Journal of Computer Science and Cybernetics, V.41, N.2 (2025), 135–161
DOI no. 10.15625/1813-9663/22522

REVIEW

A SURVEY ON GENERATIVE ADVERSARIAL NETWORKS FOR MALWARE ANALYSIS

LAM BUI THU[1], HOANG THANH NAM[1,*], PHAM DUY TRUNG[1], NGUYEN LE MINH[2]

[1] *Academy of Cryptography Techniques, 141 Chien Thang Street, Tan Trieu Commune, Thanh Tri District, Ha Noi, Viet Nam*
[2] *Japan Institute of Advanced Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan*

Abstract. Generative Adversarial Networks (GANs) have recently become an interesting subject for researchers due to their diverse applications across various fields. Initially focused on image-related tasks, they then have been used to generate new synthetic data for applications across many areas of machine learning research. In malware analysis, GANs have rapidly expanded and are used to generate adversarial data for enhancing the effectiveness of malware detection systems. This paper reviews the application of GANs in malware analysis to generate adversarial examples, to modify semantic information within data, to augment datasets for rare classes, and to support representation learning. The paper provides an extensive overview that serves both as a primer for cybersecurity specialists and a resource for machine learning researchers. The paper outlines the fundamentals of GANs, their operational mechanism, the current types of GANs, the challenges faced, and future directions for exploration in malware analysis.

Keywords. Generative adversarial network, malware analysis, cybersecurity.

1. INTRODUCTION

The malware is the top threat to computer systems. Recent studies have shown that approximately 190,000 malware attacks occur every second, with nearly 90% of which are phishing attacks using social engineering [1]. Recently, with the digital transformation, new types of malware have increasingly appeared. In addition, the prevalence of mobile malware and document-based malware is increasing rapidly. Statistics highlight the following trend: (1) by the end of 2023, cryptojacking attacks increased to over 1 billion, likely correlated with Bitcoin's value, which reached an all-time high [1]; (2) the number of encrypted threats has more than doubled over the past year, especially in the retail, government, and education sectors [1]; (3) malicious code is frequently transmitted through PDF files, particularly in social engineering attacks like phishing schemes that imitate "expiring password" notifications [2]; (4) current security incidents impacting well-known companies remain a concern in 2024, including the breaches at SpaceEyes and AT&T [3].

*Corresponding author.
E-mail addresses: lambt@actvn.edu.vn(L.B. Thu); hoangthanhnam@actvn.edu.vn(H.T Nam); trungpd@actvn.edu.vn(P.D Trung); nlminh2007@gmail.com(N.L Minh).

In recent years, a large number of researchers have focused on GANs, which are originally introduced in 2014 by Ian Goodfellow and his team [4]. The GAN model is grounded in game theory and represents a fascinating recent advancement in machine learning. Like generative models, GANs produce new data instances that mimic training data. For instance, GANs can generate images that resemble photographs of human faces, even if those faces do not correspond to real individuals. The applications of GANs cover diverse fields, from general areas such as image synthesis, image-to-image translation[5], video generation [6], generating images from text[6], language generation, data augmentation, to more specific applications in agriculture, medical imaging and drug discovery. In cybersecurity, various initiatives have leveraged GANs, including intrusion detection systems (IDS)[7], mobile and network security[8], BotNet detection and malware analysis [9], etc...

Our paper focuses on the use of GANs for malware analysis, which addresses the challenges and limitations of employing GANs and suggests potential avenues for further research. The paper will provide an in-depth examination of how GANs can strengthen cybersecurity measures. It highlights the need for further work to exploit these capabilities. In detail, the paper first examines the core principles of typical adversarial algorithms and categorize adversarial attacks within the malware sector, considering different attack vectors. Given that Windows malware is the most common and frequently targeted area, these attacks are further differentiated according to the algorithms used. Then, this study investigates attacks on less typical file formats, including those associated with Android. In addition, it addresses the challenges and constraints of applying existing adversarial attack-based GANs in practical settings. Finally, the paper emphasizes future research avenues focused on creating more effective, resilient, efficient, and generalized adversarial attacks on malware classifiers.

The paper is organized as follows: Section 2 offers a background overview of malware, covering detection methods and the challenges faced in malware analysis. Additionally, we will introduce the concepts of GAN composition and principles, along with a review of common problems encountered by GANs and the various solutions proposed for each issue in Section 3. Section 4 examines the application of GANs in malware analysis, focusing on their use in cybersecurity, particularly for Windows and Android malware, and compares and evaluates significant studies on GANs. Section 5 will discuss the challenges currently faced in the field and explore potential directions for applying GANs in malware analysis.

2. MALWARE ANALYSIS

2.1. Overview

Malicious software, often called malware, is a key factor in most computer breaches and security events. Any program that causes damage to an individual, computer, or network can be categorized as malware [10]. Malicious software can be embedded in various binary formats, each functioning in distinct ways. These formats include the Windows PE files encompassing Portable Executable files with extensions such as .exe, .dll, the Unix ELF files being used as Executable and Linkable Format files in Unix operating systems, the Android APK files referring to Android Package Kit files, which typically have the .apk extension, and additionally, document-based malware can be found in files with extensions such as .doc, .pdf, and .rtf. It is important to note that malware can also be embedded within extensions and plug-ins for popular software platforms, including web browsers and frameworks. The

details of malware types can be found in Table 1.

Malware analysis assesses the effect, source, and purpose of the malware and aims to (1) malware detection: identifying if a sample is harmful; (2) malware similarity analysis: finding commonalities among malware samples, for instance, to evaluate whether a new sample resembles those previously encountered, this aim has multiple versions, such as variant detection, similarity detection, family detection and difference detection; and (3) malware category detection: identifying the particular type of malware based on a specific malware taxonomy.

Table 1: Types of malware

Type	Actions	Real-work example
Ransomware	Prevents the victim from accessing their data until a ransom is paid.	RYUK
Fileless Malware	Modifies files that are inherent to the operating system.	Astaroth
Spyware	Gathers user activity information without their awareness.	DarkHotel
Adware	Delivers undesirable promotions.	Fireball
Trojans	Camouflages itself as appealing code.	Emotet
Worms	Disseminates across a network by copying itself.	Stuxnet
Rootkits	Allows cybercriminals to control a victim's device from a distance.	Zacinlo
Keyloggers	Tracks the keystrokes made by users.	Olympic Vision
Bots	Launches a broad flood of attacks.	Echobot
Mobile Malware	Infects smartphones and tablets.	Triada
Wiper Malware	Eliminates user information to a point where recovery is impossible.	WhisperGate

Malware analysts play a critical role in investigating security in order to understand the specific actions that a potentially harmful binary can execute, establish methods for its detection within the network, and evaluate the extent of its impact to contain any resulting damage. After dissecting the samples thoroughly, analysts create the malware signatures that facilitate the detection of malware infections on the network. The insights from malware analysis are essential for creating host-based and network signatures, enhancing the overall security posture.

Host-based signatures (or indicators, patterns) are employed to detect malicious codes on compromised computers by identifying files created or modified by malware and changes to the system registry. Unlike antivirus signatures focusing on malware characteristics, these indicators track malware's actions, making them more effective in detecting malware that changes form or has been deleted. Meanwhile, network signatures are used to identify malicious codes by monitoring network traffic. While signatures can be created without malware analysis, those developed with it are usually more effective, leading to higher detection rates and fewer false positives. Ultimately, the goal is to understand how the malware operates, a question frequently posed by senior management following a major security breach [11].

2.2. Malware analysis techniques

Malware analysis can be divided into two primary approaches: static and dynamic. Static analysis examines the malware's characteristics without executing it, while dynamic one involves executing the malware to observe its behavior. Each of these approaches can be further

categorized into basic and advanced techniques, allowing for a comprehensive evaluation of potential threats [11].

2.2.1. Static analysis

Basic static analysis entails analyzing an executable file without running it or directly inspecting its instructions. This approach can help identify whether a file is malicious and offer insights into its functionality. It may also aid in the creation of simple network signatures. While fundamental static analysis is relatively quick and easy to perform, it is often insufficient against more advanced malware and may miss significant behaviors. On the other hand, advanced static analysis entails reverse-engineering a malware by loading its executable file into a disassembler and scrutinizing the program's instructions to understand its behavior. As the CPU executes these instructions, advanced static analysis affords a comprehensive understanding of the program's functionality. Nonetheless, this approach presents a steeper learning curve than fundamental static analysis and necessitates specialized knowledge in disassembly, coding constructs, and concepts pertinent to the Windows operating system.

2.2.2. Dynamic analysis

Basic dynamic analysis techniques involve executing malware to observe its behavior on a system, either removing the infection, creating effective signatures, or both. However, before safely running malware, it is crucial to establish a controlled environment that allows for thorough examination without risking damage to your system or network. Similar to basic static analysis techniques, most individuals can employ basic dynamic analysis methods without requiring extensive programming knowledge. Nevertheless, it is important to recognize that these techniques may not be effective for all types of malware and may overlook critical functionalities [12].

