, China

⁴ Beijing DigApis Technology Co., Ltd., Beijing 100081, China; yuweiqiang@digapis.cn

* Correspondence: yguo56@uis.edu

Abstract: Coverage-guided fuzzing has been widely applied in software error and security vulnerability detection. The fuzzing technique based on AFL (American Fuzzy Loop) is a common coverage-guided fuzzing method. The code coverage during AFL fuzzing is highly dependent on the quality of the initial seeds. If the selected seeds' quality is poor, the AFL may not be able to detect program paths in a targeted manner, resulting in wasted time and computational resources. To solve the problems that the seed selection strategy in traditional AFL fuzzing cannot quickly and effectively generate high-quality seed sets and the mutated test cases cannot reach deeper paths and trigger security vulnerabilities, this paper proposes an attention mechanism-based generative adversarial network (GAN) seed generation approach for vulnerability mining, which can learn the characteristics and distribution of high-quality test samples during the testing process and generate high-quality seeds for fuzzing. The proposed method improves the GAN by introducing fully connected neural networks to balance the competitive adversarial process between discriminators and generators and incorporating attention mechanisms, greatly improving the quality of generated seeds. Our experimental results show that the seeds generated by the proposed method have significant improvements in coverage, triggering unique crashes and other indicators and improving the efficiency of AFL fuzzing.

Keywords: security vulnerabilities; fuzzing; seed generation; generative adversarial network; attention mechanism

MSC: 90C70



Citation: Du, C.; Xu, G.; Guo, Y.; Wang, Z.; Yu, W. A Novel Seed Generation Approach for Vulnerability Mining Based on Generative Adversarial Networks and Attention Mechanisms. *Mathematics* **2024**, *12*, 745. <https://doi.org/10.3390/math12050745>

Academic Editor: Andrea Scozzari

Received: 26 January 2024

Revised: 27 February 2024

Accepted: 28 February 2024

Published: 1 March 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the rapid development of computer technology and its widespread application in various fields, software plays an irreplaceable role in the defense and military industries, finance and economy, daily life, and other aspects. However, there are also software security concerns, escalating the severity of security risks attributable to software vulnerabilities. Even ostensibly minor vulnerabilities have the potential to result in the compromise of sensitive information. Conversely, high-risk vulnerabilities possess the capability to facilitate remote device control, thereby precipitating significant information disclosure, service disruptions, and, in extreme cases, systemic destruction culminating in substantial losses. In 2016, the Japanese space satellite “Tong” disintegrated due to underlying software failures, causing billions of dollars in losses. In 2018, the National Information Technology Commission of Pakistan suffered from the leakage of personal information of millions of citizens due to network vulnerabilities in its applications. In 2018 and 2019, the Boeing 737MAX aircraft crashed twice due to software design flaws, resulting in 346 fatalities.

Software security has risen to national security, and software serving national key projects must undergo rigorous and meticulous testing and inspection to prevent irreparable losses caused by vulnerabilities. In order to effectively ensure software security, it is urgent to identify potential vulnerabilities before the software is put into use.

Within this framework, fuzzing technology has surfaced as a methodology [1]. Fuzzing involves subjecting the target program to automatic or semi-automatic testing methods, utilizing a meticulously crafted seed file as input. Throughout the testing process, dynamic information is gathered to guide the mutation of the seed. When a crash occurs during testing, the fuzzer captures the current input file, enabling subsequent assessment to ascertain potential defects in the target program. The quality of seed files is crucial as it significantly influences the effectiveness and efficiency of fuzzing.

Among numerous fuzzing tools, the American Fuzzy Loop (AFL) is a coverage-guided fuzzing framework developed by American security researchers. AFL has become one of the foremost choices in the realm of fuzzing. Its deployment not only substantively enhances the security of the tested software but also actively promotes the development of fuzzing technology.

The AFL fuzzing process is a dynamic and systematic approach to identifying software vulnerabilities by injecting mutated inputs into a target program. The process begins with an initial set of seeds, representing diverse input files, and the target program is instrumented to track code coverage during the execution. AFL employs a coverage-guided strategy, prioritizing inputs that lead to the exploration of new or less-explored code paths. The fuzzer continually mutates and tests inputs, adapting seeds based on the feedback received from the coverage analysis. Seed selection is a critical component of AFL's strategy, starting with a diverse set of seeds to cover a broad range of program behaviors. The fuzzer dynamically adjusts its seed set during the fuzzing campaign, favoring inputs that contribute to increased code coverage and have the potential to uncover novel vulnerabilities. This iterative and adaptive process, guided by coverage feedback, enables AFL to efficiently explore the vast input space and discover security issues in target applications.

This article endeavors to tackle the pertinent issue and directs attention toward enhancing research in the domain of fuzzing seed generation. Our proposition introduces an innovative approach that integrates generative adversarial networks (GAN) and attention mechanisms to elevate the quality and diversity of fuzzing seeds. To solve the problems of gradient vanishing and pattern collapse, we propose an attention-based GAN seed generation method (AtGAN) for vulnerability mining. This paper makes the following contributions:

1. A pioneering approach AtGAN is devised for vulnerability mining seed generation, amalgamating GAN with attention mechanisms. This method not only inherits the generation power of GAN but also enhances the generator's learning capability for input data through the incorporation of attention mechanisms. The synergistic integration of GAN and attention mechanisms in AtGAN presents a novel approach to test case generation, thereby making innovative contributions to the realm of software testing.
2. An effective solution has been proposed to address the prevalent issue of gradient vanishing encountered during the training of GAN generators. Through the incorporation of refined normalization operations into the attention mechanism, we have effectively alleviated the problem posed by excessively large values in the query-key (QK) pairs after dot product operations and mitigated gradient vanishing issues in softmax operations. The implementation of this normalization operation facilitates a more seamless learning process for the generator during training, thereby averting the detrimental effects of gradient vanishing on model performance.
3. To tackle the challenge of pattern collapse, an enhanced residual connection strategy has been introduced. Following each stage of the attention mechanism, we refined residual connections and incorporated scaling factors. This improvement not only

helps to better maintain information transmission but also introduces more reasonable weights in residual connections, effectively resolving the problem of pattern collapse. Generators can consistently produce diverse samples without undue bias towards a specific pattern, ensuring greater stability in the generation process.

