IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 20 December 2023, accepted 14 January 2024, date of publication 25 January 2024, date of current version 26 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3358454

RESEARCH ARTICLE

Design and Performance Analysis of an Anti-Malware System Based on Generative Adversarial Network Framework

FAIZA BABAR KHAN[1], MUHAMMAD HANIF DURAD[1], ASIFULLAH KHAN[2,3], FARRUKH ASLAM KHAN[4], (Senior Member, IEEE), MUHAMMAD RIZWAN[1], AND AFTAB ALI[5]

[1]CIPMA Laboratory, Department of Computer and Information Sciences (DCIS), Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad 45650, Pakistan
[2]Pattern Recognition Laboratory, Department of Computer and Information Sciences (DCIS), Pakistan Institute of Engineering and Applied Sciences (PIEAS), Nilore, Islamabad 45650, Pakistan
[3]PIEAS Artificial Intelligence Center (PAIC), Pakistan Institute of Engineering and Applied Sciences (PIEAS), Nilore, Islamabad 45650, Pakistan
[4]Center of Excellence in Information Assurance, King Saud University, Riyadh 11653, Saudi Arabia
[5]School of Computing, Ulster University, BT15 1ED Belfast, U.K.

Corresponding author: Faiza Babar Khan (faiza_19@pieas.edu.pk)

This work was supported in part by the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia under Grant IFKSUOR3-507-2; in part by the Information Technology & Telecom (IT&T) Endowment Fund Pakistan Institute of Engineering and Applied Sciences (PIEAS), Higher Education Commission of Pakistan, under Grant PIN 315-7318-2EG3-116; and in part by the National Center in Cyber Security, Pakistan, through the Critical Infrastructure Protection and Malware Analysis Laboratory, PIEAS.

ABSTRACT The cyber realm is overwhelmed with dynamic malware that promptly penetrates all defense mechanisms, operates unapprehended to the user, and covertly causes damage to sensitive data. The current generation of cyber users is being victimized by the interpolation of malware each day due to the pervasive progression of Internet connectivity. Malware is dispersed to infiltrate the security, privacy, and integrity of the system. Conventional malware detection systems do not have the potential to detect novel malware without the accessibility of their signatures, which gives rise to a high False Negative Rate (FNR). Previously, there were numerous attempts to address the issue of malware detection, but none of them effectively combined the capabilities of signature-based and machine learning-based detection engines. To address this issue, we have developed an integrated Anti-Malware System (AMS) architecture that incorporates both conventional signature-based detection and AI-based detection modules. Our approach employs a Generative Adversarial Network (GAN) based Malware Classifier Optimizer (MCOGAN) framework, which can optimize a malware classifier. This framework utilizes GANs to generate fabricated benign files that can be used to train external discriminators for optimization purposes. We describe our proposed framework and anti-malware system in detail to provide a better understanding of how a malware detection system works. We evaluate our approach using the Figshare dataset and state-of-the-art models as discriminators. Our results showcase enhanced malware detection performance, yielding a 10% performance boost, thus affirming the efficacy of our approach compared to existing models.

INDEX TERMS Anti-malware system, generative adversarial networks, malware sandboxes, malware, unpacker, performance.

I. INTRODUCTION

With the latest advancements in computing technology, one of the biggest assets of today's era is data. The data is

The associate editor coordinating the review of this manuscript and approving it for publication was Zesong Fei.

so important that it has effectively changed the world and how we do various tasks. This gives rise to the need to secure the data and its transmission over communication channels [1]. Cybersecurity seems to be a great concern for the functioning of institutions like banks, government and private organizations, and businesses, where a considerable amount of sensitive information in the form of data is continuously generated. Owing to the rising volume of malware [2], multiple challenges are being faced, but the foremost challenge is to protect the data from the damage of malware. Malware developers have become so smart that they find ways to hide their malware before introducing it to the world [3]. Thus, an efficient and error-free system is much needed to stop malware-assisted attacks even before the availability of malware signatures. It is observed that even after 30 years, the same techniques are also being used for inducing malware, which is mainly due to the outdated anti-malware solutions. Injecting malware is easier than detecting that malware. The conventional Malware Analysis System (MAS) mostly works on signature-based detection. Most of the previous work done for the design of anti-malware solutions focuses on carrying out signature matching [4], i.e., it generally acts on the principle of signature-based detection, which works on the availability of malware signatures. Therefore, it cannot be an efficient detection system for zero-day attacks [5]. Modern malware analysis and detection systems must not only be based on static features, heuristics matching, and emulation techniques, but with the onset of machine learning, it can also be enriched to detect novel attacks [6]. Anti-malware products perform malware detection based on signatures of malware, so the signature repository needs to be checked for updates periodically to keep the anti-malware product useful. However, the same detection methodology does not apply to detect zero-day attacks or even existing malware variants. Therefore, it has become vital to put in efforts to detect malware to stop the attacks and chaos created by the malware, especially when it appears for the very first time. A complete Anti-Malware System (AMS) should be equipped with static and dynamic analysis modules. The static analysis focuses on detecting the nature of the file without execution, whereas for dynamic analysis, firstly the file is executed and then inspected against its run-time behavior [7]. A hybrid strategy is a combination of both static and dynamic analysis. Moreover, the capability of an AMS can be enhanced by adding components furnished by various Artificial Intelligence (AI) techniques [8], [9], [10].

Since we focus on an optimum solution, we utilize generative modeling for domain-specific data generation and augmentation for continuous training and learning of the AI model employed by the AMS. For this purpose, we use the algorithmic architectures of Generative Adversarial Networks (GANs) [11] that use two neural networks, opposing each other for the generation of new, fabricated instances of data that can pass for actual data. These two neural networks are: the generator, which is used for the generation of new data, and the discriminator, which is used for the evaluation of genuineness. The job of the discriminator is to differentiate between the real and fake instances, whereas the generator generates synthetic images in the hope that these fake instances too will be deemed authentic, even though they are fake.

Keeping in view the perspective of GANs, a signature-based malware detector is integrated with a GAN-based framework with the capability of generating samples for enhancing the existing architecture of the malware discriminator. Most techniques of malware code detection suffer due to the smartness of the attacker. As we go through previous research conducted in the area of malware investigation and detection, a big gap is sighted between the knowledge of a malware developer and the malware analyst or anti-malware software developer. There is a dire need to bring about an optimum architecture for enhancing the robustness of the existing malware detectors.

TABLE 1. Comparison with existing work.

Features	Previous Work	This paper
Focus on Signature-based malware detection	[12]–[15]	✓
Machine Learning based Framework	[16]–[20]	✓
Comparisons with existing approaches	[18], [21]–[23]	✓
Focus on single issue encountered by AMS	[18], [24]–[26], [26]–[30]	✓
Detailed architecture of conventional AMS	X	✓

To do research in the field of malware detection, the sole inspiration has always been to bring forward the best possible capabilities of discovering the malware in an economical and real-time manner so that malware risks can be alleviated before a successful attack is launched. For this purpose, in this paper, we propose the architecture of an efficient AMS that is equipped with a Generative Adversarial Network (GAN) based Malware Classifier Optimizer (MCOGAN) framework. We present a two-fold solution with the perspective of not only elaborating on the whole concept of the structure of an AMS but also presenting a framework for enhancing the sturdiness and robustness of the Machine Learning (ML) based models. Different levels of an AMS architecture have been unfolded so that even a novice may understand the workings of an AMS. In the existing literature, we have not seen such a detailed architecture of an AMS. The whole process is described meticulously, i.e., starting from getting a suspicious file, passing through detection of malware using a signature database, and then ending up with anomaly-based detection with an assurance to enhance the strength of the utilized ML models. In the proposed MCOGAN framework, we investigate the use of GAN-based data augmentation to artificially enlarge training instances for model tuning. We worked on the idea that a GAN is not only

helpful for data-deficient scenarios but also works wonders for the training of any existing AI model by continuously generating fake examples and enhancing the robustness of the model. MCOGAN is comprised of two sub-modules, i.e., data generator and classifier optimizer. These two modules complement each other in bringing up the optimal malware classifier. Our proposed work will serve to achieve the same purpose. Table 1 provides a comparison of our proposed work with the work done previously on malware detection. The details will be discussed in Section IV. Table 2 shows the nomenclature explaining the acronyms and abbreviations used in the paper.

