

System call frequency analysis-based generative adversarial network model for zero-day detection on mobile devices

Akram Chhaybi, Saiida Lazaar

Mathematics, Computer Science and Application (ERMIA Team), Department of Mathematics and Computer Science, National School of Applied Sciences, AbdelMalek Essaadi University, Tangier, Morocco

Article Info

Article history:

Received May 15, 2023

Revised Sep 3, 2023

Accepted Dec 18, 2023

Keywords:

Android

Generative adversarial networks

Loss function

Malware

Mobile applications

Security

Zero-day

ABSTRACT

In today's digital age, mobile applications have become essential in connecting people from diverse domains. They play a crucial role in enabling communication, facilitating business transactions, and providing access to a range of services. Mobile communication is widespread due to its portability and ease of use, with an increasing number of mobile devices projected to reach 18.22 billion by the end of 2025. However, this convenience comes at a cost, as cybercriminals are constantly looking for ways to exploit security vulnerabilities in mobile applications. Among the several varieties of malicious applications, zero-day malware is particularly dangerous since it cannot be removed by antivirus software. To detect zero-day Android malware, this paper introduces a novel approach based on generative adversarial networks (GANs), which generates new frequencies of feature vectors from system calls. In the proposed approach, the generator is fed with a mixture of real samples and noise, and then trained to create new samples, while the discriminator model aims to classify these samples as either real or fake. We assess the performance of our model through different measures, including loss functions, the Frechet Inception distance, and the inception score evaluation metrics.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Akram Chhaybi

Mathematics, Computer Science and Application (ERMIA Team), Department of Mathematics and Computer Science, National School of Applied Sciences, AbdelMalek Essaadi University

Old Airport Road, Km 10 Ziaten B.P. 1818 Tangier, Morocco

Email: Akram.chhaybi@etu.uae.ac.ma

1. INTRODUCTION

Mobile malware has become a significant security threat, particularly for the open-source operating system Android. Malware can target mobile systems at multiple levels, posing a considerable user risk. Mobile malware typically uses two methods to infect victims, which can compromise the security and integrity of their devices and data. The first method involves tricking the victim and obtaining permissions from the Android manifest file to access sensitive data. The second approach consists in exploiting phone vulnerabilities to access user information by granting administrator privileges. Both methods involve passing system calls between the application and the system [1].

Machine learning solutions can effectively defend against various potential threats, including classifying malicious applications. Classifiers use multiple techniques to detect such applications, including signature-based detection, where a dataset of malware and benign signatures is used to train the classifier [2], [3]. This enables the classifier to identify patterns and characteristics indicative of malicious behavior, accurately classifying new and unknown applications.

In malware detection, two types of analysis are used: static and dynamic. Static analysis entails inspecting an executable file without running it. It is carried out by examining the file's code structure, metadata, and other aspects. Static analysis can discover malware hiding in code or masquerading as legitimate software. It cannot, however, detect malware that is encrypted or disguised. The process of evaluating the behavior of an executable file while it is executing is known as dynamic analysis. It is carried out by running the file in a controlled environment and watching its behavior. Dynamic analysis can discover malware hiding in code or masquerading as genuine software. It is also capable of detecting malware that has been encrypted or disguised. Dynamic analysis, on the other hand, can be time-consuming and resource-intensive [4].

Attackers' use of machine learning is leading to the development of increasingly stealthy and elusive ways of system penetration. The increasing number of mobile malware variants is undermining the efficiency of machine-learning approaches, creating a substantial danger. As the number of mobile malware types grows, machine-learning technologies face a big challenge [5]. Machine learning models are trained on historical data, and it is possible that those models did not encounter the precise patterns or features of zero-day attacks during their training period. As a result, these assaults can circumvent detection methods based on learnt patterns, allowing hostile operations to go unnoticed, where hackers modify certain features of the original malware, such as byte code, application programming interfaces (APIs), system calls, and different parts strategies, to evade detection by classifiers. System calls are functions and APIs that request services from the operating system's kernel, and malware writers aim to fool machine-learning classifiers by disguising the system calls used between the program and the kernel. To address this problem, we propose a model that utilizes generative adversarial networks (GANs) to generate new periodicities of system calls. By doing so, we aim to improve the robustness of malware classifiers against zero-day malware.

GANs consist of two main neural networks operating in a mini-max optimization framework; the first is called the generator, and the second is the discriminator. The generator aims to produce synthetic samples that resemble actual data, while the discriminator's task is to distinguish between real and fake data. Based on the feedback provided by the discriminator, the generator can iteratively refine its output until it produces data indistinguishable from the genuine samples as shown in Figure 1.