Advanced dynamic analysis entails utilizing a debugger to assess the internal state of a running malicious executable. This methodology provides a robust means of extracting intricate details that may prove challenging to obtain through alternative techniques.

2.3. Malware detection

Current malware detection techniques are typically categorized into two main approaches: signature-based and behavioral-based methods. Traditionally, signature-based methods serve as the primary means of malware detection. However, due to their inability to identify zero-day attacks, there has been a notable shift towards dynamic and online detection based on behavior. Many modern practical solutions utilize hybrid methods combining signature and behavioral techniques. The following sections will explore the various malware detection approaches in detail [13].

2.3.1. Signature-based malware detection

A signature is defined as a short sequence of bytes unique to each type of malware, facilitating its identification among various files. These detection methods depend on maintaining a comprehensive database of malware signatures, which leads to a notably low rate of

false positives. While signature-based detection has proven effective and efficient in identifying known malware, it has certain limitations, particularly its inability to detect previously unknown malware. Figure 1 represents the malware detection process utilizing signatures.

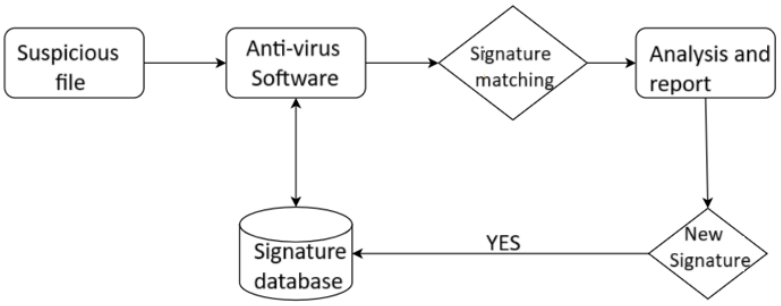

Figure 1: A malware detection based on signature

The signature database consists of a comprehensive list of all potential malware signatures and plays a crucial role in malware detection. When the anti-malware engine identifies malicious objects, it updates the signature database with the corresponding malware signatures for future reference. A robust malware detector features signature database rich in signatures, enabling it to effectively recognize various forms of malware.

Signature-based malware detection excels in speed, operational efficiency, and accessibility. However, its inability to identify zero-day malware—those threats lacking a known signature in the anti-malware engine's database—raises concerns about its reliability. Attackers can easily extract digital signature patterns, enabling them to obscure the actual signature of the malware. Modern malware often exhibits polymorphic and metamorphic traits, allowing it to alter its behavior and file signature seamlessly. Consequently, signature-based detection is heavily dependent on known malware, proving ineffective against zero-day attacks or variations of existing threats. The signature database is also expanding rapidly, keeping pace with the alarming proliferation of malware families.

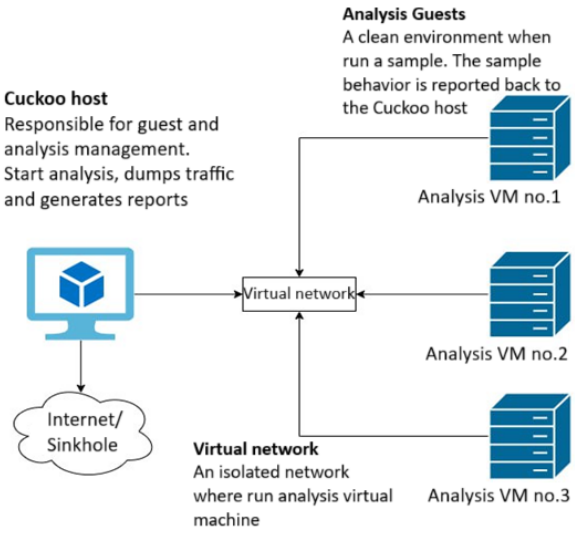

Figure 2: The architecture of Cuckoo sandbox

2.3.2. Behavioral-based detection

Dynamic methods for malware analysis involve executing potentially harmful files within isolated virtual environments, commonly known as sandboxes. This methodology enables researchers to identify a malware based on its behavior during execution while safeguarding the primary system from potential threats. During this process, the malware may manipulate registry keys and attempt to gain elevated privileges within the operating system. As the malware operates, various operating system characteristics may change; these alterations are meticulously recorded by monitoring agents in the controlled environment.

Each analysis (see Figure 2) is performed on a dedicated and isolated virtual or physical machine. Cuckoo's infrastructure consists of two main components: The *Host* machine, which acts as the management software, and several *Guest* machines designated for the analysis process. These *Guest* machines can either be virtual or physical [14]. The *Host* machine executes the core components of the sandbox, overseeing the entire analysis process. In contrast, the *Guest* machines serve as isolated environments where malware samples are safely executed and analyzed. The following Figure 2 depicts Cuckoo's primary architecture [14].

Dynamic analysis effectively captures real-time indicators, including API (application programming interface) calls, registry modifications, file locations, domain names, dumps traffic, generates reports and other relevant system metrics.

2.4. Malware classification

The classification of malware is an important goal in malware analysis. This process involves organizing different malware samples into categories based on shared characteristics. Depending on a specific task at hand, malware analysts might select different methods for classifying malware [15]. For instance, security operations teams may classify malware based on its severity and function to efficiently manage the risks it poses to an organization; security response teams may categorize malware according to its potential damage and entry points to formulate remediation and mitigation plans; meanwhile malware researchers could organize malware by its origins and authorship to gain insights into its lineage and intent.

In general-purpose malware analysis, it is a standard procedure to group samples into families. Malware analysts employ this classification approach to monitor authorship, link relevant information, and identify new variants of recently discovered malware. Malware samples that belong to the same family may exhibit shared code, capabilities, authorship, functions, purposes, or origins.

Nevertheless, attributing malware to specific families poses a considerable challenge, with results often differing based on the definitions and techniques utilized for classification by the analyst. Traditionally, this process depends on signature matching. By evaluating the characteristics and behaviors of previously identified malware, analysts can compare incoming binaries against this dataset to determine if they align with known samples.

2.5. Machine learning in malware analysis

An essential phase in detecting and classifying malware using machine learning techniques involves the practical training of the models. This training process necessitates rigorous preprocessing of malware datasets to create accurate and informative feature vectors. It is a common practice to employ a combination of feature extraction and feature selection

techniques to construct a robust feature vector space (see Table 2). Feature extraction involves identifying and isolating relevant characteristics from the raw data and transforming it into a structured format suitable for analysis. This can include methods such as analyzing byte sequences, opcode frequency, or behavior-based features, depending on the type of malware and the detection model being utilized [15].

Table 2: Key features for malware analysis

Features	Description	Features for ML-based analysis
Byte sequences	Identifying the frequency of occurrence for specific sequences and combinations of n bytes.	n-grams
Opcodes	Machine-level interaction of PE	Opcode sequences
API and system calls	Compilation of calls made and referenced by the PE file.	Execution traces
Network activity	Collaboration between PE and the network concerning the C&C server.	Protocols, DNS interactions, HTTP requests, TCP/UDP port
File system	Malware performs certain file operations to establish persistence on a system.	Number of files created, modified, encrypted, deleted
CPU registers	Records of concealed registers utilized by the PE.	FLAG register values
PE file characteristics	Details regarding the sections of the PE file.	Imports, compilers, symbols
Strings	Strings found in a PE file that hold information related to	Signatures, file names, code fragments
Control Flow Graphs (CFGs)	Details regarding the structure of the program.	Functions, control relationships, code sequences

Once the relevant features have been extracted, feature selection comes into play. This process entails evaluating the various features to determine which ones contribute the most to the model's ability to differentiate between malware and benign software. By selecting the most significant features, this method can enhance the model's performance, reduce dimensionality, and prevent overfitting, all of which are critical for achieving accurate malware detection and classification outcomes.

2.6. Challenges in malware analysis

Malware are developed with particular objectives, such as keystroke logging, establishing backdoor access, or utilizing a target system to send excessive emails that can disrupt servers. Malware authors frequently go beyond these fundamental functions by implementing advanced techniques to obscure their activities from users or system administrators. This may involve rootkits, process injection, or strategies designed to evade analysis and detection. The field of malware analysis presents a range of challenges, which can be summarized as follows:

Anti-disassembly techniques that deter disassembly aim to postpone or obstruct the examination of harmful code. Any code that runs without issues can be reverse-engineered. However, by equipping their code with anti-disassembly and anti-debugging measures, malware creators raise the expertise the malware analyst needs. The investigative process, which is often time-sensitive, is complicated by the analyst's difficulty in comprehending the malware's functionalities, extracting essential host and network signatures, and creat-

ing decoding algorithms. These extra protective measures may deplete the skills available within many organizations, necessitating the involvement of expert consultants or extensive research efforts to reverse-engineer the code.