The primary contributions of this research are as follows:

1) A detailed description of the problems encountered by the AMS is presented. Moreover, year-wise malware statistics are presented graphically.
2) A comprehensive signature-based malware detection architecture is presented and its components are described in detail.
3) A GAN-based anti-malware framework is proposed and evaluated for optimizing machine learning-based malware classifiers.
4) Potential study directions in this area are discussed.

The remainder of the paper is organized as follows: Section II presents the previous work done for malware detection and anti-malware architectures. Section III describes the details of the problems encountered by an anti-malware system. Section IV presents the related theory and background. In Section V, the proposed architecture is presented with details of every component. Section VI discusses the proposed GAN-based framework for our AMS architecture. In Section VII, the experiments are conducted for the proposed architecture, and the results are analyzed. Finally, Section VIII concludes the paper with upcoming research directions.

TABLE 2. List of acronyms.

Acronym	Definition
AI	Artificial Intelligence
AMS	Anti-Malware System
CNN	Convolutional Neural Network
CSG	Candidate Sample Generator
C&C	Command and Control
DGA	Domain Generation Algorithm
DB	Database
DLoss	Discriminator Loss
DNN	Deep Neural Network
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
GAN	Generative Adversarial Network
GLoss	Generator Loss
IDS	Intrusion Detection System
JIT	Just-In-Time
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
Malware	Malicious Software
MAS	Malware Analysis System
MCO	Malware Classifier Optimizer
ML	Machine Learning
Opcode	Operational Code
PE	Portable Executable
SVM	Support Vector Machine
TP	True Positive
TN	True Negative
UPX	Ultimate Packers for Executables
VAE	Variational AutoEncoder
WMI	Windows Management Instrumentation
YARA	Yet Another Recursive Acronym

II. RELATED WORK

In this paper, we propose the complete architecture of an AMS encompassing every aspect of analysis necessary for the eradication and detection of malware. This is somewhat different from earlier works, which tend to focus on one method to detect malware, such as identifying malicious software using the static or dynamic method. Since we present a signature-based malware detection method along with a framework to improve the efficacy of the malware classifier, in this section, we will go through the previous work in both dimensions.

A. CONVENTIONAL MALWARE DETECTION APPROACHES

Previous researchers have worked on different ways of presenting the architecture for an AMS. In [31], the authors discussed the setup designed for malware research along with network traffic dissection and execution of samples in a virtualized environment. However, their approach was dependent on the malware run timeouts. The authors in [12] proposed a heuristic-based malware detection approach aimed at Windows binary files that have been encoded and loaded into memory. Berger-Sabbatel et al. [13] described an architecture of a platform for reviewing botnets, finding sufficient analysis methods, and monitoring the activities of botnets only. The authors found efficient countermeasures or confinement methods. Santos et al. [14] proposed a way for the detection of variants of known malware families by using the occurrence frequency of opcode sequences. In [32], a framework is used by combining static, dynamic, signature-based, and behavior-based malware analysis. The approach is implemented using the API call graph system. The authors in [33] proposed detection methods only for metamorphic malware. They used API call sequences for extracting malware semantics. Firdausi et al. [16] performed malware detection by applying machine learning techniques, e.g., Naive Bayes, decision trees, support vector machine (SVM), and K-nearest neighbors (KNN) on API/system calls, file system, and Windows registry.

Previous researchers also explored machine learning and deep learning techniques for malware detection [34] but some improvements can be made in their work to devise an efficient anti-malware solution. In [35], the authors presented a malware detection approach using a rule-based classifier and Naïve Bayes by making use of strings and byte sequences. The authors in [15] and [17] used decision trees, Naïve Bayes, and SVM using byte sequences. In [36], the belief

propagation method is used. Similarly, other researchers also used SVM and KNN [16], [37] for malware detection. In [25], a semi-supervised learning approach is used to detect unknown malware.

In [38], the authors proposed ResNets-based architecture to detect the malware using behavior parameters, e.g., CPU, memory, and disk utilization. But as they did not address the issue of malware variants, FNR was high. In [39], the authors emphasized using API call sequence patterns of the malware. They described their results only on a specific dataset ignoring the fact that how well the model will be reusable for emerging malware. In [40], authors focused to work on behavior-based malware detection using API call sequencing. They used different algorithms to reach up to 97-98% accuracy. Although the work done by existing researchers provide a good addition to the malware detection field, we still lack a complete architecture for an AMS with all the integrated set of components.

B. GAN-BASED APPROACHES

GANs were first introduced by Ian Goodfellow and his team in 2014 [41]. GAN has been used for the synthesis of data based on training over some actual data. This feature is beneficial in a security environment where there is a lack of data for unseen attacks or malware. Consequently, many potential attacks, including formerly undetected attacks, are discovered using this augmented data, which usually remains unseen to the user in training [22]. GAN has also been used by researchers for malware detection. Anderson et al. [42] showed that by training data augmentation with adversarial examples generation, the classifier can detect additional malware families as compared to other methods. Chen and Jiang [43] illustrated the usability of GAN for the implementation of the Intrusion Detection System (IDS). They used GAN for understanding the normal data features. They proposed a GAN-based model with a better cost function. Burks et al. [44] performed a correlative study between Variational Autoencoder (VAE) and GANs. They presented improvement in malware detection using GANs. Zhao et al. [45] proposed a GAN-based framework to produce instinctive adversarial examples based on the data by searching the latent semantic space of thick and steady data representation. The authors in [46] and [47] also proposed a defense mechanism using generative adversarial examples generation. The authors in [44] proposed a defense mechanism using GANs and improved the existing performance of the attack defense mechanism. Yang et al. [48] proposed a method that utilizes adversarial example and attention models to generate facial images for de-identification purposes.

In [49], authors used GANs to amplify the malware using the Operational Code (Opcode). They amplified the Opcode part of a PE file and used it for malware detection. PE file was dissected for segregation of different parts, i.e., PE header, data, text, etc. They obtained 75% accuracy as they faced data loss due to dropping off the other parts of the suspected file. In [50], the researchers proposed the concept of only using the suspected file's header. They generated the new data by using header information but they succeeded to enhance the accuracy by only 1%.

III. ISSUES ENCOUNTERED BY AN ANTI-MALWARE SYSTEM

As malware developers are getting smarter day by day, more and more complications are encountered by malware analysts and detectors. Previous researchers have done a lot of work to address these issues and created awareness to understand the areas in which a system can be attacked (Table 3). Following is a brief description of these issues and challenges (Figure 1).

A. INCREASING THREAT VECTORS

A threat vector is a kind of pathway or methodology an attacker adopts for penetrating the target system and generating malicious outcomes, e.g., popup windows, deception, email attachments, etc. [51], [52]. Threat vectors may include phishing attacks, PDF malware, and social engineering attacks. Earlier researchers have emphasized the importance of understanding threat vectors and proposed attack taxonomies. The authors in [24] proposed a taxonomy of threat vectors in terms of five classifiers to illustrate the category of an attack using attack vector, defense, operational influence, informational repercussion, and by attack target. Ivaturi and Janczewski [53] also considered increasing threat vectors as a threat to malware analysis and proposed a taxonomy for social engineering attacks as attackers are using social engineering methods by aiming at the human elements and merging such techniques with conventional technical methods. They divided these attacks into two major categories of person-person and person-person over media. Pitropakis et al. [18] also studied the importance of threat vectors for approaching the solution of the problem involving the detection of machine learning attacks. The authors presented a taxonomy of attacks against systems using machine learning. The taxonomy identified the attack sharing key characteristics that can be addressed by the same defense tactics. In [27], the authors emphasize the attack types targeting data exfiltration, which can be committed by the outsider or insider of an organization. Since previous researchers put a lot of effort into understanding the reasons for increasing threat vectors, it can be considered one of the main problems encountered by an anti-malware system. Techniques like digital forensics and IP attribution can only clean up data breaches, but these techniques cannot prevent them. The increase in threat vectors is mainly due to the increase in recreational hackers with more time on their hands to work on their targets.