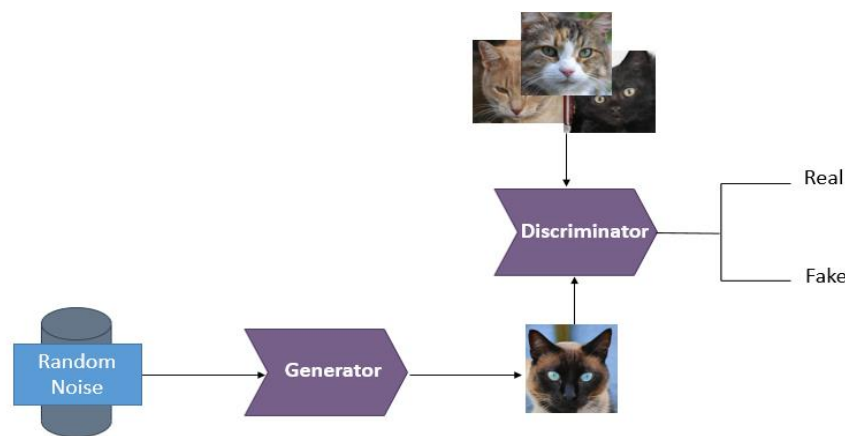


Figure 1. GAN's flowchart

The structure of this paper is as follows. Section 2 provides some related works. Our proposed approach and model are described in section 3. Section 4 discusses the results obtained from the simulation. Finally, we conclude the paper and provide directions for future work.

2. RELATED STUDIES

The popularity of GANs technology is rapidly increasing, as evidenced by the numerous applications in various fields. For instance, Yin and Yang [6] implemented GANs in mobility data to solve privacy issues. They designed a model that can train the generator and discriminator to create privacy-protected data, and they tested their model by comparing it with other works and applying attacks that can detect the user's location based on the global positioning system (GPS) information.

In another application, Moti *et al.* [7] proposed a new approach to using GANs in detecting and generating malware. They used GANs to create new malware signatures by employing a boundary-seeking GAN and a convolutional neural network (CNN) to extract features from the dataset collected by internet of things (IoT) devices, packets, and VX-Heaven. The cycle of their framework starts by reading the header of the EXE file to contain the signature. CNN extracts the features and sends them to the GAN model to generate new malware samples in a mini-max game between the generator and the discriminator. Finally, they train a deep learning model to detect the new samples using the long short-term memory (LSTM) method.

In study [8], a distributed intrusion detection system (IDS) based on GANs was proposed for IoT systems, aiming to detect intrusions on every device while preserving privacy. Each IoT device (IoTd) uses its own dataset without sharing it with others, and the authors define a discriminator in every IoTd and a central generator. The generator is responsible for creating new data, which is sent to each IoTd for comparison with real data collected from 30 subjects using smartphones with different features. After the training phase, the discriminators can act as IDS on the devices, and the central generator no longer needed to be used, resulting in a lighter IoTd. The distributed IDS achieved an accuracy of 83% for external attacks and 81% for internal attacks, improving the quality of the training dataset with 99.03% accuracy.

Taheri *et al.* [9] developed a robust architecture called Fed-IIoT (FL) for malware detection on Android devices using a client/server model. The client-side works as an attack mechanism based on a generative adversarial network (GAN) system, which generates malicious data to be injected into the dataset. On the server side, a malware detection model is implemented using a GAN and a federated learning-based architecture.

In another study, Kim *et al.* [10] proposed a zero-day malware attack detection method using GANs. This method employs GANs to generate and learn how to differentiate between fake and real malware. Learning involves extracting features from actual and counterfeit data using a deep auto-encoder (DAE). The trained discriminator then passes the detection mechanism to the detector. The results of the study showed an accuracy of 95.74%.

In their study, Hao *et al.* [11] proposed an asymmetric encryption function based on GANs to enhance the security of the IoT. They developed a method to adjust the parameters of the artificial network and improve reinforcement learning. The method provides a unique system for information exchange based on blockchain technology and the zero-trust concept. Throughout the sharing process, their protocol successfully filters out fake information while protecting participants' privacy. Furthermore, inside the universally composable safe framework, they established formal verification of the protocol's security. They conducted a number of experiments and analyzed its performance to determine its practicability. The findings show that the average execution timings for their protocol's three critical phases are 0.059 s, 0.060 s, and 0.032 s, confirming its feasibility for real-world deployment.

Shin *et al.* [12] proposed a defense procedure against Android pattern attacks using GANs and a replay buffer from a deep reinforcement learning LSTM network. The network model records the trajectory and touch pressure of the mobile device. The LSTM receives the data combined with some noise to describe the features, and the generator generates fake data to pass to the LSTM to capture the features. The discriminator evaluates the similarity between actual and generated data. The results of the study showed an accuracy of 95%.