Malware frequently employs anti-debugging techniques to identify when it is being analyzed with a debugger and to obstruct the debugging process. Malware creators understand that malware analysts depend on debuggers to grasp how the malware operates, so they incorporate anti-debugging tactics to hinder the analyst's work. When malware recognizes that it is running under a debugger, it may alter its usual code execution flow or change the code in a manner that leads to a crash. This disrupts the analysts' efforts to study the malware, ultimately prolonging the time and resources required for their investigation. Malware uses a variety of methods to determine if a debugger is present. These techniques involve leveraging the Windows API, inspecting memory structures for indications of debugging activity, and looking for traces left by a debugger on the system. Recognizing debuggers is one of the most prevalent anti-debugging approaches utilized by malware. Methods to counteract virtual machines are employed to hinder analysis attempts. These methods allow malware to determine if it operates within a virtual machine environment. If a virtual machine is recognized, the malware may alter its actions or choose not to run. This scenario poses considerable difficulties for analysts.

Packers, also called packing programs, have gained significant popularity among malware creators because they aid in hiding malicious software from antivirus tools, hinder malware analysis, and decrease the size of harmful executable. Most packers are easy to use and available at no cost. Fundamental static analysis proves ineffective on packed applications; packed malware must be unpacked first for static analysis to occur, complicating the evaluation process. Two primary purposes for using packers on executable are to reduce program sizes or avoid detection and analysis. While there are numerous types of packers, they typically operate similarly: they transform an executable file into a new version that houses the transformed executable within it, along with an unpacking stub that the operating system calls. When malware is packed, analysts generally can only access the packed file and not review the original unpacked program or the packing software itself. To unpack an executable, it is necessary to reverse the actions performed by the packer, which requires a solid understanding of the packer's mechanics [11].

Polymorphic malware is regarded as a complete entity equipped with a compiler capable of decrypting and obfuscating the code, followed by recompiling it as a whole. The unencrypted virus body generates a new mutated decryptor utilizing a random encryption algorithm and enables the decryptor to self-encrypt before reconnecting both parts. Nevertheless, the fundamental emulation issue persists: the virus code section is decrypted in memory, rendering it detectable and flaggable by security analysts [16].

The emergence of metamorphic viruses introduced the idea that no two generations of these viruses can have similar signatures. This differs from polymorphic malware, which maintains a consistent structure. In contrast to polymorphic malware, a metamorphic virus completely alters and obscures its entire code. An essential issue for malware analysts is the lack of sample data for malware analysis. There are many different strains of malware, but collecting samples faces significant challenges due to the scarcity of available samples. Therefore, additional tools, such as machine learning applications, are needed to analyze malware.

3. GENERATIVE ADVERSARIAL NETWORKS

3.1. Overview

A GAN comprises two fundamental neural network models: the generator (referred to as G) and the discriminator (referred to as D). It operates on the principle of a zero-sum game played between the generator and the discriminator, aiming to make the samples produced by the generator closely resemble the original input data [4]. The readers are recommended to look at Table 3 for all relevant notations.

Table 3: GAN's notations

Notation	Description
G	Generator responsible for producing synthetic (fake) data samples.
D	Discriminator that assigns a probability score (between 0 and 1) indicating whether an input is real or fake.
x_{data}	A real data sample drawn from the original dataset.
P_{data}	The underlying probability distribution from which the real data x_{data} is sampled.
z	A noise vector sampled from a predefined latent space; serves as input to the generator
P_z	The distribution from which the latent noise vector z is drawn.
$G_{(z)}$	A generated data instance produced by feeding z into the generator G .
P_g	The distribution learned by the generator; ideally approximates P_{data}
D_x	The discriminator's confidence score that input x is a real sample.
D^*	The discriminator's optimal state after convergence during training.
G^*	The generator's optimal state when it successfully mimics the real data distribution.

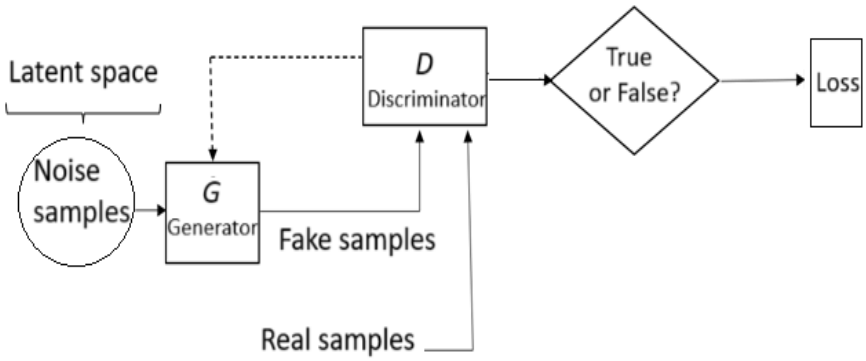

Figure 3: A snapshot of the GAN architecture

As depicted in Figure 3, G employs a generation process (typically by introducing a Gaussian noise function) to create a set of samples known as "fake samples" from a predefined latent space. Subsequently, G submits these fake and real samples to D, which assesses how realistic the fake samples appear. Following this, D produces an output of either 0 or 1, based on the sample's authenticity, where "0" denotes "fake" and "1" signifies "real" The concept of adversarial training entails that G strives to convince D to classify the samples it generates as true. At the same time, D aims to classify those generated samples as false whenever possible. Ultimately, after the training, D and G may reach a Nash equilibrium.

The training process is described in Algorithm 1, where two networks are manipulated interchangeably. The overarching goal of adversarial training is to refine the generator so

that the distribution of its data generated P_g closely matches the original, accurate data distribution P_{data}. This matching is crucial for the generator to produce realistic samples that are indistinguishable from actual data.

Meanwhile the discriminator network operates as a binary classifier. Its primary function is to differentiate between the actual data samples, which are expressed as $x \sim P_{\text{data}}(x)$ and the synthetic samples generated by G, represented as $G(z) \sim P_g(G(z))$. The discriminator's effectiveness plays a critical role in the training of the GAN. It provides feedback to the generator on the quality of the generated outputs, ultimately driving the entire process toward key objectives, such as achieving high fidelity in the generated samples and improving the discriminator's ability to distinguish between real and fake data.

Algorithm 1 Training algorithm of original GANs

1: **for** number of training iterations **do**
2: **for** k steps **do**
3: Sample minibatch of m noise samples from noise prior $P_z \rightarrow$ fake_data
4: Sample minibatch of m examples from data generating distribution $P_{\text{data}} \rightarrow$ true_data
5: Discriminator training (D, G, fake_data, true_data)
6: **end for**
7: Sample m samples from $P_{\text{data}} \rightarrow$ true_data
8: Generator training (G, D, fake_data)
9: **end for**

Note that the variable z represents the random noise vector serving as input to the generator G. This generator is designed to create synthetic data distributions that resemble real-world data. The random noise vector samples are drawn from $z \sim P_z$ where the distribution P_z can be either a uniform (providing a constant probability across a specified range) or a normal (centered around a mean with values tapering off as they move away from that center). Once processed by the generator G, the output $G(z)$ follows a specific distribution denoted as P_g.

The overall loss function for GAN training is as follows

$$\min_G \max_D V(D,G) = E_{x\sim P_{\text{data}}(x)}[\log(D(x))] + E_{z\sim P_z}[\log(1 - D(G(z)))]. \tag{1}$$

The term $\log(D(x))$ in the first expression indicates $D's$ assessment of genuine data, while the second term $\log(1 - D(G(z)))$ reflects $D's$ evaluation of the synthesized data. $D's$ objective during training is to categorize the training samples with the highest probability, while G aims to reduce $\log(1-D(G(z)))$, effectively seeking to increase $D's$ loss. Throughout the training phase, the network's parameters are adjusted by keeping G and D fixed alternately. Ultimately, G becomes capable of estimating the distribution of the sample data. This process resembles a minimax game where G and D are optimized in turn to develop the necessary generative and discriminative networks until they reach a Nash equilibrium. Algorithm 1 outlines the training procedure for the original GAN. In lines 2–6, D undergoes training for k iterations based on a batch of m samples, and once the training is concluded,

the optimal D is achieved, as demonstrated in Equation (2)

$$D^*(x) = \frac{P_{\text{data}}(x)}{P_g(x) + P_{\text{data}}(x)}. \tag{2}$$

Subsequently, as demonstrated in lines 7–8, G is trained while keeping D constant. Ideally, when G achieves optimal training, the distribution $P_g(x)$ will closely match $P_{\text{data}}(x)$, at which point $D^*(x)$ in Equation (2) will be equal to $1/2$. Once D is functioning at its best, $D^*(x)$ can be substituted into Equation (1), enabling us to derive $G's$ loss function as outlined in Equation (3)

$$G^* = 2D_{JS}(P_{\text{data}}(x) \| P_g(x)) - 2\log 2. \tag{3}$$

At this stage, the Jensen-Shannon (JS) divergence between the generated data distribution and the real data distribution has been effectively minimized. A key factor contributing to the success of Generative Adversarial Networks (GANs) is their ability to address the limitations of Kullback-Leibler (KL) divergence, which is asymmetrical and inadequate as a distance measure. The gradient descent strategies used in these processes can be any stochastic gradient method, such as the Adam optimizer or Momentum [17].