2. Related Work

Fuzzing has emerged as a widely adopted and scalable methodology for uncovering software vulnerabilities. Within the dynamic execution phase, commonly known as the fuzzing loop, the fuzzer employs a seed selection algorithm to choose optimal seeds. These selections are based on feedback information gathered from the execution of the Program Under Test (PUT). Subsequently, the fuzzer engages in seed mutation, employing a series of strategies to generate new samples. This iterative process aims to explore different paths within the target program. The versatility of fuzzing extends its application to diverse domains, including testing for vulnerabilities in application software, libraries, kernel codes, protocols, and more. The subsequent discussion delves into various dynamic technologies and methods that are popularly employed in fuzzing.

2.1. Coverage Guide Fuzzing

Traditional seed selection strategies typically require a significant amount of time to obtain high-quality seeds, and in many cases, they have not shown significant advantages over random selection strategies [2]. To address this issue, Wang et al. [3] proposed a seed generation tool called Skyfire, which targets programs that require highly structured files as input. Through probabilistic context-sensitive grammar (PCSG) learning, Skyfire is able to automatically extract semantic information and grammar rules based on input and utilize left inference and character prediction probabilities when generating new seeds to ensure passing semantic parsing and grammar checks. However, due to its ability to only generate seeds for context-independent lightweight scripting languages and because it is limited by syntax models, it cannot cover all input file formats.

Currently, many vulnerability detection studies have also adopted deep learning technology [4–6]. The combination of deep learning and fuzzing provides new ideas for solving the bottleneck problem of traditional fuzzing technology. Deep learning technology can automatically learn grammar from a corpus set that conforms to input grammar rules and explore hidden correlations that are difficult to detect by manual methods during the learning process [7]. Learn & Fuzz [8] attempts to learn PDF syntax rules through the Seq2Seq model and iteratively predicts characters by moving the input sequence position step by step to intelligently guide seed generation. However, it only migrates the generated PDF objects to the original file objects, and the improvement in fuzzing performance is limited. Cheng et al. [9] proposed a machine learning-based framework that utilizes neural networks to discover the correlation between PDF files and the execution in the target program, in order to improve the seed input quality of fuzzing programs that use PDF files as input, but it mainly focuses on the generation of PDF files and does not provide a detailed explanation of the feasibility for other file types. Faster Fuzzing [10] uses generative adversarial networks (GAN) [11] instead of recurrent neural networks (RNN) to learn the distribution characteristics of data and directly generate new samples. However, due to the small size of the test cases used in the experiment, their effectiveness is limited. Smartseed [12] implemented seed generation using a Wasserstein GAN (WGAN) model. It faces problems due to the low quality of generated data samples and difficulty in convergence, and there is no clear explanation of the effectiveness of more complex GAN models. Ganfuzz [13] demonstrated the ability of GAN to generate highly structured inputs, learn the structure and distribution of data frames through GAN, and generate models that can resemble protocol message sequences. However, it only focuses on public or private industrial agreements.

Present seed generation endeavors encounter several challenges: (1) The efficacy of current seed generation strategies remains suboptimal, prompting the need to devise

methods for generating seed files swiftly and effectively. (2) Ensuring the intrinsic value of automatically generated seed files poses a significant concern, warranting attention to the quality and relevance of the generated seeds. (3) The question of whether the generated seeds possess universality across various fuzzing tools necessitates exploration as the compatibility and applicability of seeds across different tools remain a critical aspect of seed generation research.

2.2. Directed Greybox Fuzzing

Directed grey-box fuzzing (DGF) is a fuzzing approach based on the target location or the specific program behavior obtained from the characteristics of a vulnerable code. Unlike CGF, which blindly increases path coverage, DGF aims to reach a predetermined set of places in the code (potentially vulnerable parts) and spends most of the time budget arriving there, without wasting resources emphasizing irrelevant parts.

AFLgo [14] and Hawkeye [15] were performed within the program against user-specified target sites using distance metrics. A disadvantage of the distance-based approach is that it only focuses on the shortest distance; therefore, when there are multiple paths to the same goal, longer paths may be ignored, resulting in lower efficiency. MemFuzz [16] focused on code regions related to memory access and further guided the fuzzer by memory access information executed by the target program. UAFuzz [17] and UAFU [18] focused on UAF vulnerability-related code regions, leveraging target sequences to find use-after-free vulnerabilities, where memory operations must be performed in a specific order (e.g., allocate, free, and then store/write). Memlock [19] mainly focused on memory consumption vulnerabilities, took memory usage as the fitness goal, and searched for uncontrolled memory consumption vulnerabilities but did not consider the influence of data flow. AFL-HR [20] triggered hard-to-show buffer overflow and integer overflow vulnerabilities through coevolution.

3. GAN Based on Attention Mechanism for Vulnerability Mining

In response to the low efficiency of current seed selection strategies, the inability to guarantee seed quality, and the insufficient universality of seed generation schemes, this paper proposes to generate seeds through an attention-based GAN (AtGAN) as shown in Figure 1. This method collects excellent test samples from the fuzzing process and inputs them into the AtGAN for training and optimization. The trained AtGAN can fit the real data distribution, and its attention mechanism can focus on specific effective parts of the input data, learn the structural characteristics of high-quality test samples, and generate high-quality test seeds to improve the testing performance of AFL.

3.1. Seed Generation Framework

In software testing, an initial input seed with good characteristics is crucial for improving the efficiency of fuzzing. When we obtain a batch of test cases (seeds) with good characteristics, being able to reuse them multiple times in different target programs is crucial for the efficiency and results of fuzzing. This advantage is particularly important in the initial path exploration stage as it effectively reduces the time and resource costs required for fuzzing while also reducing the difficulty of software vulnerability mining.

This article proposes a seed generation approach utilizing an improved GAN to generate high-quality seeds. The core idea is to use GAN to learn input test samples and then obtain seeds that can integrate various excellent features and provide more comprehensive code path coverage for the tested program. In the process of selecting GANs, this article opts for fully connected GANs. These networks are characterized by their lightweight design, making them well-suited for scenarios with low problem complexity or limited data scales. Notably, in seed generation, fully connected GANs stand out by efficiently producing high-quality test cases at a minimal computational cost compared to other GANs. This approach also introduces an attention mechanism in the process of integrating GAN, which can make the GAN model more focused on the key parts

of the input test samples and more targeted when generating new seeds. The core goal of introducing an attention mechanism is to enable the GAN to learn input test samples more intelligently and to pay more attention to preserving test samples that can trigger crash points when generating seeds. This mechanism of focusing on key points helps to generate seeds with higher effectiveness, thereby enhancing the effectiveness of fuzzing, improving code coverage, and ultimately discovering more potential security vulnerabilities in the target programs.