B. FAST FLUX NETWORKS

Attackers deploy exceptionally refined strategies to compromise consumers' systems and sustain complete control over

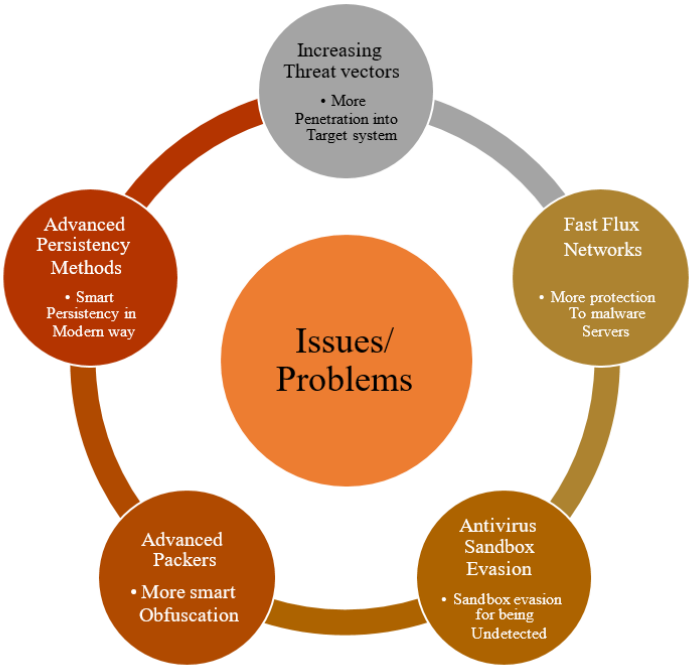


FIGURE 1. Problems encountered by anti-malware systems.

them for longer periods. Therefore, they use Domain Generation Algorithms (DGAs) considerably, not only for command and control (C&C) operations, but also for drive-by download and domain names generation for malware propagation. This stops their servers from being blacklisted or captured. These algorithms' subroutines provide malware with new domains on-demand or on-the-fly, e.g., 500,000 domains in a few lines of code. The domain names help to communicate with their C&C servers. This causes trouble for law enforcement organizations to effectively switch off the botnets as contaminated machines will try to interact with a few of the domain names daily to get updates or commands. The victim gets caught by the fake domains and the task of detection becomes critical.

Due to the perilous nature of DGAs, researchers focused their work on the nature and taxonomy of DGA and presented remedies for its abdication. Sood and Zeadally [26] proposed a taxonomy of DGA and focused on its impact caused to harm the C&C servers. The authors classified DGAs into binary and script-based DGAs and presented a deep overview of the characteristics of DGAs and their differences. Their work helped to come up with automated solutions for DGA detection with the help of ML techniques to offer more insight into the malware activities in the network. In [19], to mitigate the issue, the authors presented a machine learning-based approach for recognizing DGA domains. They used real-time threat data from live traffic collected for one year. Then, they classified DGA domains using a deep learning model. In [54], the authors suggested a framework for domain name generation prediction based on Long Short-Term Memory (LSTM) architecture.

C. ANTI-MALWARE SANDBOX EVASION

A sandbox is a tool or mechanism for malware detection that runs a distrustful object in a virtual machine (VM) with a fully-equipped operating system and senses the object's malicious activity by examining its actions. If the object performs malicious activities in a VM, the sandbox perceives it as malware. But sandboxes do not mean to always help with the detection of malware, rather they can misguide during this procedure of detection. Sandbox evading malware can sense the existence of a sandbox and avoids executing the malicious code until it gets out of this controlled environment. In [28], the author discussed various techniques used to evade the sandbox. Different anti-sandbox techniques look for the process name that may run on sandboxes, specific registry entries, module names used by sandboxes, virtualization software, and different artifacts to find the presence of sandbox and then go for evasion methods. The authors in [29] also focused on the limitations of sandboxes and discussed the Code Injection Method and RunPE method for the evasion of dynamic analysis used by modern malware. In [30], the author concentrated on a novel approach, called Just-In-Time (JIT) malware assembly that is capable of evading detection from network sandbox and conventional endpoint security solutions. Some of the network sandboxes search for executables in network traffic and then intend to copy, remove and ''explode'' them within a sandbox to determine

TABLE 3. Work done for addressing issues faced by malware analysis system.

Paper	Year	Focused Issue	Threat
[24]	2014	Proposed taxonomy of attack vectors using five key classifiers to characterize the nature of an attack.	Attack Vector
[53]	2011	Proposed taxonomy for social engineering attacks.	Attack Vector
[18]	2019	Proposed attack taxonomy and surveyed the attacks against machine learning.	Attack Vector
[27]	2018	Emphasized the attack vector types targeting data exfiltration.	Attack Vector
[26]	2016	Focused to propose a taxonomy of DGA and its impact caused on the C&C servers.	DGA
[19]	2019	A machine learning-based framework for recognizing and detecting DGA domains is presented for the alleviation of the threat.	DGA
[54]	2019	Real-time prediction of domains generated by DGAs is also considered as an issue, so Long Short-Term Memory (LSTM) based deep learning framework is proposed for addressing the same.	DGA
[28]	2015	Identified anti-sandbox techniques focused on the methodologies that malware adopts for the evasion of sandboxes.	Sandbox Evasion
[29]	2014	Discussed the limitation of sandboxes and explained Code Injection Method and RunPE method for evading dynamic analysis techniques used by modern malware.	Sandbox Evasion
[30]	2015	Focused on a novel approach, called “Just-In-Time (JIT) Malware Assembly" that is capable of evading detection from network sandbox and conventional endpoint security solutions.	Sandbox Evasion
[55]	2020	Employed Recurrent neural networks to categorize files that have been compressed using metamorphic packers.	Advanced Packers
[56]	2018	Comprehended the intricate technical aspects of fileless malware and the related assaults, and outlined numerous methods for identifying and minimizing their impact with a comprehensive analysis.	Advanced Persistence Methods
[21]	2020	Conducted an all-encompassing examination of fileless malware, and a series of protocols were put forward to address the issue in the event of an incident. Additionally, the difficulties linked to fileless malware were identified.	Advanced Persistence Methods

their potential for harm. But the JIT assembly technique being a big threat successfully evades security solutions.

D. ADVANCED PACKERS

With the progression in packers’ technologies, an overhead is being delivered during the process of malware detection. A malware packer is a mechanism or software developed to disguise a malware-containing file. Packers have the capability to encode, flatten, compact, or alter the layout of a malicious file to hide its identity entirely. This action of compaction or enciphering transforms the file from its real code to a new form using tested obfuscation techniques. Consequently, the malware remains to continue in a system unidentified by anti-malware software and other security solutions, hurting the integrity of the system. Packers can be metamorphic [55], which means that the output of the packer will never be precisely the same, even if the packing is applied again on the same file. In such a case, the detection of unknown packers is more needed without having any known samples of a given packer. To detect the packers in their system, it is recommended to have acquainted with some of the most used packers for malware obfuscation.

1) ULTIMATE PACKERS FOR EXECUTABLES (UPX)

UPX stands for “Ultimate Packer for Executables”. It is an open-source tool and does not use extra memory for unwinding. It makes the content of a malware binary file unreadable. This calls for some tools to unpack the data to get its meaning. So unpacker is one of the most important components of an AMS.

2) THE ENIGMA PROTECTOR

This is not only good for persons and businesses to protect files from hacking, but it can also pack and hide malware.

3) MPRESS

It was initially developed to compact files and minimize applications’ starting times. As this is free, it is also easily available to hacking applications and other malware coders.

4) MEW

MEW is used for the compression of smaller-sized malware files by making use of the LZMA algorithm. It can also be used to obfuscate larger malware files.

5) EXE PACKER 2.300

It is a free software and a famous packer for malware file obfuscation.

6) OBSIDIUM

Obsidium can be used for 32-bit and 64-bit Windows-based applications and performs encryption, compression, and obfuscation of malware.

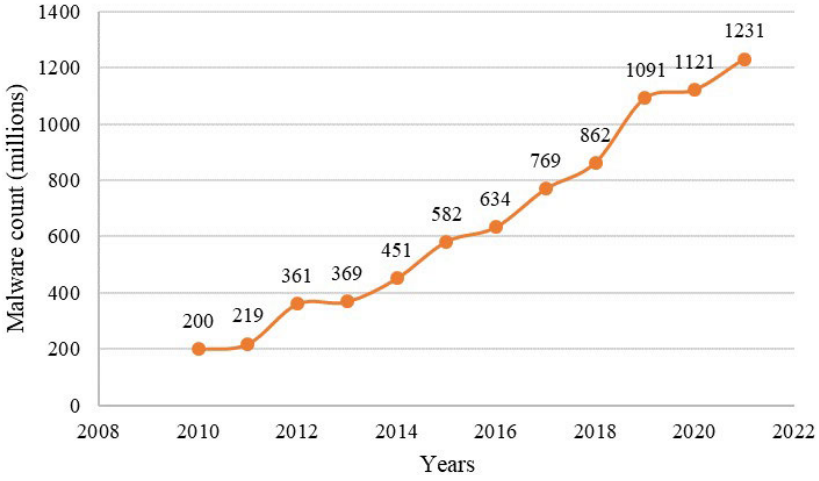


FIGURE 2. Malware statistics.