3.2. Selected GAN models

3.2.1. Deep convolutional GAN (DCGAN)

The deep convolutional generative adversarial network, known as DCGAN, was first presented in 2015. The foundation of DCGAN was derived from research related to convolutional neural networks and their capacity to enhance various models in machine learning. This study focused on generating pseudo-natural images, showcasing the capabilities of GAN models in image generation tasks. While the majority of deep learning algorithms operate as black-box systems, it is possible to examine the internal workings of a convolutional neural network (CNN) model through careful adjustments. The authors made several alterations to the standard CNN architecture informed by findings from various research papers [18].

3.2.2. Conditional GAN

Generative adversarial networks can be adapted to function as a conditional model by incorporating additional information into both the generator and discriminator. This information could represent various auxiliary data, including class labels or information from different modalities. Conditioning can be achieved by integrating y as an extra input layer for both the generator and discriminator [19].

In the generator, the input noise $P_z(z)$ and y are merged to form a joint hidden representation, and the adversarial training framework provides significant flexibility in the construction of this hidden representation. The discriminator takes both x and y as input to a discriminative function, which is again realized by a multi-layer perceptron (MLP). The objective of the two-player minimax game can be described as follows

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}\big[\log D(x|y)\big] + E_{z\sim p_z(z)}\big[\log(1 - D(G(z|y)))\big]. \tag{4}$$

3.2.3. CycleGAN

The primary function of the model is to serve as a mechanism for image translation. Its main objective is to transform an 'unpaired' image from one domain to another. Introduced by Zhu et al. in 2019 [20], CycleGANs have emerged as a dependable resource in image processing and have aided researchers across various fields. In the context of security, it is crucial to recognize that CycleGAN models have applications in biometrics, especially in facial recognition. Recently, a variant of CycleGAN has been introduced for translating videos, termed Mocycle-GAN. This development opens up the potential for incorporating this type of GAN into CCTV systems for facial recognition purposes.

3.2.4. LSGAN

The least square GAN (LSGAN) as a variant of generative adversarial networks (GANs) is utilized the least squares equation for its discriminator. This innovative approach helps minimize the Pearson X^2 divergence. The Pearson X^2 divergence is a specific type of f-divergence. By employing the least squares function, the model effectively distances correctly classified samples from the real data, which improves the performance of the classifier and enhances the training efficiency of the generator. Equation (5), (6) shows objective functions for the LSGAN model, as referenced in [21].

$$\min_D V(D)_{\text{LSGAN}} = \frac{1}{2}E_{x\sim p_{\text{data}}}\left[(D(x)-b)^2\right] + \frac{1}{2}E_{z\sim p_z}\left[(D(G(z))-c)^2\right], \tag{5}$$

$$\min_G V(G)_{\text{LSGAN}} = \frac{1}{2}E_{z\sim p_z}\left[(D(G(z))-c)^2\right]. \tag{6}$$

3.3. GAN training issues

Given the unique characteristics of GANs outlined earlier, there are several issues of GAN training that require careful consideration [17].

3.3.1. Mode collapse

The goal of producing synthesized data from a latent space necessitates generating high-quality data and ensuring diversity and generalization across various synthesized samples. In addition, GAN models should be able to reproduce data that has not been encountered before. However, it is not always the case, there is an issue related to this and called "mode collapse". The mode collapse takes place when identical outputs are produced from different inputs within the latent space. Research has indicated a connection between the quality and diversity of GANs. Although numerous attempts have been made to tackle mode collapse, it continues to be a persisting issue.

In practice, a GAN model rarely generates the same output for distinct inputs; this situation is referred to as complete mode collapse. While complete mode collapse is infrequent, partial mode collapse is more widely encountered, wherein many outputs are the same. For example, in the context of image generation, partial mode collapse may surface when various outputs exhibit the same color or texture. Investigations have shown that mode collapse

impacts the convergence of GANs, even if an optimal solution is reached. Several recent variants of GANs have been introduced to help alleviate the mode collapse issue. For instance, it has been shown that the Wasserstein GAN (WGAN) effectively minimizes mode collapse.

3.3.2. Gradient vanishing

The training of a GAN must be well-balanced, requiring both the generator G and discriminator D to learn together in a synchronized manner. A highly accurate discriminator can effectively distinguish between genuine and generated data, represented as $D(x) = 1$ and $D(G(z)) = 0$. In this scenario, the loss function approaches zero, leading to gradients that are nearly zero and offering minimal feedback to the generator. Conversely, a poorly performing discriminator is unable to tell the difference between real and generated data, thus giving the generator ineffective information.

3.3.3. Instability

The distinctive characteristics of GANs render the interaction between the two models quite intricate as they learn from one another. The training of GANs functions in which both networks compete to find their results, essentially participating in a minimax game. This model architecture depends on collaboration to enhance the overall loss function; however, the goals that the discriminator (D) and generator (G) aim to optimize are fundamentally conflicting. Given the nature of these networks' objective functions, minor adjustments in one network can result in substantial changes in the other, initiating a series of cascading effects.

During training, when the two networks start to lose synchronization, the learning process becomes unstable; significant fluctuations in gradients may cause one of the networks to lose its learning capacity. It is crucial to understand that periods of instability often lead to further instability, exacerbating the problem. Although networks can sometimes recover from instability, doing so typically means sacrificing training efficiency. Numerous recently introduced GAN architectures concentrate on stabilizing their training processes. By achieving a more consistent training procedure, superior network performance is generally secured; this emphasis on stability is why recent innovations in the field often focus on enhancing training stability.

3.3.4. Stopping problem

Conventional neural networks are required to minimize a loss function in a steadily reducing manner, theoretically reflecting the cost function. In contrast, GANs engage in a minimax game for their optimization, which disrupts this monotonic behavior. During the training of GANs, the loss function displays no consistent pattern, making it impossible to gauge the networks' status based solely on this function. As a result, it is difficult to determine when the models have reached complete optimization during the training process.

3.4. Appications of GANs

3.4.1. GANs on augmenting the training process

One of the most significant challenges in machine learning for cybersecurity is the imbalance in training datasets. Malware samples are often limited in number compared to benign software, leading to models that are biased toward the majority class. This imbalance hinders the model's ability to generalize effectively. GANs present a promising solution by generating synthetic malware samples that mimic the characteristics of real malware. This approach enriches the training dataset and provides a more balanced representation of both benign and malicious classes.

Machine learning methods, especially deep learning approaches, require vast data to perform effectively. Data augmentation, or oversampling, is a common strategy to address data deficiencies, helping to prevent model overfitting and mitigate data imbalance. Imbalanced datasets—where one class (the majority) vastly outnumbers the other (the minority)—pose a significant challenge. In binary classification, for example, the model tends to favor the majority class, which results in skewed performance. Two primary techniques are used to address this: under sampling the majority class or oversampling the minority class through methods such as GAN-generated synthetic data.

Furthermore, adversarial examples can be generated by GANs, which can be critical in assessing the performance of detection systems under adversarial conditions. Hu and Tan [22] emphasized this utility, noting that "the generated adversarial examples can be used to evaluate the performance of malware detection systems under adversarial conditions." This highlights the dual utility of GANs—not only do they augment training data, but they also help assess the robustness of detection mechanisms.

3.4.2. Representation learning

GANs are primarily designed for data augmentation, which involves generating new data to enhance an existing dataset. They are also utilized in representation learning, helping to extract meaningful features or patterns necessary for building classifiers or other predictive models. By taking advantage of their ability to model complex data distributions, GANs can learn normal and abnormal data characteristics. The generator and discriminator components of GANs can effectively model the data properties of specific categories, which can then be used to identify instances that deviate from expected patterns.

MIGAN is a method that depicts malware binaries as grayscale or color visuals to recognize and extract distinctive features for classification purposes. This approach proves to be more effective than conventional machine learning-based malware detection methods, which often necessitate considerable domain knowledge and lengthy behavioral analysis to identify unique traits. MIGAN framework is tailored for malware image generation. It can rapidly generate high-quality synthetic malware visuals and categorize malware samples into various families. It comprises a generator and discriminator network as well as a classification module. The innovation of this method lies in its distinctive GAN architecture, hybrid loss function, newly created dataset, and classification network design [23]. More detail of MIGAN will be given in Section 4.

4. GANS FOR MALWARE ANALYSIS

GANs have greatly advanced the fields of artificial intelligence, particularly in computer vision and natural language processing. Recently, their application in malware analysis has attracted increasing attention. This section explores the various roles of GANs in malware detection, generation of malware data, and their implications in related areas, supported by findings from extensive scholarly research.

Firstly, for malware detection, Sonam Bhardwaj et al. [24] introduce MD-ADA see Figure 4, a framework that employs adversarial domain adaptation (DA) to tackle these difficulties. This approach alters a source dataset for use in a target domain. MD-ADA leverages image representations of malware binaries extracted from Windows PE files via a convolutional neural network (CNN) to achieve lossless image embedding. It incorporates a GAN for classification, enabling it to function effectively even in situations with limited labeled data and varying feature distributions. Experimental results show that MD-ADA outperforms the fine-tuning method, attaining an accuracy of 99.29% on the BODMAS dataset, 89.3% on the Malevis dataset under consistent feature distribution, and 90.12% on the CICMalMem2022 dataset (Target) and 83.23% on the Microsoft Kaggle dataset (Target) for inconsistent feature distribution. The recorded F1-scores of 99.13% and 87.5% for consistent feature distributions, along with 91.27% and 81.7% for inconsistent distributions, indicate that the performance of MD-ADA is commendable for both types of data distributions, particularly when the target domain has very few labeled samples.