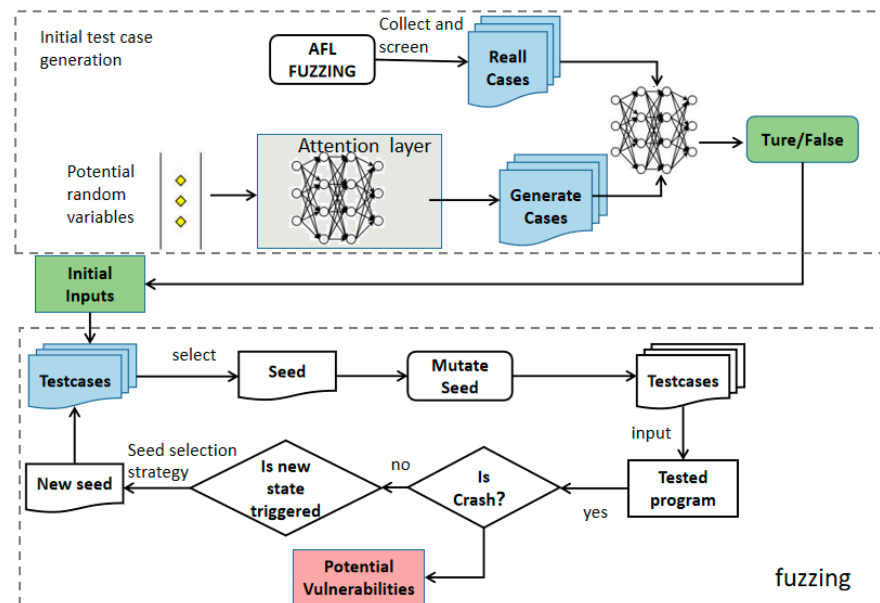


Figure 1. A GAN based on an attention mechanism for vulnerability mining.

3.2. GAN for Seed Generation

A GAN for seed generation consists of two main components: a generator and a discriminator, which can use different neural networks (NN) to adapt to specific task requirements. The specific implementation of GAN for seed generation is shown in Figure 2. The generator network consists of three subparts: attention block, local detail block, and merge block. The combination of multiple networks is used to capture features with multiple scales and multiple receiving fields. The attention module extracts features by introducing attention-driven and remote dependencies. Local detail blocks extract and process local features from input data, including convolutional layers, normalization layers, and activation functions, which ensure the generator better mimics the features of real data and improve the whole approach’s performance and stability. Finally, the merge module is applied to merge the features obtained from attention blocks and local detail blocks and generate the final seed samples.

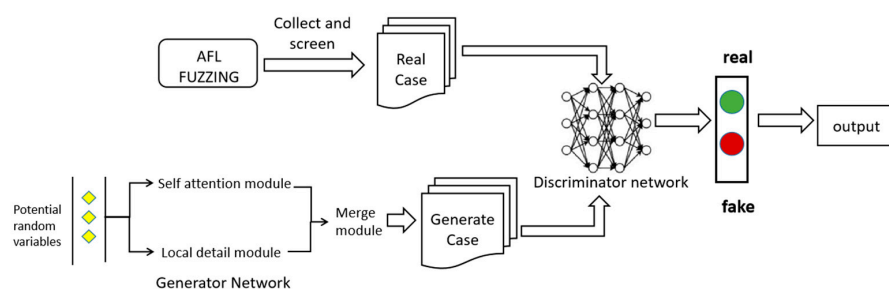


Figure 2. Specific Implementation Process of Fuzzing Seed Generation.

3.3. Construction of Initial Seed Set

To train the AtGAN, an initial seed set is needed. We performed a 48-h fuzzing on the target software using AFL tools and collected and screened high-quality mutation seeds as an initial seed set for training. Then, we defined test samples that met the following criteria as high-quality seeds: (1) test samples that can cause program crashes during runtime, (2) test samples with high code coverages, and (3) test samples that can trigger new code paths during runtime. These types of test samples exhibit significant differences in program behavior during runtime compared to programs that use the original input seed as input, making them more likely to trigger new execution paths. Therefore, the test cases we collected not only retained the original structure of the seed files but also contained mutation information.

The seed data must be processed to meet the input format requirements of the GAN. Firstly, each seed x in the filtered dataset is tensorized to obtain the tensor form X , where x_i is the i -th byte of seed x .

$$X = [x_1, x_2, x_3, \dots, x_n] \quad (1)$$

Then, each seed is padded so that each input tensor seed has the same length. To improve training performance, the filled length is an integer multiple of 32 which is defined as:

$$maxlen = maxlen + (32 - maxlen \% 32) \quad (2)$$

Finally, all input tensors are traversed, and zeros are padded at the end of each input tensor with a length less than $maxlen$. After the above processing, the training samples are transformed into equal-length tensors that can be input into the GAN for training. The size of each element in the constructed tensor is within the range of $[0, 255]$. To accelerate the convergence of the network, the value of each element is normalized into the range of $[-1, 1]$ through normalization operation as:

$$X' = (X - 128) / 128 \quad (3)$$

where X is the input tensor before normalization and X' is the input after normalization.

3.4. AtGAN

To improve the performance of GAN in generating high-quality seeds, the proposed AtGAN utilizes attention mechanisms to focus on the specific characteristics of input data, learning the structural features of high-quality test samples and thereby improving the diversity and quality of generated samples.

3.4.1. GAN

In GANs, the role of the generator is to generate as many realistic samples as possible, while the role of the discriminator is to determine whether the samples come from real data or fake data simulated by the generator. During the training process of the GAN, the generator is continuously trained to generate realistic samples, attempting to deceive the discriminator into recognizing the generated samples from real data, while the discriminator tries to improve its recognizing ability as much as possible to avoid being deceived by the generator. They play games and optimize each other until the generator can produce samples comparable to real data (Figure 3).

The training process of GAN can be seen as a zero-sum game, where there is a competitive and cooperative relationship between the generator and discriminator. The task of the generator is to estimate the distribution of real data features to generate synthetic data with similar characteristics, while the task of the discriminator is to accurately distinguish between real data and generated data as much as possible. The goal of GANs is to find a balance point where the outputs of the generator cannot be effectively distinguished by the discriminator.