Li *et al.* [13] proposed a new method called E-MalGAN, which utilizes GANs to generate malware attacks. This method was inspired by MalGAN [14], which creates black-box adversarial examples of attacks against Android malware detection. The system model includes a generator that learns from two discriminators: one functioning as an adversarial example detector and the other as a malware detector. The loss function of the generator decreases from 0.6 to zero after 120 rounds. Furthermore, the study results showed that over 95% of the adversarial examples generated were classified as regular programs.

Deb *et al.* [15] proposed a mobile touch stroke authentication model based on a GAN. The system architecture consists of two sides: the mobile side and the server side. The mobile side collects the raw touch stroke data from the user and sends it to the server side, where the features are extracted, and the GAN is trained. The GAN model is deployed on the mobile platform in a lightweight process. The dataset used includes 21,158 touch strokes collected from four different Android phones, and the model's performance ranged between 92% and 98%. The GAN model can be used for privacy, user authentication, and confidentiality.

A strategy was proposed for the development of malware detection models that can withstand adversarial attacks [16]. This strategy involved creating twelve distinct malware detection models using different categorization methods. Subsequently, an adversarial assault was simulated by assuming the role of an adversary and generating adversarial attacks on the aforementioned detection models using a gradient-based adversarial attack network. The objective was to alter each malware sample as minimally as possible while converting the highest number of samples into adversarial ones.

Deep feature selection (DEEPESEL) is a novel method proposed in [17] that utilizes deep learning to detect malware and malicious code in Android apps. DEEPESEL employs a set of characteristics to analyze the behavior of Android apps and classify them as either genuine or malicious. The crucial component of this method is the utilization of particle swarm optimization for feature selection. To assess the effectiveness of DEEPESEL, the authors utilized a public malware dataset comprising samples from 39 distinct malware families. The study's findings demonstrated that the suggested technique achieved high accuracy, with an approximate accuracy of 83.6% and an F-measure of around 82.5%.

Shahpasand *et al.* [18] presented a machine-learning model that utilizes the power of generative adversarial networks (GANs) to launch attacks on malware classifiers. By harnessing the expressive capabilities of GANs, we generate potent adversarial samples, ensuring that the distortion amount remains below a predefined threshold. Our results demonstrate that these generated samples successfully evade detection in 99% of the attempts, using a real dataset of Android applications.

GANs can manifest in various model types, as detailed in [19]. In their comparative study, the authors explored well-known GAN models, including Wasserstein GAN, conditional GAN, and deep convolutional GAN. They provided insights into these models' architectures, objective functions, and pivotal scenarios where they can be employed to bolster the security of mobile applications.

In general, there are two major approaches to using GANs in malware studies. The first approach aims to increase the efficiency of malware detectors by enlarging the size of the training dataset. The second approach is to help classifiers detect malware created by GANs, which is potentially indistinguishable from benign samples.

3. METHOD

Our approach involves applying GANs to generate new system call frequencies primarily used by zero-day malware. The model comprises two neural networks, as illustrated in Figure 2. The generator takes a noise signal, z , and a set of frequency examples, S , as input, while the discriminator takes both actual data and the generator's output as input and performs a comparison. The objective functions for the generator, G , and the discriminator, D , are described below.

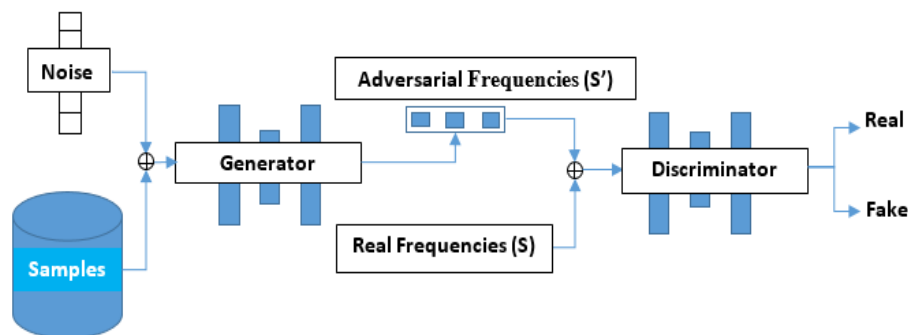


Figure 2. The proposed GAN model

This study uses a feed-forward neural network with weights assigned to both the generator and discriminator components. Specifically, the generator receives a concatenation of samples and noise as input, passing through distinct layers within the network. On the other hand, the discriminator takes as input the adversarial frequencies generated by the generator and the actual frequencies from the dataset, which are then also processed by different layers in the neural network. The primary function of the discriminator is to differentiate between the natural frequencies (S) and the generated frequencies (S').