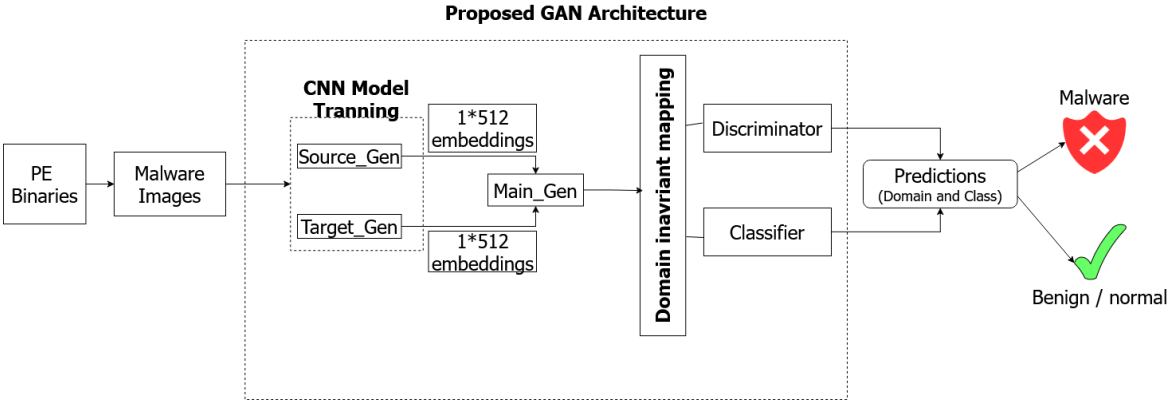

Figure 4: The framework of MD-ADA

Caixia Gao et al [25] propose a new classification model based on an improved lightweight neural network in Figure 5 that can effectively improve the execution efficiency and detection performance of malware detection methods against adversarial malware samples. At first, their strategy utilizes a local information-entropy-driven image generation technique to develop efficient image feature vectors. Following that, they improve the functionality of the lightweight neural network model ESPNetV2 by concentrating on four critical factors. They introduce a novel adversarial malware generation framework for Android applications known as Mal-WGANGP, which independently generates a substantial volume of adversarial samples to strengthen the resilience of their model.

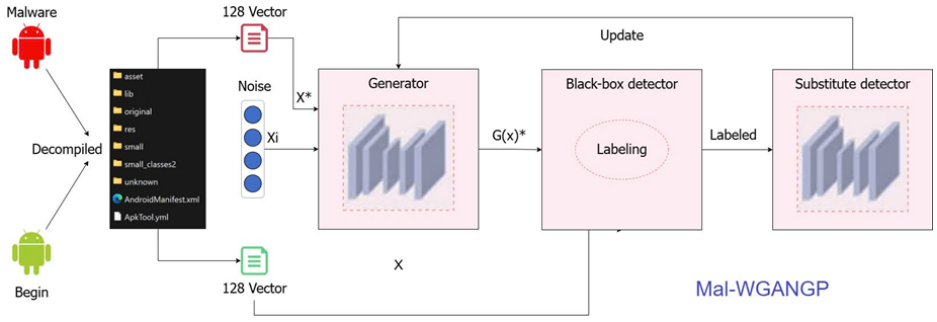

Figure 5: The framework of Mal-WGANGP

In their proposal [25], the authors introduce a highly effective approach for identifying adversarial malwares. They begin by creating images of Android applications utilizing a local-information-entropy-based technique. Afterwards, they refine the ESPNetV2 algorithm to establish a framework for detecting adversarial malware. By employing the MAL-WGANGP method, they produce many adversarial samples while maintaining the functionality of the malware, which are incorporated into their experimental dataset for evaluation. They construct different test datasets and assess 20 neural network detection models. Their findings validate the efficacy of their method. The static detection technique they implement is cost-effective and provides rapid detection speeds, making it well-suited for large-scale batch applications, unlike dynamic detection methods. In the structure of Syndroid in Figure 6, the gathered samples for classifying Android malware often experience considerable class imbalance, with ratios that can exceed 100:1. To tackle this issue, they present SynDroid, a classification model that leverages CTGANSVM to produce high-dimensional samples while filtering out subpar results. They propose the KS-CIR test to determine which classes need the most improvement, evaluating both the quality and quantity of the samples. Ultimately, a Random Forest is employed as the classifier. When assessed on the CCCS-CIC-AndMal2020 dataset, SynDroid surpasses conventional methods, achieving 12% greater accuracy and alleviating imbalance concerns [26].

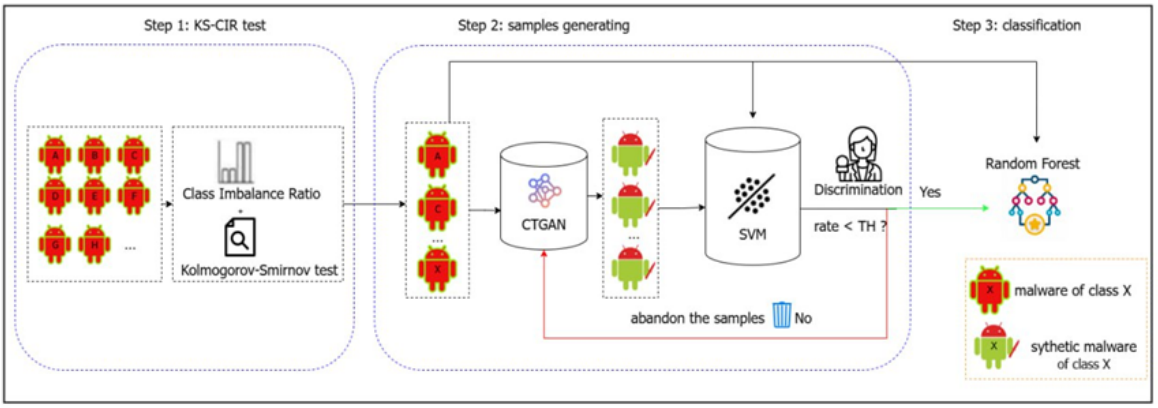

Figure 6: The framework of SynDroid

Actually SynDroid is a method for classifying Android malware that uses GAN to expand the dataset. To identify classes needing improvement, they present the KS-CIR test,

which combines the K-S test with the Class Imbalance Ratio, addressing both the quality and quantity of samples. While SynDroid improves classification outcomes and shows promise at CCCS-CIC-AndMal2020, it does not fully resolve dataset imbalance and incurs higher processing times. Although using SVM has reduced poor results, intrinsic GAN issues remain. Future work aims to refine the method and test additional datasets for broader application in Android malware detection. Additionally, they seek to develop a metric to assess overall dataset quality, focusing on error rates and missing values, making malware data enhancement more systematic and quantifiable [26].

As we early pointed out in the previous section, malware visualization involves converting malware binaries into grayscale or color images to identify distinct features for classification. This approach proves to be more efficient than conventional machine learning techniques, which necessitate significant expertise and lengthy analyses. In [23], the authors present a GAN architecture named 'MIGAN' that rapidly generates high-quality synthetic malware images and categorizes them into families. The framework consists of a generator, discriminator, and classification module, alongside innovations in structure, loss functions, and datasets. Images generated by MIGAN attain a superior Inception Score (2.81 compared to 1.90) relative to original malware images and show enhanced Fréchet Inception Distance and Kernel Inception Distance scores.

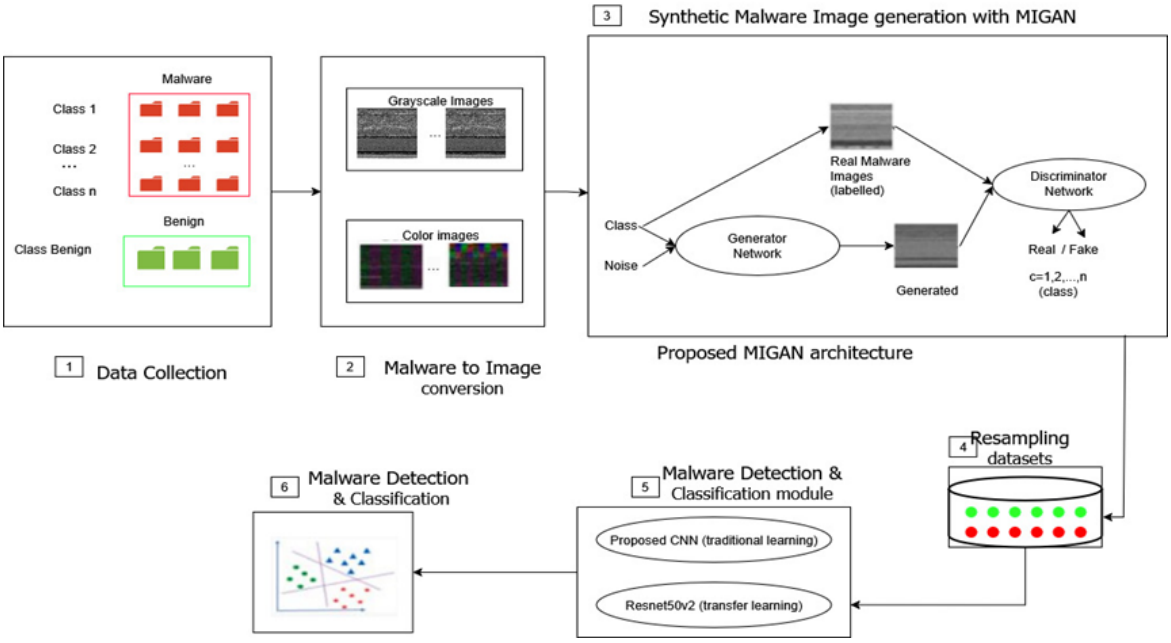

Figure 7: Components of the proposed MIGAN framework

As depicted in Figure 7, MIGAN addresses the problem of class imbalance in both self-generated and publicly available malware image datasets. The images produced closely mimic existing malware, proving beneficial for the identification of potential "zero-day" samples. The model's robustness improves through training on a variety of samples. The malware visualization technique assists in recognizing patterns and categorizing samples into different families, utilizing both traditional and transfer learning approaches. The framework's