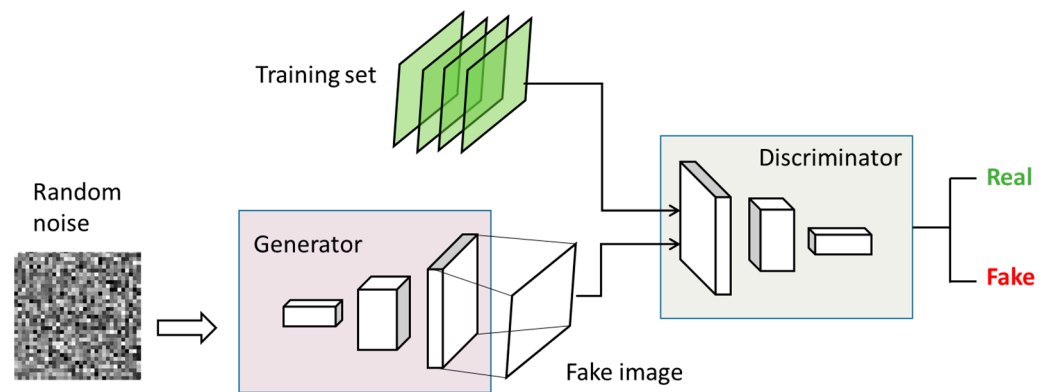


Figure 3. Training Process of Generative Adversarial Network.

The training objective for a GAN can be obtained as follows: Given real data x and discriminator $D(x)$, training the discriminator maximizes the output value of function $D(x)$. The generator function corresponding to random noise z is defined as $G(z)$. The training objective of the output $G(z)$ of the generator needs to satisfy the maximization of $D(G(z))$. The discriminator needs to correctly judge the real data and generated data, and the training objective for the generated data satisfies $D(G(z))$ output minimization.

The objective function in the training process of GAN can be expressed as:

$$\min \max V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (4)$$

where x represents real data, $p_{data}(x)$ represents the distribution that real data follow, z represents data generated by random noise, and $p_z(x)$ represents the distribution that the generated data follow. $E(x \sim p_{data}(x))$ is the expectation of the distribution of real data $x \sim p_{data}(x)$, and $E(z \sim p_z)$ is the expectation for generating the distribution of data $z \sim p_z$.

The specific training process is as follows: At the start of the training, the parameters of generator G are first fixed and the discriminator D is specifically trained, that is, the value of $D(x)$ is maximized first and the $V(D, G)$ function is maximized. After the discriminator is trained, the parameters in discriminator D are fixed, and the generator is trained to minimize the value of part $\log(1 - D(G(z)))$, promoting the generator to generate data close to reality. Through long-term mutual confrontation, the discriminator and generator in the GAN tend to converge in multiple rounds of training, ultimately resulting in similar properties and characteristics between the distribution $p_{data}(x)$ of real data and the distribution $p_z(z)$ of generated data. The discriminator cannot accurately distinguish between real data and generated data.

3.4.2. Attention Mechanism

Attention mechanism is a technique in deep learning that mimics human vision and perception mechanisms. Its core principle is to effectively convert input data into vector form, assigning corresponding weights to each input position to represent the level of attention. The essence of the attention mechanism is to filter out a small amount of important information and focus on important information. The larger the weight, the more focused it is on its corresponding value, that is, the weight represents the importance of information, and the value is its corresponding information. The key calculation steps involve calculating the correlation between the model state and each input position, usually achieved by measuring the similarity between the input vector and the model state, such as dot product, scaled dot product, etc. In calculations, these weights are dynamically generated, allowing the model to flexibly adjust its attention to different location information in different contexts and input contexts. This dynamic adjustment mechanism enables the model to focus more on critical information in the input data, improving the model's perception ability and information processing efficiency. As shown in Figure 4, the

attention mechanism aggregates attention between the query and the key. Given a query, it calculates the correlation between the query and the key. Then, based on this correlation, it identifies the most suitable value. This process achieves attention weight allocation for the value, generating the final output result.

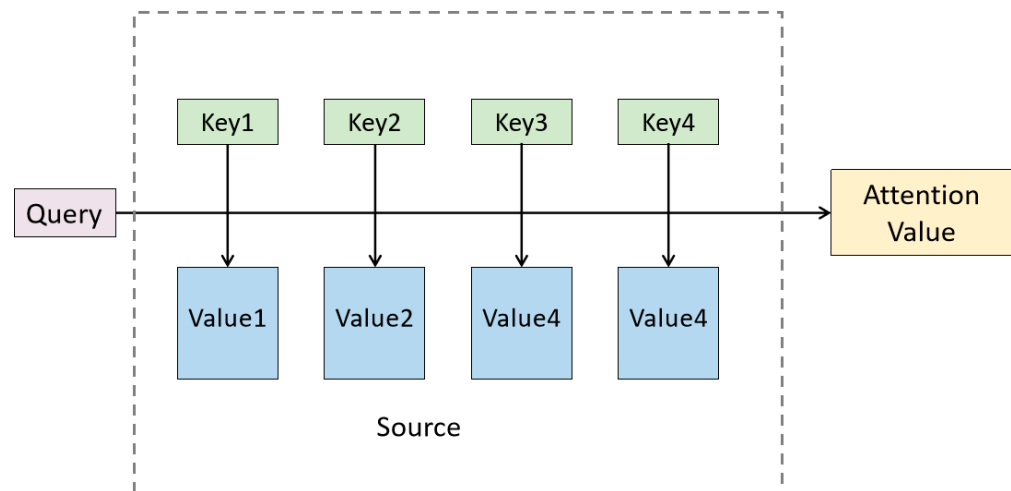


Figure 4. Principle of attention mechanism.

By introducing attention mechanisms, the deep learning model can better capture the dependency relationships between different parts, thus better adapting to the requirements of different tasks.

Overall, the attention mechanism introduces flexible information processing methods, enabling the model to respond to changes more intelligently in different scenarios and input data, improving the performance and generalization ability of deep learning models.

3.4.3. GAN with Attention Mechanisms

In the implementation of the generator, the proposed AtGAN model introduces an attention mechanism to enhance the generator’s learning ability on input data. The attention mechanism allows the generator to focus on specific parts of the input seed and improves the generator’s ability to learn important features.