3.1. Data collection

This section describes how the dataset for the suggested technique was acquired. The dataset CICMal2017 was used in our technique. The Android samples are divided into four categories: adware, scareware, SMS malware, and ransomware. To avoid runtime behavior modification of complex malware samples that might identify the emulator environment, the authors run both malware and benign programs on actual cellphones. The suggested approach makes use of a comma separated values (CSV) file containing 140

extracted system call frequencies from 11,599 Android application package (APK) files belonging to four malware types.

3.2. Data preprocessing

This section involves transforming the data into something that the model can use. This section focuses on increasing the model's efficiency. Finding missing data, eliminating NULL values or extraneous data, and importing the appropriate libraries are the general phases of data preparation. Although there were no missing or NULL values in the dataset used in this study, the frequencies were given as decimal figures. We converted them to integer numbers to simplify the training process, and we assigned each malware category to an array, with the four arrays included in a NumPy file.

3.3. Training process

In this section, we establish our generative adversarial network (GAN) model by configuring both the generator and the discriminator with parameters outlined in the simulation and results section. The entire training process, specific to GANs, is succinctly captured in algorithm 1, providing a clear and systematic overview. This configuration involves careful consideration of key variables, shaping the behavior and performance of our GAN model.

Algorithm 1. Training process

1. Initialize generator (G) and discriminator (D) networks with random weights.
2. Repeat until convergence (or a predefined maximum number of iterations):
 - a. Generate new system calls (S) by feeding random noise and real samples through G.
 - b. Combine real system call samples and generated samples to create a training dataset for D.
 - c. Calculate the discriminator's loss by comparing its predictions for real and generated samples.
 - d. Update the discriminator's weights using backpropagation and gradient descent.
 - e. Generate new system calls (S) by passing random noise through G.
 - f. Calculate the generator's loss based on discriminator's predictions for the generated samples.
 - g. Update the generator's weights using backpropagation and gradient descent.
 - h. Evaluate the quality of the generated samples using appropriate measures.
 - i. Check for convergence: if the difference in losses from the previous iteration is below the convergence threshold, exit the loop.
3. End of algorithm. The generator and discriminator networks are now trained.

In our study, we conducted evaluation tests using the widely-used binary cross entropy loss function specifically designed for binary classification problems. This function assesses the disparity between predicted and actual binary classification outcomes, where each instance in the dataset has only two possible classes, typically labelled as zero or one. It quantifies the deviation between the predicted probability distribution and the actual probability distribution of binary classification results. It is computed as the negative log-likelihood of the actual class, given the predicted probability of that class as described in (1).

$$Loss = - [y \log(p) + (1 - y) \log(1 - p)] \quad (1)$$

where y is the actual class label, p is the predicted probability of the positive class, and the log is the natural logarithm. During training, the goal is to minimize the value of the loss function by adjusting the model's parameters, which helps to improve the model's ability to classify new examples correctly. According to the notations we settled, our GAN might be stated as a minimax with the value function $V(G, D)$ under the following equality:

$$Min_G Max_D V(D, G) = E_{z \sim P_z} [\log (1 - D(G(z)))] + E_{x \sim P_{data}} [\log (D_x)] \quad (2)$$

On the one hand, the G generator tries to understand the distribution by learning from the noise distribution and adjusting its own output to make it similar to the actual data distribution P_{data} . On the other hand, the discriminator D aims to distinguish between generated samples and real data by classifying them as real or fake. Both the generator and discriminator train their networks during the training phase, following (2). Consequently, the objective function of GANs can be formulated as a minimax game, involving the natural data distribution x , the expectation denoted by E , and the vector z originating from the random noise distribution P_{data} .

The elements P_z , $G(z)$, and D_x represent the generator's samples and the likelihood that D recognizes x as actual data, respectively. $D(G(z))$ represents the likelihood that D determines the data generated by G . To trick the generator G , the discriminator probability $D(G(z))$ must be maximized, therefore $\log(1 - D(G(z)))$ will be minimized. A cross entropy function is utilized to discriminate between $G(z)$ and x for the discriminator D , and D desires $V(D, G)$ to be maximized. In reality, G is established firstly, and then the parameters of discriminator D are adjusted to optimize D 's accuracy.

It is important to note that the type of malware does not matter since we deal with the requests of services asked by the applications to the kernel. The system calls were extracted from 5,000 samples installed in real devices. In our model based on reinforcement learning, the loss function plays a crucial role in updating the parameters of the value function. Through this process, the generator network learns to generate better actions or rules, leading to more significant rewards as it progressively refines the loss function. This iterative procedure empowers the value function's decision-making capabilities to evolve and improve over time.