7) EXESTEALTH

ExeStealth is a free software to encode files to evade detection and hacking.

8) MORPHINE

It contains its private PE loader to let the users encrypt the output of compressed data. Its polymorphic engine is capable of creating distinctive malware cracks.

9) THEMIDA

It was originally developed by Oreans for the protection of Windows applications against hackers. But later on, it is being used to encode the malicious files.

10) ANDROMEDA

It is a custom packer and is not easy to reverse-engineer.

11) VMPROTECT

It is capable of encoding a variety of files, including drivers, executable files, and DLLs. As any application encoded by VMProtect is opened, the packer does not decipher any bit; instead, it runs on a virtual code.

E. ADVANCED PERSISTENCE METHODS

Persistence is the continuation of the malicious effects of malware, even after its removal from the system. Once malware gains access to the system, it tries to be there for a longer period. One of the most frequently achieved persistence methods is the modification of the system's registry. However, some other new methods are used by malware developers, which are more threatening to malware detectors.

1) FILE-LESS MALWARE

Over time, many different types of malware have progressed to be overlooked by anti-malware software. Fileless malware neither uses routine executables to perform its actions nor does it make use of the file management set-up, consequently eluding a signature-based discovery mechanism. The attack caused by fileless malware is more disastrous because of its persistence. Rather than writing artifacts to the disk, fileless malware is deliberately designed to be memory-resident only, leaving no traces after its execution. Windows Management Instrumentation (WMI) or PowerShell executes the provided payload or else completes the assigned tasks. Fileless malware is different, as it does not seek ways for malicious file installation like other malware programs; instead, fileless malware is trickier in its tool activation. Then it hides in the system. This malware remains ignored because it is memory-based, not file-based. It leaves no footprints for anti-malware products to detect. In [56], the technical specifics of fileless malware and related incidents were thoroughly addressed, and then detailed detection and mitigation approaches were offered. In [21], the authors surveyed an inclusive investigation of fileless malware and its detection methodologies that exist in the literature. Then, they presented a set of procedures to deal with the fileless malware attacks in case of any incident occurrence and identified the associated challenges.

2) STAGED MALWARE

As the name suggests, these malware types work in different stages. A malware payload can be stageless or staged depending on its action strategy. Staged malware payload works in different phases. Its main task is to get a successful entry into the system, maintain persistence, and then execute effectively without being detected. The payloads of staged malware break down the phases of an attack. Initially, a small payload code is set in the system, which exploits the target's vulnerability when it finds one [57].

Some staged malware, like web-based malware, are very dangerous as they perform in different stages, so they can be considered as one of the threats for malware detectors.

Malware is an umbrella word nowadays. It encompasses

1) Trojan
2) Cryptolocker
3) Rootkit/Bootkit
4) Potentially Unwanted Program (PUP), etc.

IV. RELATED THEORY AND BACKGROUND

In this section, the basic concepts of an AMS with different detection techniques are explained. GAN-based optimization for existing trained machine learning models is also discussed. The working principles of different types of GAN are also discussed to understand the concept of the optimization framework proposed for the AMS architecture.

A. SIGNATURE-BASED MALWARE DETECTION

As its name suggests, this method of detection works on the footprints or signatures of the existing malware [58]. Every program has a unique digital code that identifies that particular program, software, or application. An AMS scans the system to find out the malware signature, which is usually a sequence of bytes that categorizes the program as a malicious program [59]. For this type of detection, AMS needs to have a repository of predefined signatures for known malware. During the scanning of a system, AMS computes the signature of every file and compares them with the available signatures in its repository. Upon successful matching of the signatures, the file is declared malicious [60]. Although the False Positive Rate (FPR) of this type of detection is very less, the signature bypassing is possible with a slight change in the malware payload. Moreover, signature building is an ongoing process and the repository needs to be updated all the time [3].

B. MACHINE LEARNING-BASED MALWARE DETECTION

Machine learning methods enable computers to learn without being precisely programmed. Machine learning is a set of steps that discover the underlying patterns in the data provided and then predict the properties of unseen data [61]. Two types of approaches are used for machine learning [62]. In supervised learning, during training time, labels of the data are provided, which are used during the learning phase while reaching up to the optimal model that will yield the correct label Y for new objects when provided with the feature set X. For malware detection, X might indicate the features of a suspicious file including static and dynamic features [63]. In this scenario, label Y will be either malware or benign. For the training phase, a family of machine learning models is selected that can be decision trees, SVM, neural networks, etc [64]. Generally, parameters are the indicators to identify the model of a family. During this phase, the model is selected that produces most of the correct answers with a specific group of parameters. During testing, this model is applied to new instances with a fixed set of parameter values, and predictions are produced. In unsupervised learning, data without labels are provided and then the structure of data is explored. Clustering is one of the methods by which manual labeling of samples is optimized and it minimizes the efforts to arrange labeled data and can be helpful for the detection of new malware [65].

C. DEEP LEARNING-BASED MALWARE DETECTION

Deep learning is a type of machine learning approach through which useful high-level features can be extracted from low-level data. This is also termed as Deep Neural Network (DNN). It works on the principle of trial and error; therefore, it needs a large amount of data. DNN is being used in various fields including image recognition, computer vision, and natural language processing. Deep learning helps to recognize malicious content from micro details of the data. A deep learning model is capable enough to learn intricate feature rankings and integrate varied steps of malware detection into one compact model that is trained end-to-end, and therefore, all of the components of the DNN model are learned concurrently. Deep feature spaces, when discovered, can also help to detect malware variants [66].

D. GENERATIVE ADVERSARIAL LEARNING-BASED FINE-TUNING

In this study, machine learning (ML)-based malware detection models are optimized using GAN-based optimization. GAN is a methodology for generative modeling that generates a new set of data based on training data, which mimics the original data [67]. GAN is useful in generating realistic data that has never existed before. GAN has two main components: Generator and Discriminator, as shown in Figure 3. The generator generates fake samples that mimic the original samples to deceive the discriminator into identifying them as real samples. The discriminator identifies abnormalities in these generated data samples to classify them as fake [23]. The generator is a neural network that uses unsupervised learning approaches, including hidden layers, activations, and loss functions. The generator's training is based on feedback from the discriminator, and it stops when the discriminator is deceived [46], [47]. The discriminator is a classifier that identifies whether the data provided is real or fake. In this study, an external discriminator is used and optimized through rigorous training. The objective function is a min-max optimization formulation that optimizes the generator to minimize the objective/loss function while the discriminator aims to maximize the probability of correct label assignments to training instances and samples generated from G.

V. ARCHITECTURE OF THE PROPOSED ANTI-MALWARE SYSTEM

An AMS mostly describes malicious software detection and removal solutions. The need for an AMS ascends as we look at the year-wise malware growth (Figure 2).

We present the architecture of an AMS with a combination of conventional and GAN-based proposed frameworks.

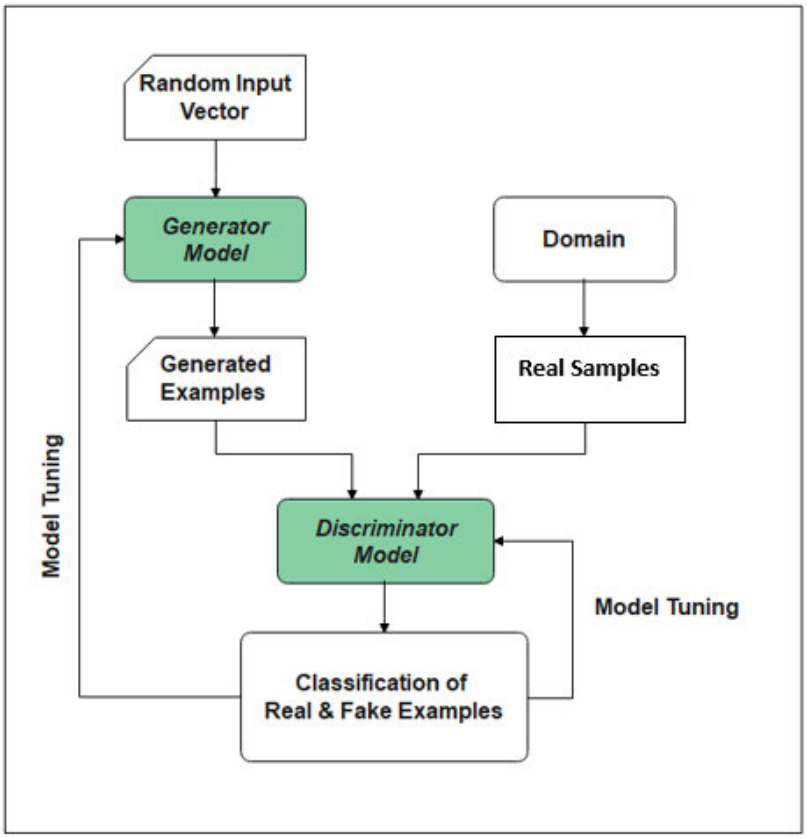


FIGURE 3. Architecture of GAN.