performance is assessed using a custom malware image dataset along with the public "Malimg" benchmark, demonstrating enhanced accuracy and F1 scores compared to other leading methods. The authors showcase a combination of GAN-based and malware visualization-based techniques for the generation, identification, and classification of Windows malware images. Nonetheless, this approach has the potential for further improvement in the future to assess the classification model's resilience against adversarial attacks. By incorporating bytes from benign executable into malware files, this experiment will alter the statistical distribution of malware samples.

In [27], features are extracted from malware executable files and depicted as images through various methods. The authors experimented with four techniques for extracting images from malware executable: grayscale, colormap, three-gram, and PE file. The work then concentrates on a GAN for multiclass classification and assess our GAN outcomes against other well-known machine learning methods, such as support vector machine (SVM), XGBoost, and restricted Boltzmann machines (RBM). Its findings suggest that the ACGAN discriminator (see Figure 8) is generally on par with other machine-learning methods. In ACGAN, the discriminator outputs both a real/fake decision and a predicted class label c of the input image through Q, the auxiliary classifier. Additionally, the authors explore the effectiveness of the GAN generative model for conducting adversarial attacks on image-based malware detection.

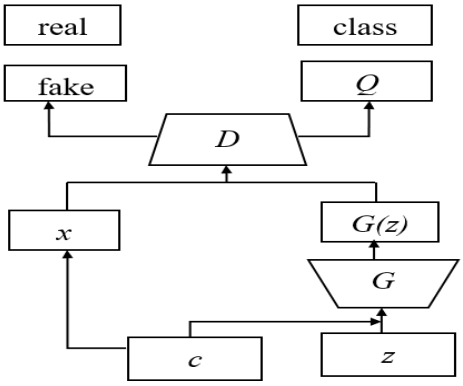

Figure 8: The ACGAN architecture

Secondly, other application of GANs in the field of malware analysis involves the generation of malware, which aids in evaluating the detection effectiveness of machine learning-based malware detectors. The capability to produce new instances of malware families for training machine learning detection systems regarding the characteristics of a malware family represents significant progress in defensive technology. For instance, deep learning GAN models are utilized to create unseen malware samples and train detection schemes against the signatures of these novel malware instances.

Several studies showcase different forms of malware created using GANs, which are commonly known as adversarial malware samples. These samples are designed to improve the resilience of machine learning-based malware detectors. Considerable research has been conducted on adversarial generation for both Windows and Android platforms. This section focuses on generating adversarial examples that can bypass malware detection systems by applying slight modifications to existing malware files. The subsequent sections will provide

Table 4: Summary of malware detection studies using GANs

Study	Method	Dataset	GAN Model Used
Sonam Bhardwaj et al. 2024 [24]	MD-ADA a malware detection framework	Windows PE	MD-ADA employs the GAN model for malware detection
Caixia Gao et al. 2024 [25]	Enhanced lightweight neural network	Android	Mal-WGANGP
Junhao Li et al. 2024 [26]	An Adaptive Enhanced Android Malware Classification Method	Android	CTGAN-SVM
Osho Sharma et al. 2024 [23]	GAN for facilitating malware image synthesis	Windows PE	MIGAN
Nguyen et al. (2023) [27]	Malware classification based on images	Windows PE	ACGAN

a brief overview of various adversarial generation studies conducted by researchers in the field of malware analysis. The adversarial research has been categorized according to the attack domain, which includes Windows and Android.

Much of the current research on adversarial malware generation focuses on utilizing gradient information and manually designed rules. However, the limited representational power of existing gradient-based models has made achieving a high true positive rate (TPR) difficult. GANs have successfully facilitated black box attacks on malware detectors with notably high TPR. The typical GAN architecture employed for executing adversarial malware attacks is illustrated in Figure 9. A summary of adversarial attacks targeting anti-malware engines is given in Table 4.

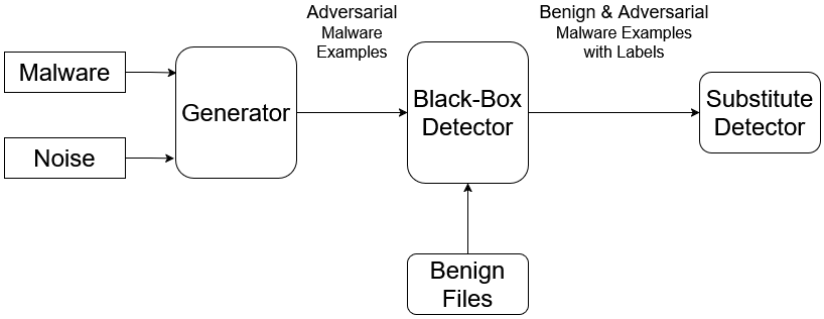

Figure 9: An adversarial malware generation architecture using GAN

Among those proposed, Hu et al. [22] introduced a technique for generating adversarial examples called MalGAN, which successfully evades black-box machine learning models. Unlike other GAN architectures, MalGAN consists of a generator and a substitute detector, both designed as feed-forward neural networks. The model takes binary features indicating the presence or absence of APIs as its input, with the number of input features corresponding to the input's dimensionality. The generator's role is to convert malware into an adversarial variant by generating a probability distribution of adversarial examples that are distant from the detector. By merging malware feature vectors with noise vectors, the generator can create multiple adversarial examples from a single malware feature vector. A substitute detector is utilized to adapt the detector model and supply gradient information necessary for training the generator. MalGAN is developed using 160 system-level Application Programming

Interfaces (APIs) focused on various machine-learning detectors. Separate experiments were carried out using both MalGAN and detector models while either sharing or dividing the training dataset. A true positive rate of zero has been recorded on the majority of machine learning models, indicating the high accuracy with which the substitute detector can fit. MalGAN's capability to execute complex transformations has led to a zero TPR for both its training and testing datasets.

Given the influence of using multiple malware to train MalGAN on its evasion performance, Kawai et al. [28] suggested an enhanced version of MalGAN that utilizes only one malware for its training process. MalGAN takes in malware detectors for both training and prediction, which is cumbersome for attackers. This enhanced version of MalGAN employs Python's subprocess library to solely import detection results into MalGAN. The authors opted to utilize all the APIs associated with malware to feature a larger quantity as opposed to the 128 APIs that were used in the original MalGAN. A different dataset was employed to create API lists for training both the detectors and MalGAN. These API lists were compiled by merging numerous cleanware and individual malware samples to prevent the malware detection process from being influenced by the inclusion of cleanware features in the malware files.

The generator and substitute model were also modified to use Deep convolutional GAN (DCGAN), as initially introduced by Radford et al. [18] in the context of image generation. In the malware domain, the activation function was switched to the Parametric Rectified Linear Unit (PReLU) due to its capacity to self-learn the negative component of the Leaky ReLU function. MalGAN improves its performance by incorporating cleanware features into the original malware file, thus avoiding detection by malware detectors. However, the assumptions underlying the design of MalGAN are somewhat unrealistic and restrict its ability to evade genuine malware classifiers. One such assumption is that it is presumed that malicious users have complete access to the feature space within the detector model. Furthermore, API features are viewed as an overly extensive means of representing malware.

To address these shortcomings, Castro et al. [29] released a poster advocating for the use of a GAN-based approach to create adversarial examples through byte-level perturbations. The suggested model operates with actual PE files rather than relying on API feature representations. By combining automatic byte-level real perturbation with feature representation, adversarial examples are generated. The generator processes a vector of 2,350 features, delivering a comprehensive outline of each piece of malware, and produces a random sequence of perturbations with nine different options for each injection. The integration of a more detailed feature representation and the capability of generating valid PE binaries enables the system to bypass not just the GBDT detector, but also to cross-evade various classifiers. Relying on API sequences or feature representations entails considerable manual effort to obtain the training data.