3.4. System calls

System calls are functions and APIs that request services from the operating system's kernel. Security operating centers (SOCs) analyze malware to gain insight into the dynamic behavior of malicious apps. System calls are considered one of the most effective methods for classifying malware. The research emphasizes the importance of system calls as an effective strategy for malware classification. Several studies, including references [20], [21] recognize the use of system calls in recognizing and describing distinct forms of malware. These findings highlight the significance of system call analysis as a powerful and dependable way of detecting and categorizing malicious software. Our method employs the frequency of system calls across different categories of malware. In this study, we experimented with 140 system calls, and the results are presented in Table 1, which lists each system call with its corresponding value.

While making system calls, we discovered that some of these calls, including *Futex()*, were identified as vulnerabilities in the kernel. We investigated further and found that the code vulnerability is CVE-2021-3347, and the *Futex()* exploit is classified as a use-after-free (UAF) vulnerability. This occurs when a program pointer refers to a data set in dynamic memory that has already been erased [22]–[24]. To ensure the security and stability of the system, it is crucial to be aware of such vulnerabilities and take appropriate measures to mitigate them. This can include implementing patches and updates, using security-focused programming practices, and regularly monitoring the system for any suspicious activity.

Table 1. Example of the used system calls and the corresponding values

System call	Value
<i>Futex()</i>	Gives a mechanism for waiting till a specific condition is satisfied
<i>Recvfrom()</i>	Used to accept data on a socket regardless of whether it is connection-oriented
<i>Sigprocmask()</i>	Used to get and/or update the caller thread's signal mask
<i>Prctl()</i>	Manipulates many elements of the caller thread's or process's behavior
<i>Ioctl()</i>	Manipulates particular files underlying device parameters
<i>Sigprocmask()</i>	Used to get and/or modify the caller thread's signal mask

3.5. Malware category

The system calls can be categorized into four distinct groups: adware, ransomware, scareware, and SMS malware. For this study, we selected malware samples from 42 different families, as outlined in Table 2. The purpose of incorporating four varied types of malware samples is to ensure the dataset's diversity and comprehensiveness [25].

Table 2. Example of malware families

Malware category	Family
Adware	Shuanet family
	Youmi family
	Mobidash family
	Kemoge family
	Charger family
Ransomware	Jisut family
	Svpeng family
	WannaLocker family
Scareware	AndroidSpy.277 family
	FakeTaoBao family
	Penetho family
	FakeApp.AL family
SMS Malware	Nandrobox family
	Zsone family
	Jifake family
	Biige family

The generator creates new frequencies of system calls (S') that allow a malware classifier to identify malicious software as benign. The discriminator's role is to differentiate between the newly generated data (S') and the actual data (S) by comparing them. This process helps to distinguish between fake and actual data.

3.6. Simulation and dataset

The proposed model was implemented in Python 3.7.15 using the TensorFlow [26] and Keras [27] packages. We used the Adam optimizer with a learning rate of 0.0001 to optimize the neural network. Further details regarding the experimental setup can be found in the following section, which provides information on the specific settings used for the tests.

Activation functions: Sigmoid, ReLU

Loss function: Binary cross entropy

Input Layer: 136

Hidden Layers: 256

Output Layer: 36

Batch size: 100

Epochs: 1200

For our numerical experiments, we employed a Windows 10 machine equipped with an Intel Core i5-4210 CPU and 4 GB of RAM. To develop our model, we used the Keras deep learning library, specifically version 2.9.0, which facilitated the seamless implementation of neural network architectures. Furthermore, TensorFlow, version 2.9.2, served as the underlying framework, harnessing the power of GPU acceleration to expedite the training process.

To test our model, we focused on Android malware. Specifically, we trained the model using the CIC-AndMal2017 dataset [28], which includes over 10,854 samples from various markets, 4,354 malware and 6,500 benign samples. The dataset captures network traffic during three stages: installation, before the restart, and after the restart, using CICFLOWMeter and network traffic analyzers.

In our experiments, we used system call frequency across 11,599 applications. We divided the dataset into 80% training and 20% test sets. Table 3 shows the system call frequencies for selected malware and their families.

Table 3. List of the system calls used in the proposed model

System call	Description	Frequencies	Fake installer	Plnktom
Write	Write to a file descriptor	2,840	4,034	604
Access	Check user's permissions for a file	362	26	93
Getpid	Get process identification	1,120	1,796	1,221
getpriority	Gets the high priority by any process	134	41	26
gettimeofday	Get and set the time.	3,720	32	465
ioctl	Manipulates particular file's underlying device parameters	2,010	2,147	3,811
Getuid32	Get the effective user ID of the calling process.	1,110	1,371	1,871

4. RESULTS AND DISCUSSION

Our model employs two activation functions: the sigmoid and rectified linear unit (ReLU). We evaluated the effectiveness of our model by analyzing the loss functions of both the generator and discriminator, which are presented in our results. To further assess the quality of our model, we used the Frechet inception distance (FID) and inception score (IS) metrics, and the corresponding performance results are included. More details are presented below.