In GAN, when the learning abilities of the generator and discriminator do not match, it may lead to the loss function of the discriminator converging too quickly during learning, making it difficult for the generator to effectively learn data features. In order to coordinate the learning steps of the generator and discriminator, in this paper, both the generator and discriminator use feedforward neural network models. The discriminator adopts a three-layer fully connected network, which can effectively learn the features in the input data and accurately determine its authenticity. To enhance the discriminative ability and accuracy of the discriminator, LeakyReLU is selected as the activation function, which has good nonlinear characteristics. The specific structure and parameters of the discriminator in the proposed GAN are shown in Table 1.

Table 1. Discriminator Network Model Parameters in GAN.

Layer (Type)	Output Shape
Fully connected 1 (Linear)	(batch size, 256)
ReLU 1 (LeakyReLU)	(batch size, 256)
Fully connected 2 (Linear)	(batch size, 256)
ReLU 2 (LeakyReLU)	(batch size, 256)
Fully connected 3 (Linear)	(batch size, 1)
Sigmoid (Sigmoid)	(batch size, 1)

The generator adopts a three-layer fully connected network, which can convert the input noise into generated data samples that conform to a specific distribution. In the first two fully connected layers of the network, each layer contains a linear transformation and a ReLU activation function, which has a good ability to fit the characteristics of real data and has certain nonlinear characteristics, improving the expression ability of the neural network. The final layer of a fully connected network includes a linear transformation and a Tanh activation function to map input values to the range of $[-1, 1]$. The specific structure and parameters of the final layer of the fully connected network are shown in Table 2.

Table 2. The final layer of the fully connected network.

Parameters	Values
Type	Linear fully connected
Input size	100
Output size	(8, 2, 2)
Activation	None (Tanh applied separately)
Weight initialization	Mean: 0, Standard deviation: 0.02
Bias initialization	Mean: 0.01

By combining the learning mechanisms of generators and discriminators, we aim to optimize the performance of GAN and solve problems that arise from mismatched learning abilities, thereby improving the diversity and quality of generated samples and improving the testing performance of AFL.

Specifically, the attention mechanism is introduced into the generator network to enhance its learning of key features in input data. This mechanism is implemented through a set of linear transformation layers, which are responsible for mapping input features to representations of queries (Q), keys (K), and values (V). Through these representations, the network can calculate the interrelationships between different positions in the feature space. At the same time, in the attention mechanism, some $\langle Q, K \rangle$ pairs have larger values compared to other pairs after dot product operation, resulting in the occupation of the vast majority of weights in softmax operations, while the softmax scores of other pairs approach zero, leading to the problem of gradient vanishing. To address this issue, we introduced a refined normalization operation, which involves division by a constant after the dot product operation. This process can be mathematically expressed as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (5)$$

where QK^T represents the dot product of the query and key and measures the similarity between features. $\sqrt{d_k}$ is the scaling factor, and its purpose is to control the size of the dot product to prevent the gradient from disappearing during backpropagation. The normalized attention score is then used to weight the value (V) to generate an output that integrates the information of the entire input sequence. In the architecture of the generator network, the attention module is designed to be located between consecutive transposed convolutional layers. Each transposed convolutional layer consists of a convolutional kernel and a batch normalization layer, followed closely by a nonlinear ReLU activation function. These layers gradually expand the size of features through multi-scale upsampling operations, thereby adding details at different scales at each step. The introduction of the attention module enables the network to not only capture local features but also learn long-range dependencies in input data, which is particularly important in generation tasks. This can be further quantified by the following formula:

$$x' = Attention(xW^Q, xW^K, xW^V) \quad (6)$$

where W^Q , W^K , and W^V are the learned weight matrices used to map the original features to the Q , K , and V spaces. x' is the output of the attention mechanism, which is added to the original input feature x to form residual connections, namely:

$$x'' = x + x' \tag{7}$$

Such residual connections help alleviate the problem of gradient vanishing and allow for deeper network structures. Finally, after a series of upsampling and attention mechanism processing, the features are normalized through a tanh activation function for output, making the generated seeds more consistent with the real data. Through this structure and mechanism, our generator network has shown significant advantages in fuzzing as it can generate seeds containing key features and complex patterns, thereby increasing the likelihood of discovering software defects. By selectively emphasizing key parts of input data through attention mechanisms, the generator’s ability to describe important features is improved.

In the proposed AtGAN, the attention mechanism is utilized in three stages, corresponding to the three transposed convolutional layers in the generator, as shown in Figure 5. At each stage, a matrix of queries, keys, and values is generated through linear transformation, and attention mechanisms are used to dynamically calculate and allocate the weights of input elements. This enables the generator to adapt more flexibly to the characteristics of existing data distributions when processing noise vectors, thereby generating high-quality new samples.

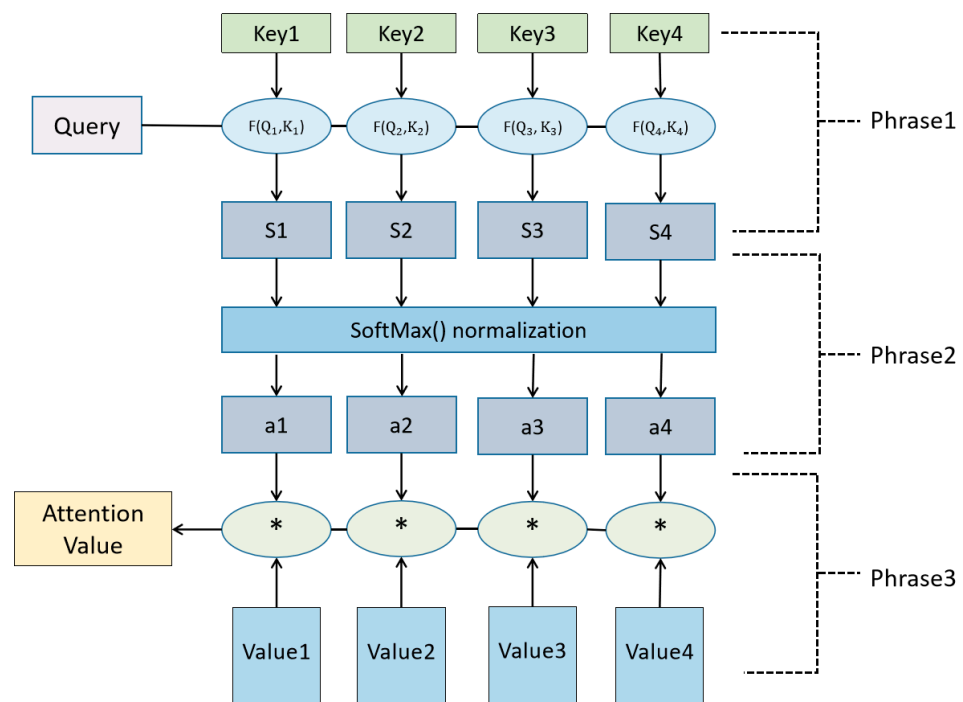


Figure 5. Specific Calculation Process of Attention Mechanism where * means multiplication.