As listed in Table 5, which presents a summary of adversarial attacks targeting anti-malware engines, Shahpasand et al. [8] created adversarial data with GAN by maintaining threshold on the distortion values of generated samples see Figure 10. The existing malware data is modified by adding the generated optimum perturbation δ to produce adversarial one. Like every other GAN architecture, the generator can learn the distribution of benign samples, generating perturbations that can bypass the learning-based detectors. The adversarial samples are identifiable with benign files by the discriminator implicitly improves the

Table 5: Windows adversarial malware generation architecture using GAN

Paper/ Year	Key - Motivation	Target Model	Byte/ Feature	Approach	Feature Count
Hu. et al. 2017 [22]	The importance of developing a flexible black box adversarial attack mechanism.	ML-based (RF, LR, DT, SVM, MLP, VOTE) detectors	Feature	- Feed Forward Neural Networks are utilized for both the generator and the substitute detector. - An iterative method involves altering one feature with each iteration.	128 APIs
Kawai et al. 2019 [28]	Using single malware for realistic attacks	ML-based (RF, LR, DT, SVM, MLP, VOTE) detectors	Feature	- A Deep Convolutional GAN is used for the Substitutor (S) and Generator (G). - An API list is compiled from multiple cleanware and a single malware.	All APIs
Castro et al. 2019 [29]	Automatic byte-level modifications	GBDT Model	Byte Level	- Enhanced Feature Representation - Produces a random perturbation sequence utilizing nine distinct options at each injection point.	2350 Features
Yuan et al. 2020 [30]	Using generating perturbations that can bypass the learning-based detectors	MalConv	Byte Level	- Dynamic thresholding is employed to enhance payload effectiveness. - An automatic weight tuning mechanism balances the attention of the generator between payloads and adversarial samples.	Raw Bytes
Shahpasand et al. 2019 [8]	End-to-End blackbox attacks at byte levels	SVM, Neural network, RF and LR	Features	- GAN architecture with a threshold on generated distortion -Different loss functions to generate benign like adversarial and to produce high misclassification	128 APIs
Nam et al. 2024 [31]	Using a flexible black box adversarial attack mechanism	ML-based (RF, LR, DT, SVM, MLP, VOTE ,AB, GB, KNN) detectors	Features	- Double detectors and a least squares loss function.	160 APIs

perturbation with escalating the loss of generator.

The authors conducted their study using the Drebin dataset, which is widely recognized for its application in malware detection. They focused on identifying a subset of features that exert the most considerable influence on the classification process. This careful selection was aimed at ensuring that the performance of the classifier would remain comparable to that achieved with the full feature set, thereby optimizing efficiency without sacrificing accuracy. The results showed that while the classifiers maintained a high level of accuracy on the original samples, the adversarial data generated by the proposed model posed a significant threat. In fact, this model was capable of successfully compromising the classifiers with a

remarkable success rate of up to 99%. This finding underscores the vulnerabilities present in current classification systems and highlights the need for ongoing research to enhance their robustness against adversarial attacks.

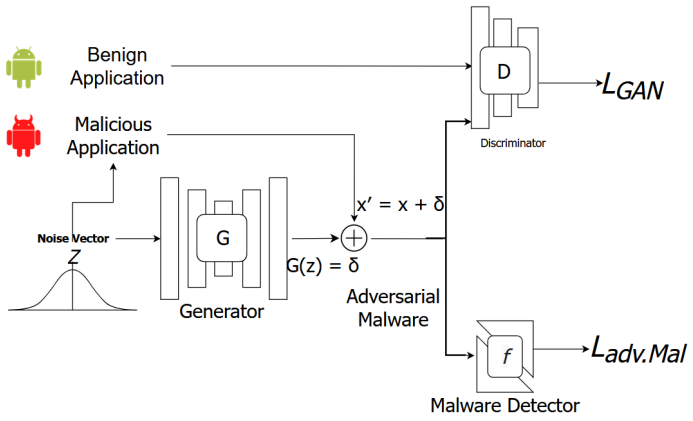

Figure 10: The architecture for generating adversarial malware

The authors in [31] proposed generating adversarial malware samples using GANs to enhance the robustness of malware detectors. The new model, Mal-D2GAN, as illustrated in Figure 11, consists of two main components: the generator and the detectors. Unlike the original MalGAN model, the Mal-D2GAN incorporates an additional detector block, which works with the substitute detector to improve its ability to identify malware generated by the generator block. They further evaluated this model using the new ACTMalv2 dataset and compared the results with those obtained from the ACTMalv1 dataset.

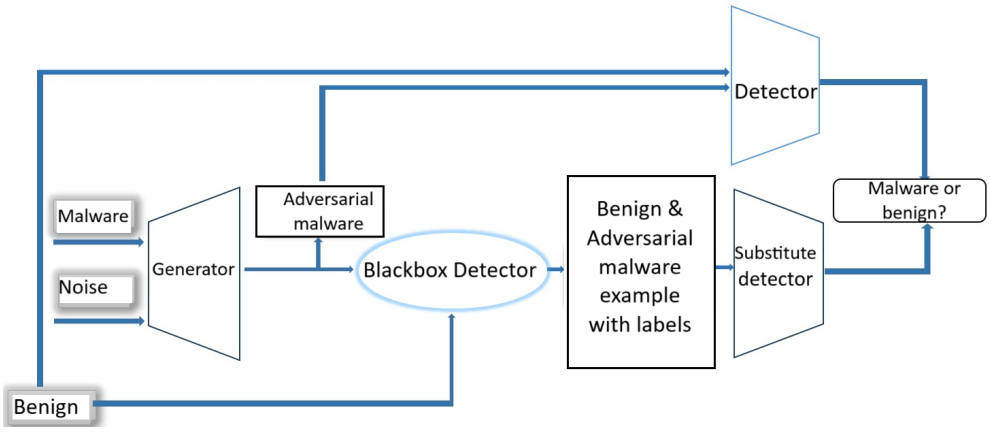

Figure 11: The architecture of Mal-D2GAN

The results presented in Table 6 indicate that the Mal-D2GAN model has reduced the detection accuracy (true positive rate, TPR) in eight malware detection scenarios across both datasets. Despite this reduction, the Mal-D2GAN model outperforms existing malware detection benchmarks regarding high-performance metrics. When evaluated across various algorithms, including Random Forest (RF), Logistic Regression (LR), Decision Tree (DT), Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), AdaBoost (AB), Gradient Boosting (GB), and k-Nearest Neighbors (KNN), the true positive rate usually drops to nearly 0%.

Table 6: The model's true positive rate (in percentage) on the training set. "Adver." represents adversarial samples.

Model	Mal-D2GAN		Mal-D2GAN	
Dataset	ACTMalv1		ACTMalv2	
	Original	Adver	Original	Adver
RF	97.25	2.28	100	0
LR	94.69	0	100	0
DT	97.35	0	100	0
SVM	96.10	0.07	100	0.75
MLP	98.31	0.16	100	0
AB	93.66	0	100	1.1
GB	97.35	0.37	100	4.5
KNN	98.10	0.15	99.9	0

These experimental results highlight the significant impact of an additional detector within the Mal-D2GAN model, which leads to a substantial decrease in detection accuracy across the eight malware detection methods.

5. CHALLENGES AND FUTURE DIRECTIONS OF GAN IN MALWARE ANALYSIS

5.1. Challenges

In this section, we examine the key challenges when applying generative adversarial networks (GAN) to malware analysis. These insights are crucial for developing more effective and reliable models in the field [32].

One of the most significant problems is limited and imbalanced datasets, which can severely hinder the training phase and performance of GAN-based models. In malware detection, particularly PE files, datasets often have uneven class distributions and a scarcity of labeled samples, thus not only degrading the accuracy of the model but also posing difficulties in generating synthetic samples. GANs must be carefully trained to avoid reinforcing imbalances [33], and several studies have highlighted these limitations. The authors in [9] show that GANs struggle with adversarial learning in botnet datasets characterized by high levels of imbalance. Also, pre-processing plays a critical role in achieving the good performance for GAN; poor data quality or inadequate feature representation can reduce the value of synthetic samples [34].

From a technical aspect, computational complexity is a major constraint because training GAN models requires significant time and processing power. In addition, GANs can alleviate data scarcity by producing synthetic malware variants, but their real-world deployment demands careful consideration of available resources. Furthermore, ensuring that these models remain robust against adversarial attacks requires extensive fine-tuning and validation [35]. Similarly, the research in [36] emphasizes the challenges posed by limited training data and underscores the reliance on domain expertise for the extraction of hand-crafted features, an approach that may not be accessible to all researchers.

In practice, detecting zero-day malware remains particularly difficult due to its previously unseen patterns and behaviors, leading to the lack of comprehensive, standardized samples to support robust model evaluation [37]. The GAN-based toolkits can address the shortage

of evaluation frameworks for malware detection methods with benchmarks. They assist in performance assessment and craft both novel attack and defense strategies. Drawing from successful practices in other domains, a collaborative and standardized approach integrating GANs into evaluation pipelines can also benefit Windows PE malware detection.