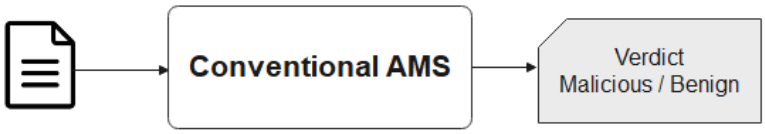


FIGURE 4. Conventional AMS Level-I.

This complete structural design is described on different levels, i.e., starting from getting a suspicious file (Figure 4), unwrapping multiple functionalities of AMS (Figure 5), and then approaching an AI-based framework (Figure 6) for enhancing the overall performance and robustness of the architecture. An AMS is split into three layers concerning the hierarchy of internal processing. These layers of filtering, unpacking, and detection engine modules are thoroughly explained below.

A. FILTRATION/UNPACKING MODULE

1) FILTERS

The working of the malware analysis system starts with the arrival of a suspicious file that needs to be checked. This file might have been downloaded or transferred from certain media. At this stage, immersions of filters begin. Filter, as its name suggests, screens the file for any malicious content. Different methodologies related to filter implementation can be stated as follows:

a: FILE SYSTEM FILTER (FS)

File system filters are commonly used for filtering everything from anti-malware and malware scanning to software license tracking and management, auditing, and change tracking on files. They inspect every open/read/write/set information calls from every application and filter Input/Output operations for file systems. These filters are very effective for scanning static signatures. Some filters like Mini Filters can be very useful for the implementation of filters in the AMS. They capture requests targetting a file system or other file system filters. By obstructing the request to reach its target, the filter driver replaces the working of the real target of the request. Similarly, Windows Filtering Platform (WFP) provides a platform with APIs and system services to create network filtering applications.

b: KERNEL MODE FILTERS

These filters can intercept anything. Many kernel-mode components are used for intercepting network/file system

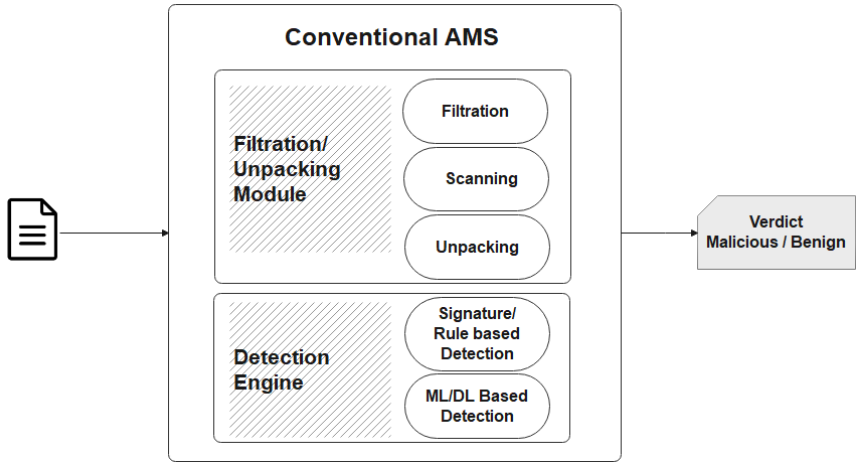


FIGURE 5. Conventional AMS Level-II.

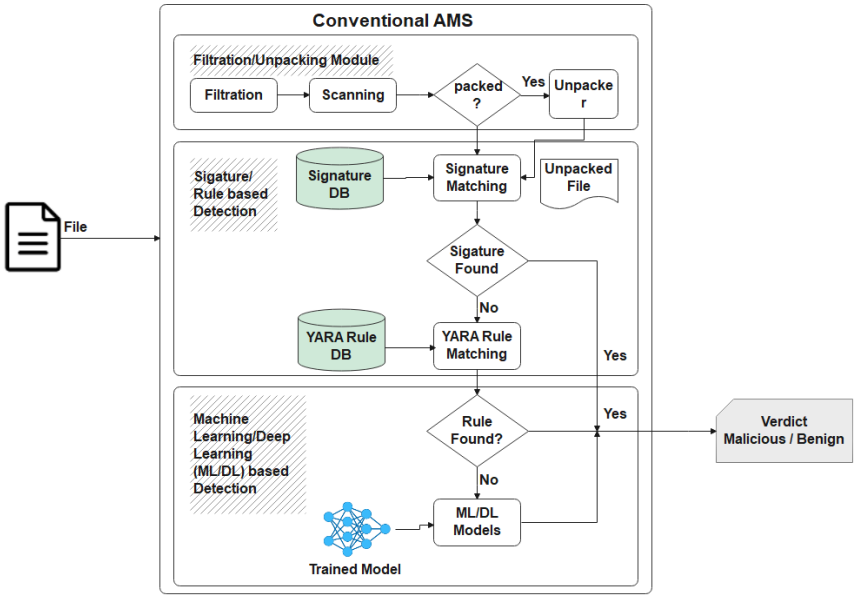


FIGURE 6. Conventional AMS Level-III.

activity even when the content is in its raw state. This is good for scanning static signatures.

c: NETWORK FILTERS

These filters can scan malicious files even before they are downloaded. As it can protect the entire network from malware, it eradicates the need to install software on individual devices as new threats are recognized.

2) UNPACKERS

As malware developers have started to use some protection methodologies to skip Anti-Malware (AM) scans, our proposed architecture of AMS is equipped with unpacking engines, i.e., unpackers and unpackers' databases. Programmers have looked at ways to safeguard their creations against intellectual theft ever since the field of software development first began. Many techniques have been employed, ranging from obfuscating the code to intricate multi-layered defenses, to reduce the files to a more manageable size while simultaneously preventing users from viewing the code itself. However, hackers have started employing countermeasures to conceal their harmful code and stay hidden when anti-malware solutions become popular. This packed malware needs to go through a process called "unpacking" for being ready to be analyzed. This process removes all protections and extracts the original code. At present, most of the malware's malicious code is packed for the sake of making malware analysis a tedious task. Packers use techniques to hide malware from anti-malware software, obscure malware analysis and cause

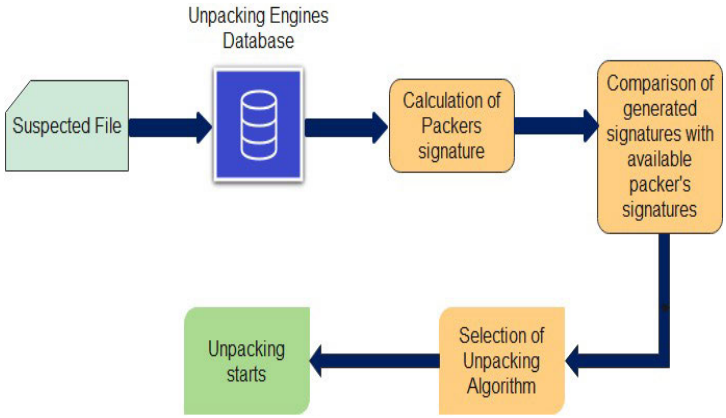


FIGURE 7. Working of unpacker.

shrinking of the malicious code. For devising the architecture of an AMS, it is mandatory to furnish it with unpackers, which can have the capacity to unpack code packed by any kind of packer. But before that, it is mandatory to learn about the structure and working of packers. Packers are not malicious by themselves; however, they can make malware more operative by being harder to be detected. For the protection of a system from malware and to reverse-engineer the malware-containing code, it is required to know about the most in-use packers to have good knowledge in hand.

a: IDENTIFICATION OF MALWARE PACKERS

The majority of malware packers make it difficult to locate and comprehend malicious code, hence using a script made especially for packer detection is required. However, there are several packer-detecting software available. Some of them are listed below:

1) Windows Executable Packer Detection
2) PackerID
3) PEiD
4) RDG Packer Detector

Besides identifying the malware packers, it is mandatory to establish a virtual environment and explore malware behavior. To unpack a suspected file, an unpacker engine database will be set up. When a match is found between the packer's signature and a database of signatures, these engines will scan a suspected file, select the appropriate unpacking technique, and proceed with the process (Figure 7).