The specific parameters and settings for the generator network model and attention mechanism are shown in Tables 3 and 4. In the generator network, a fully connected network projects the input latent vector onto an intermediate high-dimensional space, providing feature information for subsequent transposed convolutional layers. ReLU is an activation function used to introduce nonlinear features and activate intermediate feature vectors in transposed convolutional layers. Tanh is used to activate the output of the last layer of the generator by transposing the convolutional layer. In the attention mechanism, the fully connected layer is used to generate the weight matrix required for the query, key, and value. QKV is used to calculate the linear layers of the query, key, and value matrices

in the attention mechanism, mapping input data onto the three spaces calculated by the attention mechanism. The head number is the number of attention heads that allow the model to simultaneously capture different aspects of input data. The attention drop ratio is applied to the dropout layer of attention weights to prevent overfitting and improve the generalization performance of the model.

Table 3. Generator Network Model Parameters in GAN.

Layer (Type)	Output Shape
Fully connected 1 (Linear)	(Batch size, 1024)
ReLU 1 (ReLU)	(Batch size, 1024)
Fully connected 2 (Linear)	(Batch size, 1024)
ReLU 2 (ReLU)	(Batch size, 1024)
Fully connected 3 (Linear)	(Batch size, Output size)
Tanh (Tanh)	(Batch size, Output size)

Table 4. Attention Mechanism Parameters in GAN.

Layer (Type)	Output Shape
Fully connected (Linear)	(8, 2, 2)
QKV 1 (Linear)	(128, 128, 3)
QKV 2 (Linear)	(64, 64, 3)
QKV 3 (Linear)	(32, 32, 3)
Head number	8
Attention drop ratio	0.1

3.4.4. Implementation of AtGAN

Network Training

Due to the difficulty in training adversarial networks, there are issues such as training instability, gradient vanishing, and pattern collapse. Therefore, in the process of training neural networks, additional optimization methods are needed to improve training stability and the quality of seed optimization models.

The Adam optimizer combines the advantages of the momentum method and RMSProp and can adaptively adjust the learning rate on different parameter dimensions. This optimization algorithm performs well in handling sparse gradients or non-stationary objective functions, making it very suitable for training deep learning models. The Adam optimizer adjusts the learning rate of each parameter by calculating first-order and second-order moment estimates (i.e., mean and variance), which helps stabilize the training process, improve model convergence speed, and achieve better performance in seed generation and fuzzing applications.

Specifically, Adam's algorithm is used in the AtGAN, whose steps are described as follows.

First, calculate the exponential weighted average (EWA) of the gradient, expressed as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (8)$$

where g_t is the gradient of the current step, and β_1 is the decay rate, usually close to 1. Second, in order to cope with the variance change of the gradient, calculate the EWA of the gradient squared, expressed as:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (9)$$

where β_2 is another decay rate. Next, to prevent excessive deviation during the early stages of training, bias correction is carried out, which includes:

$$\hat{m}_t = m_t / (1 - \beta_1^t) \quad (10)$$

$$\hat{v}_t = vt / (1 - \beta_2^t) \quad (11)$$

Finally, use these correction values to update the model parameters:

$$\theta_{t+1} = \theta_t - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \quad (12)$$

where η is the learning rate, and ϵ is a small constant to prevent division by zero. This method makes the Adam optimizer particularly effective in various deep learning scenarios, especially in AtGAN training, balancing learning rate, and improving stability and convergence speed.

Residual Connection Scaling

Optimizing residual connections is crucial for information transmission and gradient flow in generators. After each stage of the attention mechanism, the residual connections are optimized. The proposed optimization approach introduces a scaling factor in the residual connection whose purpose is to better maintain the effective transmission of information and alleviate the impact of gradient vanishing. The proposed optimized residual connection is designed as follows:

$$Output = Input + Scaler \times Attention_Output \quad (13)$$

where *Scaler* is the scaling factor, set as a constant value between 0 and 1. The purpose of this adjustment is to introduce more reasonable weights, to better balance the information flow between the front and back layers. The scaling factor can maintain the stability of gradients, ensure the transmission of key information in the network, and improve the training effectiveness of the generator. This adjustment not only enhances the residual connection function but also plays a positive role in promoting the overall performance of the model.

In addition, in the first two layers of the fully connected network in the generator network, we adopted a more flexible activation function, LeakyReLU. This adjustment helps to improve the network's ability to learn nonlinear features while better preventing gradient vanishing problems. Compared to traditional ReLU functions, LeakyReLU allows for a small slope when the input is negative, which can better handle situations with zero gradients. The formula is as follows:

$$f(x) = \begin{cases} x & \text{if } (x > 0) \\ \alpha \times x & \text{if } (x \leq 0) \end{cases} \quad (14)$$

The adjusted activation function formula is:

$$Output = LeakyReLU(Linear(Input) + Residua(connection)) \quad (15)$$

where *Input* represents the input of this layer, *Linear()* performs a linear transformation on the input, and *Residua()* represents the residual connected part, usually the output of the previous layer or the output of another layer.

By introducing the LeakyReLU activation function, we expect the generator to be more stable and effective in handling attention mechanisms, residual connections, and network structures, thereby improving the modeling effect on complex data distributions. The introduction of this activation function helps to enhance the non-linear expression ability of the network and alleviates the problem of gradient vanishing during the gradient transfer process. Therefore, this improvement will enable the generator to better learn the features of the data and enhance its ability to fit the true distribution of the data.

4. Experimental Results

To evaluate the proposed method, we used three evaluation objectives to validate the effectiveness of the proposed model in optimizing the testing performance of AFL tools.

The specific method is to set up comparative experiments to compare whether the output of the seed by the AtGAN model proposed in this article has better quality than the original input seeds.