4.1. Tests performed with sigmoid function

Figure 3 depicts the generator and discriminator loss functions with the sigmoid activation function. The generator loss function decreases from 0.77 to 0.41 after 120 epochs, while the discriminator loss function decreases from 1.6 to 0.50 after 850 epochs. These observations indicate the model's effectiveness and highlight the significant progress made in reducing the loss over time.

4.2. Tests performed with ReLU function

Figure 4 illustrates the generator and discriminator loss functions achieved with the ReLU activation function. The generator loss function starts at 0.17, increases to 0.62, and then decreases to 0.34 at epoch 40. In comparison, the discriminator loss function starts at 9.7 and decreases to 2.1 after 20 epochs.

When comparing the generator loss function results between the two activation functions, we conclude that ReLU outperforms sigmoid. However, the discriminator loss function results are better with the sigmoid function. These observations highlight the importance of selecting the appropriate activation function for each model component to achieve optimal results.

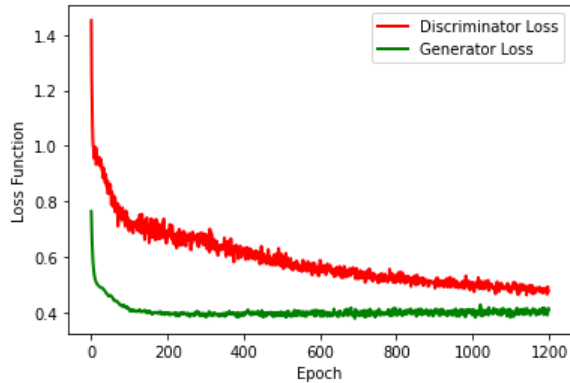


Figure 3. The generator and discriminator loss functions using sigmoid

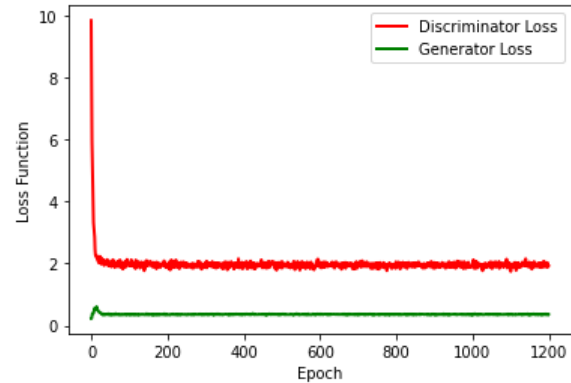


Figure 4. Generator and discriminator loss functions using ReLU

4.3. Evaluation metric FID

There are several commonly used methods for evaluating GANs. In this paper, we applied the FID to evaluate the performance of our model. FID is a metric widely used for assessing the quality of the produced pictures established expressly to assess the performance of the generative adversarial networks [29]. It measures the similarity between the generated samples' distribution and the training dataset's distribution in the feature space of a pre-trained deep neural network. A lower FID score indicates that the generated samples are more similar to the training dataset. The FID score is calculated using (3).

$$FID = \|m_x - m_y\|_2^2 + Tr(C_x + C_y - 2\sqrt{C_x C_y}) \quad (3)$$

where m_x and m_y refer to the feature wise mean of the real and generated images; the C_x and C_y are the covariance matrix for the real and generated feature vectors; and Tr is the trace of the matrix corresponding to the sum of the elements along the main diagonal of the square matrix.

We assessed the similarity between the distribution of adversarial frequencies and the distribution of the training malware dataset. The resulting score for 100 epochs is 2.34 in Figure 5, indicating a low value that suggests a slight difference between the adversarial frequencies and the actual frequencies employed by zero-day malware. This observation highlights the effectiveness of our model in generating adversarial examples that can closely mimic the behavior of real-world malware.

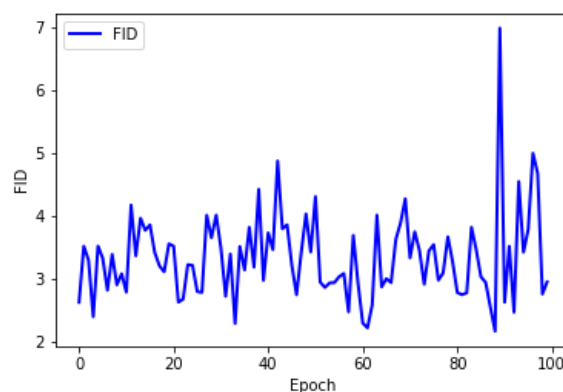


Figure 5. Representation of FID versus epochs

4.4. Inception score

The inception score is a metric used to evaluate the quality and diversity of generated images by a generative adversarial network [30]. It is calculated by feeding the generated images through a pre-trained convolutional neural network and measuring the Kullback–Leibler divergence (KL-divergence) between the conditional label distribution and the marginal label distribution of the CNN's SoftMax output. The inception score considers both the accuracy and diversity of the generated images, with higher scores indicating better quality and diversity.