3) SCANNERS

The core component of an anti-malware product is the scanner that is used for detecting patterns of bytes in file contents (Figure 9). It is very effective for static scanning. Lots of techniques have been proposed for the detection of maliciousness in the byte-level file content. The authors in [68] used opcode n-gram patterns for reaching to hidden malicious code. These patterns are used as a feature for classification. In [69], authors computed an extensive range of arithmetical and analytical features in a block-wise way to calibrate the byte-level file content. In [70], authors used structural and global entropy features from entropy of the suspected file. As more and more formats of files are evolving, it is becoming a challenge for this component to support dozens of file formats. It can support PE, Document Files, Script files, and even File-less malware.

4) MALWARE SANDBOXES

Malware sandboxes are specialized emulators designed to execute and observe the behavior of potentially malicious files. A Malware Sandbox is a piece of hardware or software that permits one computer system, i.e., host, to act like some other computer system, i.e., guest. A sandbox makes the host machine run software or use secondary devices intended for the guest system. It imitates only the execution of the sample itself. It momentarily creates objects that the sample interacts with, e.g., passwords, which malware intends to steal. With the sophistication of polymorphic malware like Stuxnet, it has become complex to detect them. Polymorphic malware unpacks itself at run-time and causes infection in the files with a new mutated malware body. Malware sandboxes deliver a much better detection rate against polymorphic samples. During the emulation, the system recognizes whether the suspected code is expected to exhibit malicious behavior. The sandboxes are highly dependent on manual analysis. They run N number of instructions and observe the state of memory (Figure 8). They are ineffective against targeted attacks that are designed to keep a low profile.

5) BEHAVIOR MONITORS

Behavior monitors used to be good before Microsoft disabled them with PatchGuard. An AM hooks critical system functions and monitors the actions of an application. They were heavily used by rootkits as well and were limited to user-mode malware. Recent versions of Windows provided

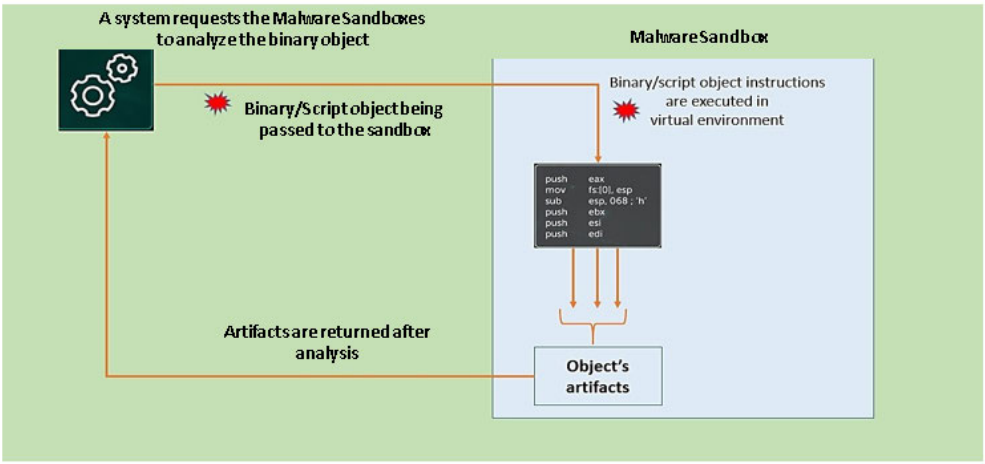


FIGURE 8. Working of a malware sandbox.

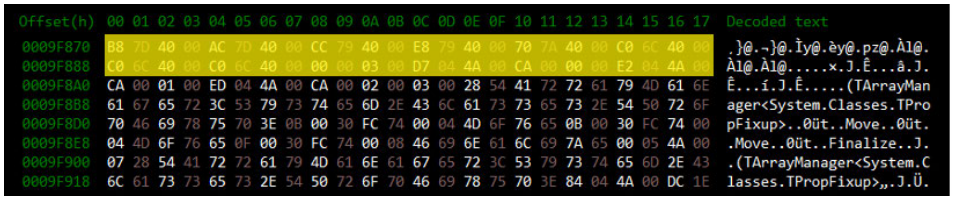


FIGURE 9. Sequence of bytes.

alternatives but in a very limited manner. People started using VT-x for monitoring the behavior.

B. SIGNATURE/RULE-BASED DETECTION

1) SIGNATURE DATABASE

An anti-malware signature database (DB) is one of the core components of an AM system. It holds the information needed for a signature-based scanner to identify and get rid of harmful code. A variety of malware signatures, such as definitions and distinct byte sequences associated with every malicious code fragment, are stored in the databases. Although nowadays, signature-based analysis is no more the first point of protection used to seize malware, it is the first place to start a malware analysis. It keeps a record of what is good and what is bad. It gets bigger and bigger every day. Some AM vendors even sanitize their DBs. It is something that the AM downloads from its server right after it is installed. There are two types of malware signatures. One-to-one signatures are used for new malware and generic signatures cover a malware family and mostly require manual analysis.

2) YARA RULE DATABASE

Our proposed architecture also contains a repository of YARA rules where different YARA signatures are classified and stored after compilation. YARA stands for Yet Another Recursive Acronym. These directives provide a way to differentiate between malware and other files by generating rules that look for specific features. YARA was initially developed by Victor Alvarez of Virustotal and is mainly used in malware study and detection. It was established with the idea of defining patterns that classify particular strains or complete families of malware. YARA helps generate malware family descriptions using binary or textual patterns. Each description, called a rule, is made up of a set of strings and a Boolean expression, which governs its logic, as shown in Figure 7. After a detailed study, it is obvious that full reliance on signature-based protection is no longer good enough. This method can easily be bypassed by the attacker's modern countermeasures. Different packers, polymorphism, and the use of various encrypting services can easily generate malware that is different enough so that it no longer matches existing signatures. So to complement the process of detection, YARA rules databases were made available to encompass a wide range of malware.

3) REPUTATION DATABASE

For a flawless anti-malware system, reputation-based security must be a part of an efficient AMS. This security mechanism categorizes a file as malicious or benign based on its intrinsically garnered reputation. So, based on reputation and usage, it becomes easy to identify and predict file safety.

C. MACHINE LEARNING/DEEP LEARNING (ML/DL) BASED DETECTION

The third level of our proposed AMS includes an ML/DL-based detection. Apart from a signature-based detection

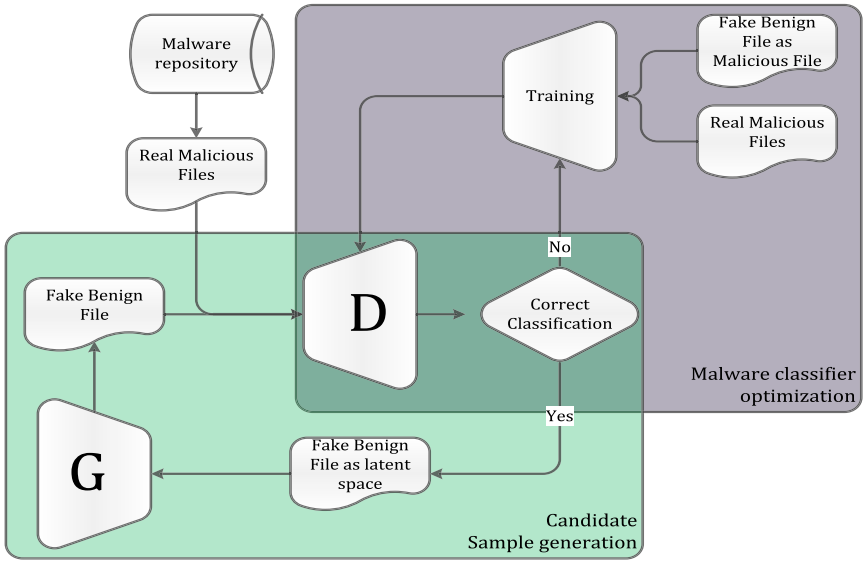


FIGURE 10. Proposed GAN-based framework.

engine, a state-of-the-art AMS needs to detect novel malware. For timely and accurate detection, machine learning and deep-learning-based models are built by researchers but they suffer due to the lack of data, which results in poor model training. Therefore, ML techniques are used. Detecting malware using ML techniques involves attaining suitable features and then training ML/DL models on derived features to identify malicious content. Initially, a set of features is extracted from training and testing data. Feature selection provides a set of limited and appropriate features. Machine learning-based algorithms are applied to medium-sized datasets. Parameter tuning based on cross-validation is performed till the model provides a good performance metric. Lastly, tests are executed on another, unseen dataset to get the final results.