5.2. Future directions

Generative Adversarial Networks (GANs) have demonstrated a pivotal role in creating synthetic data by effectively modeling inherent variability. This variability directly influences the quality, diversity, and reliability of the generated samples. Future research directions should focus on leveraging GANs to address the challenges outlined above, with emphasis on:

(1) Data augmentation: Synthesizing malware samples across various categories in order to facilitate the construction of diverse datasets, which are essential for training and evaluating malware detection systems [32]. Specifically GANs can generate synthetic samples for minority classes, thereby addressing the problem of data imbalance in cybersecurity datasets. This approach helps create more balanced training datasets, allowing classifiers to detect underrepresented malware variants better.

(2) Model stability: In order to fully leverage the potential of GANs in cybersecurity, future research should also focus on training instability, mode collapse, and solving the problems associated with evaluating the quality of generated samples.

(3) Practicality: GANs can enhance feature representation by integrating visual characteristics with sequential behavioral data of malware to solve the limitation of feature extraction capability in malware detection. This multimodal feature construction not only improves the system's ability to differentiate malware types but also significantly enhances classification performance.

These existing issues will be addressed in order to develop robust, accurate, and adaptive malware detection systems that can effectively respond to increasingly sophisticated malicious threats.

6. CONCLUSION

This survey investigates in depth the significant roles of Generative Adversarial Networks in malware analysis. GANs based on deep learning, in particular, are a potential and practical approach for tackling the continuously changing challenges of malware analysis in the digital environment.

In this study, we examined GANs in malware analysis with their current challenges and limitations. Major issues such as data accuracy, imbalanced datasets, computational expenses, zero-day malware threats, and adversarial attacks must be addressed. Future studies should aim to improve the resilience of GAN models against malicious instances, overcome resource constraints, and improve data preparation techniques. Furthermore, ethical and responsible management is essential when applying GANs in cybersecurity to avoid disseminating propaganda and misinformation because GANs can create misleading media that confuse human and machine perceptions.

In summary, this survey provides a comprehensive overview of the existing research on GANs in malware analysis. It highlights both limitations and potential advantages of GANs

in malware analysis. If those problems, such as ethical issues and managing data responsibly, can be solved, GANs can be leveraged in malware analysis and digital environment protection from emerging malicious threats.

ACKNOWLEDGMENTS

This work has been supported by Academy of Cryptography Techniques.

REFERENCES

[1] SONICWALL. (2024) Sonicwall cyber threat report. Accessed on March 7, 2025. [Online]. Available: https://www.sonicwall.com/resources/white-papers/2024-sonicwall-cyber-threat-report

[2] B. K. Luis Corrons. (2023) Report, avast q4/2023 threat. Accessed on March 7, 2025. [Online]. Available: https://decoded.avast.io/threatresearch/avast-q4-2023-threat-report/

[3] J. V. Estenssoro. (2024) Malware and virus statistics 2024. Accessed on March 7, 2025. [Online]. Available: https://www.avg.com/en/signal/malware-statistics#notable-attack-examples

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Neural Information Processing Systems*, 2014.

[5] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CoRR*, vol. abs/1611.07004, 2016.

[6] Y. Li, M. R. Min, D. Shen, D. Carlson, and L. Carin, "Video generation from text," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[7] K. T. Chui, B. B. Gupta, P. Chaurasia, V. Arya, A. Almomani, and W. Alhalabi, "Three-stage data generation algorithm for multiclass network intrusion detection with highly imbalanced dataset," *International Journal of Intelligent Networks*, vol. 4, pp. 202–210, 2023.

[8] 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)*, 2019, pp. 17–20.

[9] R. H. Randhawa, N. Aslam, M. Alauthman, H. Rafiq, and F. Comeau, "Security hardening of botnet detectors using generative adversarial networks," *IEEE Access*, vol. 9, pp. 78 276–78 292, 2021.

[10] K. Baker. The 12 most common types of malware. Accessed on March 7, 2025. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/malware/types-of-malware/

[11] M. Sikorski and A. Honig, *Practical Malware Analysis - The Hands-On Guide to Dissecting Malicious Software*, ser. AMS Chelsea Publishing Series. Chelsea Publishing Company, 2012.

[12] K. Aryal, M. Gupta, M. Abdelsalam, P. Kunwar, and B. Thuraisingham, "A survey on adversarial attacks for malware analysis," *IEEE Access*, vol. 13, pp. 428–459, 2025.

[13] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 3, 2018.

[14] Cuckoo sandbox. Accessed 7-3-2025. [Online]. Available: https://cuckoo.readthedocs.io/en/latest/introduction/what/

[15] E. Bertino, S. Bhardwaj, F. C. S. Gong, I. Karim, C. Katsis, H. Lee, A. S. Li, and A. Y. Mahgoub, *Machine Learning Techniques for Cybersecurity*. Springer, 2023.

[16] P. Beaucamps, "Advanced polymorphic techniques," *International Journal of Computer Science and Information Engineering*, vol. 2, no. 3, pp. 194–205, 2007.

[17] Y. Wang, Q. Zhang, G.-G. Wang, and H. Cheng, "The application of evolutionary computation in generative adversarial networks (GANs): a systematic literature survey," *Artificial Intelligence Review*, vol. 57, no. 182, 2024.

[18] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[19] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014. [Online]. Available: https://arxiv.org/abs/1411.1784

[20] M. Zhu, S. Gong, Z. Qian, and L. Zhang, "A brief review on cycle generative adversarial networks," *Proceedings of The 7th International Conference on Intelligent Systems and Image Processing 2019*.

[21] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2813–2821.

[22] H. Weiwei and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," in *Proceedings of the Data Mining and Big Data: 7th International Conference, DMBD 2022, Beijing, China, November 21–24, 2022*, vol. Part II. Singapore: Springer Nature, 2023, pp. 409–423.

[23] O. Sharma, A. Sharma, and A. Kalia, "MIGAN: GAN for facilitating malware image synthesis with improved malware classification on novel dataset," *Expert Systems with Applications*, vol. 241, p. 122678, 2024.

[24] S. Bhardwaj, A. S. Li, M. Dave, and E. Bertino, "Overcoming the lack of labeled data: Training malware detection models using adversarial domain adaptation," *Computers & Security*, vol. 140, p. 103769, 2024.

[25] C. Gao, Y. Du, F. Ma, Q. Lan, J. Chen, and J. Wu, "A new adversarial malware detection method based on enhanced lightweight neural network," *Computers & Security*, vol. 147, p. 104078, 2024.

[26] J. Li, J. He, W. Li, W. Fang, G. Yang, and T. Li, "Syndroid: An adaptive enhanced android malware classification method based on CTGAN-SVM," *Computers & Security*, vol. 137, p. 103604, 2023.

[27] H. Nguyen, F. Di Troia, G. Ishigaki, and M. Stamp, "Generative adversarial networks and image-based malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 19, no. 4, pp. 579–595, 2023.

[28] M. Kawai, K. Ota, and M. Dong, "Improved MalGAN: Avoiding malware detector by learning cleanware features," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2019, pp. 040–045.

[29] R. L. Castro, C. Schmitt, and G. D. Rodosek, "Poster: Training GANs to generate adversarial examples against malware classification," in *IEEE Symposium on Security and Privacy*, 2019.

[30] J. Yuan, S. Zhou, L. Lin, F. Wang, and J. Cui, "Black-box adversarial attacks against deep learning based malware binaries detection with GAN," in *European Conference on Artificial Intelligence*, 2020.

[31] N. T. Hoang, T. D. Pham, and L. T. Bui, "Mal-D2GAN: Double-detector based GAN for malware generation," *16th IEEE International Conference on Knowledge and Systems Engineering*, 2024, Accepted to appear. [Online]. Available: https://arxiv.org/abs/2505.18806

[32] M. M. Arifin, M. S. Ahmed, T. K. Ghosh, I. A. Udoy, J. Zhuang, and J. haw Yeh, "A survey on the application of generative adversarial networks in cybersecurity: Prospective, direction and open research scopes," 2024. [Online]. Available: https://arxiv.org/abs/2407.08839

[33] F. Demirkıran, A. Çayır, U. Ünal, and H. Dağ, "An ensemble of pre-trained transformer models for imbalanced multiclass malware classification," *Computers & Security*, vol. 121, p. 102846, 2022.

[34] D. Park, H. Khan, and B. Yener, "Generation & evaluation of adversarial examples for malware obfuscation," *2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1283–1290, 2019.

[35] A. Dunmore, J. Jang-Jaccard, F. Sabrina, and J. Kwak, "Generative adversarial networks for malware detection: a survey," 2023. [Online]. Available: https://arxiv.org/abs/2302.08558

[36] F. Mazaed Alotaibi and Fawad, "A multifaceted deep generative adversarial networks model for mobile malware detection," *Applied Sciences*, vol. 12, no. 19, 2022.

[37] D.-O. Won, Y.-N. Jang, and S.-W. Lee, "Plausmal-GAN: Plausible malware training based on generative adversarial networks for analogous zero-day malware detection," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 82–94, 2023.

Received on February 08, 2025
Accepted on May 07, 2025