We utilized our simulation's NumPy and Keras deep learning libraries to evaluate the inception score (IS). The obtained IS value for 100 epochs with the sigmoid activation function is 7.65, which we consider satisfactory. This indicates that our model can effectively generate synthetic images similar to actual ones.

5. CONCLUSION

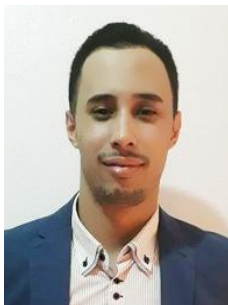
Over the last decade, mobile applications have experienced significant technological advancements. However, with the increasing prevalence of zero-day attacks, anti-malware systems need to be continuously strengthened to tackle these scalable threats. GANs provide a proactive approach to understanding how malware writers operate. During the training process, the generator creates new types of malwares, which are then used to train the discriminator to differentiate between benign programs and those infected with previously unknown viruses. In this study, we proposed a new model to detect zero-day Android malware using GANs by generating new feature vectors of the system call frequencies. We experimented with sigmoid and ReLU activation functions and evaluated the model's performance using the FID metric and IS score. Our results showed higher IS values and lower FID values (versus epochs), indicating the feasibility and potential of this approach. Overall, this study provides a promising solution for enhancing mobile security and reducing the impact of zero-day mobile malware. Our study demonstrated the potential of using GANs to detect mobile malware; in future work, we will develop an advanced intrusion detection system based on GANs to protect mobile applications against malware attacks, applying several loss functions and metrics to evaluate its performance. Furthermore, we will implement this IDS in a constrained environment, adding an extra layer of security to mobile devices.




REFERENCES

- [1] X. Liu and K. Liu, "A permission-carrying security policy and static enforcement for information flows in Android programs," *Computers & Security*, vol. 126, Mar. 2023, doi: 10.1016/j.cose.2022.103090.
- [2] S. Lazaar, "Contribution of wavelets to cybersecurity: Intrusion detection systems using neural networks," *General Letters in Mathematics*, vol. 10, no. 2, pp. 24–30, Jun. 2021, doi: 10.31559/glm2021.10.2.2.
- [3] P. R. K. Varma, K. P. Raj, and K. V. S. Raju, "Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Feb. 2017, pp. 294–299, doi: 10.1109/I-SMAC.2017.8058358.
- [4] K. Shaukat, S. Luo, and V. Varadharajan, "A novel deep learning-based approach for malware detection," *Engineering Applications of Artificial Intelligence*, vol. 122, Jun. 2023, doi: 10.1016/j.engappai.2023.106030.
- [5] S. Zhao, J. Li, J. Wang, Z. Zhang, L. Zhu, and Y. Zhang, "attackGAN: Adversarial attack against back-box IDS using generative Adversarial networks," *Procedia Computer Science*, vol. 187, pp. 128–133, 2021, doi: 10.1016/j.procs.2021.04.118.
- [6] D. Yin and Q. Yang, "GANs based density distribution privacy-preservation on mobility data," *Security and Communication Networks*, vol. 2018, pp. 1–13, Dec. 2018, doi: 10.1155/2018/9203076.
- [7] Z. Moti *et al.*, "Generative adversarial network to detect unseen internet of things malware," *Ad Hoc Networks*, vol. 122, Nov. 2021, doi: 10.1016/j.adhoc.2021.102591.
- [8] A. Ferdowsi and W. Saad, "Generative adversarial networks for distributed intrusion detection in the internet of things," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2019, pp. 1–6, doi: 10.1109/GLOBECOM38437.2019.9014102.
- [9] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "Fed-IIoT: A robust federated malware detection architecture in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 8442–8452, Dec. 2021, doi: 10.1109/TII.2020.3043458.
- [10] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, vol. 460–461, pp. 83–102, Sep. 2018, doi: 10.1016/j.ins.2018.04.092.
- [11] X. Hao, W. Ren, R. Xiong, T. Zhu, and K.-K. R. Choo, "Asymmetric cryptographic functions based on generative adversarial neural networks for internet of things," *Future Generation Computer Systems*, vol. 124, pp. 243–253, Nov. 2021, doi: 10.1016/j.future.2021.05.030.
- [12] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Android-GAN: Defending against android pattern attacks using multi-modal generative network as anomaly detector," *Expert Systems with Applications*, vol. 141, Mar. 2020, doi: 10.1016/j.eswa.2019.112964.
- [13] H. Li, S. Zhou, W. Yuan, J. Li, and H. Leung, "Adversarial-example attacks toward android malware detection system," *IEEE Systems Journal*, vol. 14, no. 1, pp. 653–656, Mar. 2020, doi: 10.1109/JSYST.2019.2906120.
- [14] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," in *International Conference on Data Mining and Big Data*, 2022, pp. 409–423.
- [15] D. Deb and M. M. Guirguis, "Use of auxiliary classifier generative adversarial network in touchstroke authentication," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2020, pp. 252–257, doi: 10.1109/ICMLA51294.2020.00049.
- [16] H. Rathore, A. Samavedhi, S. K. Sahay, and M. Sewak, "Robust malware detection models: learning from adversarial attacks and defenses," *Forensic Science International: Digital Investigation*, vol. 37, Jul. 2021, doi: 10.1016/j.fsidi.2021.301183.