VI. PROPOSED GAN-BASED FRAMEWORK

A. PRELIMINARIES

In this paper, a GAN-based framework is proposed to improve the performance and robustness of ML-based detection components. This well-trained model of the AMS will help detect anomalies even for zero-day malware detection [71]. By zero-day we mean to have data that has dispersion different from legitimate training samples. The generated fraudulent samples are ensured to have sufficient varying dispersion from the legitimate training samples. That makes the discriminator train on unknown malware also. The concept behind the proposed MCOGAN is derived from utilizing the power of data augmentation. Data augmentation is known as a procedure that aids in increasing the training data for AI-based models. GANs are used to generate imitation data founded on the training iterations over genuine data. Consequently, a lot of options are uncovered for training. This helps to improve the performance of a classifier, especially for security scenarios, because of the occurrence of unforeseen threats frequently [15]. So, GAN seems to be the optimal approach to train a neural network on a well-known attack to model alike attacks. GANs are deep learning-based generative models. They work on the principle of calculating the probability of every possible outcome for each variable and then combining them to predict the most likely outcome. When used in reverse, the probability distributions for each variable are sampled to generate new believable and independent feature values.

B. JUSTIFICATION

In our scenario, GANs have been used to create realistic malware images that are indistinguishable from the real malware. The images are then used to train a malware classifier, which learns to discriminate between malicious and benign software. For malware datasets, GANs are used to enhance the features of malware images, making them easier for a classifier to learn. This is done by increasing the contrast, sharpening the images, and removing noise.

C. METHODOLOGY

Fundamentally, two sub-models constitute the GAN architecture: a generator model for creating new samples and a discriminator model for identifying whether generated examples are real or fake, i.e., they belong to some real domain or are generated by the generator model. These two models are trained together. The generator generates a batch of samples. Then, these samples and real examples are sent to the discriminator and then the discriminator classifies them as real or fake. The discriminator is then updated based on the criterion of better discriminating real and fake samples in the next iteration and the generator is updated based

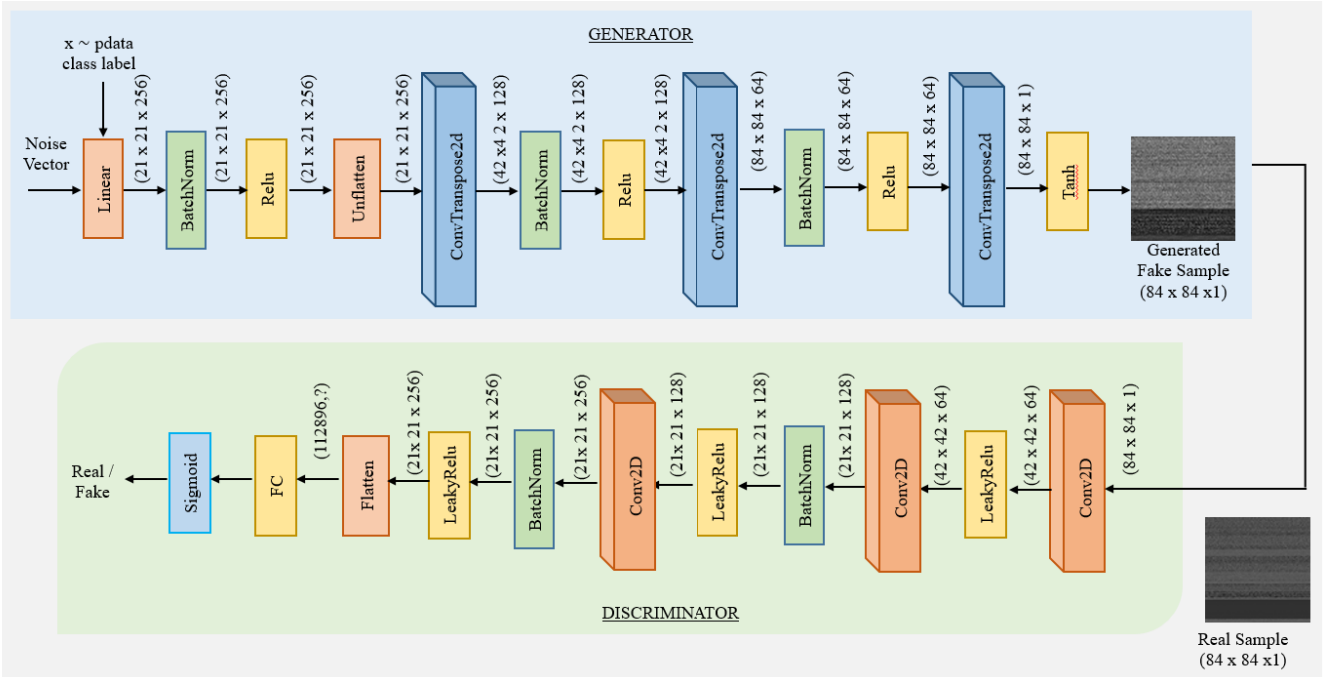


FIGURE 11. Structure of MCOGAN for malware image synthesis.

on the correctness of the generated samples to befool the discriminator (Figure 3).

Extending the principles that underlie GANs, this paper harnesses a similar paradigm to enhance the efficacy of a malware classifier. By adopting the GAN framework, the aim is to elevate the precision of the malware classifier to a heightened level. This strategic amalgamation is projected to exert a considerable influence on the overarching performance of the malware classifier, potentially ushering in a notable advancement in the domain of malware detection and analysis.

Our proposed MCOGAN framework constitutes two main parts (Figure 10).

1) Candidate Sample Generator (CSG):
   CSG is designed for generating malware samples. This is comprised of a generator and a discriminator. From the malware domain, a fixed-length vector is created to be given to the generator as input. This vector is then trained to reflect the compact rendition of the actual data dissemination. This vector, precisely sculpted via training, serves as a simplified illustration of the complicated patterns inherent in malware data transmission. It acts as a seed or a latent code that guides the generator's process of creating synthetic data, which in our case are malware images. The vector is made up of random 30 numbers derived from a steady and unvarying spread. When the generator receives this vector as input, it interprets the contained numbers as instructions on how to create a corresponding malware image. Each component of the vector influences different aspects of the image, such as its texture, structure, or content. The generator uses its learned parameters to transform these input numbers into visual features that resemble malware patterns. The generator transforms the input vector into an image format. It does so by employing various layers of neural networks and transformations. The result is a synthetic malware image of dimension $84 \times 84 \times 1$ that ideally captures the essence of the malware patterns embedded in the input vector. During training, the generator, G, refines its ability to translate the input vectors into realistic malware images. This is achieved through iterative optimization, where the generator's parameters are adjusted to minimize the discrepancy between its generated images and real malware images. Auto-encoders are utilized for the implementation of CSG. For the generator part, the encoders EN and $\widehat{EN}$ learn to obtain the representations of original features O and generated features $\widehat{O}$ respectively. The decoder DEC tries to reconstruct $\widehat{O}$ at the same time.
   The optimization procedure for CSG is presented as a minimax game between the generator and the discriminator. The Generator's goal is to minimize loss by producing synthetic data that the discriminator classifies as real. The discriminator in the system is implemented as a neural network, taking input samples from both genuine and generated data. It produces a probability, approaching one for authentic data and close to zero for synthetic data. The cross-entropy function assesses the error rate of the discriminator network, and this error is subsequently back-propagated to the generator and discriminator architectures to

fine-tune their respective weights. The discriminator keeps on minimizing loss by properly discriminating between real and synthetic data. The comprehensive configuration of both the generator and discriminator is depicted in Figure 11. The optimization is shown in the mathematical Eq. (1).

$$min_G max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \quad (1)$$

where
$V(D,G)$ is the Value Function to represent the GAN;
$x \sim p_{\text{data}}$ represents that x is drawn from distribution p_{data};
$z \sim p_z$ represents that z is drawn from distribution p_z;
p_{data} is the real distribution of training data;
$pz(z)$ shows the distribution of data generated by Generator (G).
In the architecture of the generator model, a series of transpose convolutional layers are interleaved with batch normalization layers, working collaboratively to transform the input noise vector into a 1-channel image with dimensions of 84 X 84. Similarly, the discriminator structure incorporates traditional convolutional layers to reduce the height and width of the feature maps. Experiments with different learning rates for both the generator and discriminator structures are conducted. To address overfitting, adjustments are made to dropout layers and additional data are introduced. Multiple dropout values are experimented with, and a value of 0.5 is determined as optimal during training. The rectified linear unit (ReLU) activation function is applied after each convolutional layer. In our study, we opt for the Adam Optimizer, known for its adaptive learning rate. The Adam optimizer systematically calculates gradients, updates weights, and iteratively progresses toward achieving optimal weights.