4.1. Experimental Environment

This experiment is based on Linux systems, namely Ubuntu 18.04. The specific configuration is shown in Table 5. The experimental object is selected as the commonly used toolset Binutils on Linux system, which includes commonly used software such as readelf, nm, and objdump. The version selected is the older version 2.25, which has many vulnerabilities and is conducive to verifying whether the seed optimization model used in this experiment can enhance the AFL vulnerability detection ability.

Table 5. Ubuntu Experimental Environment Configuration.

Configurations	
Operation system	Ubuntu 18.04.1
CPU	Intel (R) Xeon (R) CPU E7-4820 v2 @ 2.00 GHz
Number of CPU cores	16
Memory	16 GB
Hard disk	100 GB
Clang/LLVM	ver. 11.0.0
AFL	ver. 2.57b

In order to improve the testing performance during the testing process, the AFL testing mode in this article was based on source code instrumentation. During the source code compilation phase, the AFL-GCC compiler was used for instrumentation compilation, replacing the QEMU mode for binary program instrumentation. This avoids wasting system resources during the testing process and improves testing efficiency.

To fully demonstrate the effectiveness of the proposed improvement method in optimizing the performance of AFL fuzzing, the following three experiments were set up:

- (1) AFL group: This group is a control group that does not use any optimization methods and only uses ordinary seeds to input into the AFL tool for fuzzing experiments.
- (2) GAN-AFL group: This group uses a seed optimization model based on the original GAN to optimize the seed input of AFL.
- (3) MOPT-AFL group: This group utilizes a customized particle swarm optimization (PSO) algorithm to find the optimal selection probability distribution of operators with respect to fuzzing effectiveness.
- (4) ECOFUZZ group: This group is based on a unique adaptive scheduling algorithm as well as a probability-based search strategy.
- (5) GSA-FUZZ group: This group is based on learning the optimal selection probability distributions of operators and mutation positions and designs a position-sensitive strategy to guide seed mutation with learned distributions.
- (6) AtGAN-AFL group: This group uses the proposed AtGAN to optimize the seed input of AFL.

This article evaluates the performance of the generated seeds using three metrics: code coverage, new edges, and crashes.

Code coverage: The meaning of this indicator is the code statement executed in the test program during fuzzing. The higher the code coverage, the more fully tested the program, and the more powerful the vulnerability.

New edges: The meaning of this indicator is that it is mutated by fuzzing tools in the process of fuzzing. The total number of new paths of the program is triggered by the generated test samples. This metric measures the scalability of the input seeds.

Crashes: This metric means that, during fuzzing, the program will be executed with invalid, unexpected, or random inputs to crash the system under test. The higher the number, the more likely it is that there will be a high number of vulnerabilities.

4.2. Results

In the experiments, ordinary seeds and model output seeds were input into AFL tools for 24-h fuzzing on four software types: readelf, nm, objdump, and tcpdump. The specific results are shown in Table 6:

Table 6. Performance of experiments using AFL, GAN-AFL, MOPT-AFL, ECOFUZZ, GSA-FUZZ, and AtGAN-AFL.

Fuzzing Model	Comparison	readelf	nm	objdump	tcpdump
AFL	Number of crashes	0	96	3	0
	Program path gain	3049	1203	868	1285
	Code coverage	35.50%	25.90%	31.60%	16.40%
GAN-AFL	Number of crashes	0	208	20	0
	Program path gain	3120	1926	3147	1432
	Code coverage	36.40%	37.70%	37.70%	18.10%
MOPT-AFL	Number of crashes	0	0	0	0
	Program path gain	3629	11,813	3308	10,413
	Code coverage	39.5%	41.82%	41.80%	26.60%
ECOFUZZ	Number of crashes	0	0	0	0
	Program path gain	5580	11,740	6260	11,266
	Code coverage	42.28%	41.06%	41.90%	23.43%
GSA-FUZZ	Number of crashes	0	0	0	0
	Program path gain	4789	11,488	13,219	11,822
	Code coverage	40.84%	41.54%	45.80%	27.05%
AtGAN-AFL	Number of crashes	3	105	60	0
	Program path gain	3680	1302	3192	3532
	Code coverage	38.70%	30.70%	39.50%	28.10%

5. Discussion

In terms of vulnerability discovery, no vulnerabilities were found in the six experiments conducted on the tcpdump software. This result is attributed to the powerful security measures embedded in the software, which prevent vulnerabilities from being detected within a limited experimental time. During the testing of nm, more vulnerabilities were detected compared to AFL, MOPT-AFL, ECOFUZZ, and GSA-FUZZ, but less than GAN-AFL. This difference can be traced back to the introduction of attention mechanisms. These mechanisms seem to have not effectively adapted to the generalization of various programs during the training process, leading to limitations in the ability of nm test seeds to fully cover or trigger target program vulnerabilities.

Regarding the assessments of readelf and objdump, the AtGAN-AFL group discovered more vulnerabilities than the other groups, demonstrating a significant advantage. This result emphasizes the effectiveness of the seed optimization method proposed in this paper, which integrates attention mechanisms. This combination effectively solves the problem of gradient vanishing during the training process of generative adversarial networks. Therefore, it significantly optimized the quality of output seeds and improved the ability of AFL to explore vulnerabilities.

In terms of code coverage, AtGAN-AFL has better overall performance than AFL and GAN-AFL, with the highest code coverage on readelf, objdump, and tcpdump software and is second only to GAN-AFL on nm software. Although the code coverage of AtGAN-AFL is lower than that of MOPT-AFL, ECOFUZZ, and GSA-FUZZ experiments, more vulnerabilities have been discovered than these three experiments. This validates the superiority of the seed optimization method that combines attention mechanisms in generating adversarial networks.

In terms of the number of program path gains, AtGAN-AFL performs better overall than AFL and GAN-AFL. However, the overall performance is not as good as MOPT-AFL,

ECOFUZZ, and GSA-FUZZ, indicating that these three groups of experiments tested a large number of invalid paths during the testing process, causing resource waste. This indicates that the attention mechanism-based generative adversarial network vulnerability mining model can effectively enhance the scalability of seeds, and mutated test samples are more likely to trigger new effective paths in the program.