- [17] M. A. Azad, F. Riaz, A. Aftab, S. K. J. Rizvi, J. Arshad, and H. F. Atlam, "DEEPESEL: A novel feature selection for early identification of malware in mobile applications," *Future Generation Computer Systems*, vol. 129, pp. 54–63, Apr. 2022, doi: 10.1016/j.future.2021.10.029.
- [18] M. Shahpasand, L. Hamey, D. Vatsalan, and M. Xue, "Adversarial attacks on mobile malware detection," in *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, Feb. 2019, pp. 17–20, doi: 10.1109/AI4Mobile.2019.8672711.
- [19] A. Chhaybi, S. LAZAAR, and M. Hassine, "A recent benchmark study of GANs models for securing mobile applications," in *Proceedings of the 6th International Conference on Networking, Intelligent Systems & Security*, May 2023, pp. 1–5, doi: 10.1145/3607720.3607730.
- [20] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting android malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, Aug. 2015, pp. 13–20, doi: 10.1145/2804345.2804349.
- [21] M. Nunes, P. Burnap, P. Reinecke, and K. Lloyd, "Bane or boon: Measuring the effect of evasive malware on system call classifiers," *Journal of Information Security and Applications*, vol. 67, Jun. 2022, doi: 10.1016/j.jisa.2022.103202.
- [22] T. expert team, "Yet another Futex vulnerability found in the Kernel (CVE-2021-3347)," *TuxCare*. Accessed: Nov. 03, 2022. [Online]. Available: <https://tuxcare.com/yet-another-futex-vulnerability-found-in-the-kernel-cve-2021-3347/>
- [23] CVE, "CVE-2021-3347," CVE, <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=futex> (accessed Nov. 03, 2022).
- [24] NIST, "National vulnerability database (NVD)." <https://nvd.nist.gov/vuln/detail/CVE-2020-14381> (accessed Jun. 06, 2023).
- [25] A. Guerra-Manzanares, M. Luckner, and H. Bahsi, "Android malware concept drift using system calls: Detection, characterization and challenges," *Expert Systems with Applications*, vol. 206, Nov. 2022, doi: 10.1016/j.eswa.2022.117200.
- [26] "TensorFlow." Accessed: Nov. 01, 2022. [Online]. Available: <https://www.tensorflow.org/>
- [27] "Keras." Accessed: Nov. 01, 2022. [Online]. Available: <https://keras.io/>
- [28] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCSST)*, Oct. 2018, pp. 1–7, doi: 10.1109/CCST.2018.8585560.
- [29] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, 2017.
- [30] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," *Advances in neural information processing systems*, 2016.

BIOGRAPHIES OF AUTHORS



Akram Chhaybi    is now pursuing his Ph.D. at ERMIA Team from ENSA of Tangier, AbdelMalek Essaadi University, Morocco. His primary research interests are mobile security, malware, and system security using artificial intelligence techniques. In 2020, he obtained his master's degree in cybersecurity and cybercrime from Morocco's National School of Applied Sciences of Tangier, AbdelMalek Essaadi University. He can be reached at akramchhaybi1@gmail.com.



Saiida Lazaar    holds a Ph.D. in applied mathematics from Aix Marseille I University in France and currently serves as a full professor at AbdelMalek Essaadi University in the Department of Mathematics and Computer Sciences, at ENSA of Tangier, Morocco. With extensive expertise in the field of cybersecurity, Dr. Lazaar has played a role as the head of the master's program in CyberSecurity and CyberCrime. Her track record includes notable research positions such as CNRS and IFP in France, as well as ONDRAF in Belgium. Throughout her career, she has demonstrated an exceptional passion for advancing the field of cybersecurity. She can be reached at slazaar@uae.ac.ma or s.lazaar2013@gmail.com.