6. Conclusions

This article focuses on the low efficiency and low quality of seed selection strategies in current fuzzing. By introducing GANs to optimize seed generation, the goal of efficiently obtaining high-quality seeds is achieved. The main contribution is reflected in enhancing the GAN by incorporating attention mechanisms, improving the quality and diversity of seed generation, and enabling the generation of high-quality seeds in a short period. The experimental results show that this method can obtain high-quality seeds in a short period, significantly improving the efficiency and accuracy of fuzzing and providing a more convenient and reliable means for vulnerability detection and repair in the field of software security.

While the vulnerability mining method has exhibited success in utilizing fully connected GANs and attention mechanisms, there are still notable limitations and opportunities for refinement. One of these limitations manifests in the method's performance weakness when testing specific programs, such as nm programs. This may be attributed to the insufficient adaptation of attention mechanisms to the generalization performance of diverse programs during training. Additionally, opting for a relatively simplistic fully connected GAN structure may pose constraints when dealing with intricate, high-dimensional data distributions.

In the future, our focus will be on refining attention mechanisms to enhance generalization performance. This entails introducing more program-specific information or exploring more complex attention mechanism models. Simultaneously, we plan to experiment with alternative GAN structures to identify a more suitable model configuration for vulnerability mining. These enhancements are anticipated to bolster the overall effectiveness and applicability of the model in the realm of vulnerability mining.

Author Contributions: Conceptualization, C.D. and Y.G.; methodology, G.X., Y.G. and G.X.; software, G.X.; validation, C.D. and Y.G. investigation, G.X. and Z.W.; resources, Z.W. and W.Y.; data curation, G.X.; writing—original draft preparation, G.X.; writing—review and editing, C.D. and Y.G.; visualization, G.X.; supervision, Z.W.; project administration, C.D. and Y.G.; funding acquisition, C.D., Z.W. and W.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China grant number 62172006 and the National Key Research and Development Plan of China grant number 2019YFA0706404.

Data Availability Statement: The test results data presented in this study are available upon request. The data set can be found on public websites.

Conflicts of Interest: Weiqiang Yu was employed by Beijing DigApis Technology Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. Beijing DigApis Technology Co., Ltd. had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Manes, V.J.M.; Han, H.S.; Han, C.; Sang, K.C.; Woo, M. Fuzzing: Art, Science, and Engineering. *arXiv* **2018**, arXiv:1812.00140.
2. Rebert, A.; Cha, S.K.; AVGERINOS, T.; Brumley, D. Optimizing seed selection for fuzzing. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, USA, 20–22 August 2014; pp. 861–875.
3. Wang, J.; Chen, B.; Lei, W.; Yang, L. Skyfire: Data-driven seed generation for fuzzing. In Proceedings of the 32nd IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 579–594.
4. Wartschinski, L.; Noller, Y.; Vogel, T.; Kehrer, T.; Grunske, L. Vudenc: Vulnerability detection with deep learning on a natural codebase for python. *Inf. Softw. Technol.* **2022**, *144*, 106809. [[CrossRef](#)]

5. Zhang, L.; Wang, J.; Wang, W. A novel smart contract vulnerability detection method based on information graph and ensemble learning. *Sensors* **2022**, *22*, 3581. [[CrossRef](#)]
6. Cao, S.; Sun, X.; Bo, L. MVD: Memory-related vulnerability detection based on flow-sensitive graph neural networks. In Proceedings of the 44th IEEE/ACM International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 1456–1468.
7. Du, L.X.H.; Chen, J.; Yang, X.X. Targeted password guessing scheme combined with GAN. *J. Xidian Univ.* **2022**, *49*, 129–136.
8. Godefroid, P.; Peleg, H.; Singh, R. Learn&fuzz: Machine learning for input fuzzing. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, USA, 30 October–3 November 2017; pp. 50–59.
9. Cheng, L.; Zhang, Y.; Zhang, Y.; Wu, C.; Li, H. Optimizing seed inputs in fuzzing with machine learning. In Proceedings of the 41st IEEE/ACM International Conference on Software Engineering: Companion Proceedings, Montreal, QC, Canada, 25–31 May 2019; pp. 244–245.
10. Nichols, N.; Raugas, M.; Jasper, R.; Hilliard, N. Faster Fuzzing: Reinitialization with Deep Neural Models. Available online: <https://arxiv.org/pdf/1711.02807.pdf> (accessed on 8 January 2023).
11. Goodfellow, I.; Pouget-abadie, J.; Mirza, M. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [[CrossRef](#)]
12. Liu, C.; Ji, S.; Li, Y.; Zhou, J.; Chen, J.; Zhou, P. Smartseed: Smart Seed Generation for Efficient Fuzzing. Available online: <https://arxiv.org/pdf/1807.02606.pdf> (accessed on 8 January 2023).
13. Hu, Z.; Shi, J.; Huang, Y.H.; Xiong, J.; Bu, X. GANFuzz: A GAN-based industrial network protocol fuzzing framework. In Proceedings of the 15th ACM International Conference on Computing Frontiers, Ischia, Italy, 8–10 May 2018; pp. 138–145.
14. Böhme, M.; Pham, V.-T.; Nguyen, M.-D.; Roychoudhury, A. Directed greybox fuzzing. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 2329–2344.
15. Chen, H.; Xue, Y.; Li, Y.; Chen, B.; Xie, X.; Wu, X.; Liu, Y. Hawkeye: Towards a desired directed grey-box fuzzer. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 2095–2108.
16. Coppik, N.; Schwahn, O.; Suri, N. Memfuzz: Using memory accesses to guide fuzzing. In Proceedings of the 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), Xi’an, China, 22–27 April 2019; pp. 48–58.
17. Nguyen, M.-D.; Bardin, S.; Bonichon, R.; Groz, R.; Lemerre, M. Binary-level Directed Fuzzing for Use-After-Free Vulnerabilities. In Proceedings of the RAID, San Sebastian, Spain, 14–16 October 2020; pp. 47–62.
18. Wang, H.; Xie, X.; Li, Y.; Wen, C.; Li, Y.; Liu, Y.; Qin, S.; Chen, H.; Sui, Y. Typestate-guided fuzzer for discovering use-after-free vulnerabilities. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 999–1010.
19. Wen, C.; Wang, H.; Li, Y.; Qin, S.; Liu, Y.; Xu, Z.; Chen, H.; Xie, X.; Pu, G.; Liu, T. Memlock: Memory usage guided fuzzing. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 5–11 October 2020; pp. 765–777.
20. Medicherla, R.K.; Komondoor, R.; Roychoudhury, A. Fitness guided vulnerability detection with greybox fuzzing. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 513–520.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.