2) Malware Classifier Optimizer (MCO):
MCO is designed to train a malware classifier in case of misclassification. It constitutes a critical component of the proposed model, specifically focused on refining the performance of a malware classifier in scenarios where misclassification occurs. This component employs a systematic process of training to enhance the classifier's accuracy, particularly by learning from both authentic and synthesized malicious files. When the malware classifier incorrectly identifies a file, MCO comes into play. Instead of rejecting misclassified cases, these incorrect predictions are used to provide significant learning opportunities. By recognizing and correcting these misclassifications, the classifier can develop to handle similar scenarios more effectively in the future. These training iterations tune the classifier and thus lead to an optimized version. As the MCO is designed for universal application across various malware classifiers, the underlying process is elucidated through the equations below. Let D be the malware classifier and θ_D represent its parameters.

Loss Function for Misclassification:
Let $D(x)$ be the output of the malware classifier for input x, and y is the true label. The objective is to minimize the misclassification loss $L_{\text{misclassify}}$ with respect to the parameters θ_D:

$$\min_{\theta_D} L_{\text{misclassify}}(D(x), y)$$

Gradient Descent Update:
Utilizing Adam optimizer, the parameters θ_D are updated iteratively:

$$\theta_D \leftarrow \theta_D - \eta \nabla_{\theta_D} L_{\text{misclassify}}(D(x), y)$$

where η is the learning rate.
This formulation outlines the process of training the malware classifier using the MCO module to address misclassifications.

Our proposed MCOGAN model (Figure 10) takes real malware samples as input from the malware repository and sends them to the CSG module where existing discriminator D classifies them into benign and malicious files. In the case of correct classification, a latent space for fake benign files is built up based upon the tested file to follow a similar probability distribution as of the real files. The generative model in GAN design absorbs to map points in the latent space to newly generated images. The latent space structure has to be focused to understand the ways for interpolation between points and perform vector arithmetic between points, which have eloquent and directed effects on the generated images. Latent space will be based on reducing the distance between the real file and the generated file.

A fake benign file is a file that is neither to be claimed as malicious nor benign, yet it pretends to be benign. The latent space is input to Generator G for the generation of fake benign files. These fake benign files are again given to D along with real malicious files, which ultimately leads to the better performance of D. MCO plays its part when the given file is misclassified. Rigorous training keeps on improving the learning process with the help of malicious and fake benign files. These two modules complement each other by training over multiple files continuously generated by the CSG. This approach will minimize the security system's dependency on ML-based methodologies for data generation.

To provide a comprehensive understanding of our methodology and facilitate reproducibility, we present the pseudo-code (Algorithm 1) outlining the key components of our proposed MCOGAN framework for malware sample generation and classifier optimization. This pseudo-code offers a detailed breakdown of the operations performed within our framework, enabling researchers to grasp the intricacies of our approach and implement it in their work. The provided pseudo-code covers critical aspects of our methodology,

Algorithm 1 Proposed MCOGAN Framework

procedure CandidateSampleGenerator
    Initialize weights and biases for CSG's generator and discriminator
    Training:
      Initialize loss function for CSG's generator
    while not converged **do**
        Input: Random vector z (malware domain)
        Generate synthetic malware sample $G_{malware}(z)$
        Calculate discriminator's response $D_{malware}(G_{malware}(z))$
        Compute CSG generator loss based on $D_{malware}(G_{malware}(z))$
        Update CSG generator weights to minimize loss
    end while
end procedure
procedure MalwareClassifierOptimizer
    Initialize weights and biases for malware classifier
    Training:
      Initialize loss function for malware classifier
    while not converged **do**
        Input: Real or generated malware sample
        Perform classification using the malware classifier
        Compute classifier loss based on misclassification
        Update classifier weights to minimize loss
    end while
end procedure
procedure MinGeneratorMaxDiscriminator($V(G, D)$)
    Value Function:
    $V(G, D) =$ Expectation over real data$[log(D(x))]$ + Expectation over generated data$[log(1 - D(G(z)))]$

    Optimization:
    Initialize optimizer for generator and discriminator
    while not converged **do**
        Sample real data x from pdata (real data distribution)
        Sample random vector z from noise distribution
        Generate synthetic image $G(z)$
        Compute discriminator loss based on real/fake classification
        Update discriminator weights using the discriminator loss
        Compute generator loss based on discriminator response
        Update generator weights using the generator loss
    end while
end procedure
procedure Training
    Initialize CSG, MCO, and malware classifier
    Set hyperparameters (learning rates, batch size, etc.)
    for a fixed number of iterations **do**
        Sample real data batch from pdata
        Sample random noise vectors for generator and CSG
        Optimize generator, discriminator, CSG, and malware classifier using the minimax game and classifier optimization
    end for
end procedure

including the training of the generator and discriminator, the minimax game optimization, the Candidate Sample Generator (CSG), and the Malware Classifier Optimizer (MCO). Each section of the pseudo-code is accompanied by explanatory comments to elucidate the purpose and functionality of the code. Researchers interested in exploring and building upon our work will find this pseudo-code instrumental in achieving a deeper insight into our approach.

TABLE 4. Description of dataset.

Class Name	Samples
coinhive	405
emotet	907
fareit	1584
gandcrab	390
icedid	459
mirai	1440
ramnit	337
razy	668
gafgyt	1485
lamer	692
meapow	820
Benign	1000

VII. EXPERIMENTATION AND ANALYSIS

A. DATASET DESCRIPTION

The proposed model was assessed using a publicly available open-source dataset from Figshare [72]. We selected malicious samples from the Figshare malware dataset, which is a collection of malicious files that have been infected with malware. The dataset includes over 9187 files that have been initially collected from Malshare [73], Virusign [74] and Dasmalware [75]. The collective dataset is downloaded from Figshare [72]. A collection of 9187 malware files, belonging to 11 malware families, was obtained. The files have been infected with a wide range of malware, including viruses, worms, trojans, and other types of malicious software. The dataset includes samples of malware that target a wide range of Windows operating systems, including Windows 7, Windows 8, and Windows 10. It also includes malware that has been designed to evade detection by antivirus software, making it a valuable resource for researchers who are interested in developing new techniques for malware detection and analysis.

The benign files are collected from Figshare's ''Windows Portable Executable File Dataset for Malware Detection'' dataset. This dataset includes over 8,000 benign PE files, which were collected from various sources, including open-source software and Windows operating system files. The files are organized into different categories based on their purpose, such as system files, application files, and library files.

We created a mixed dataset comprising 10,187 samples, consisting of both benign and malicious files, to compare our approach with state-of-the-art deep ML models. The dataset contained 1000 benign files and 9187 malicious Portable Executable (PE) files, classified into various malware families listed in Table 4. The dataset size was around 5.5 Gigabytes, with 70% of the files used for training and the remaining 30% for testing.

B. MALWARE VISUALIZATIONS

Malware authors often create new malware by building on their previous code, only making small modifications or removing parts of the old code. By representing the entire binary file as an image, analysts can visually compare the differences between malware from different classes and identify unique patterns that distinguish them. The dataset on Figshare comprises PE files. However, since the classifiers that we have chosen work only with images, we need to convert the PE files into images before proceeding.

To transform a malware binary file into an image, we converted the file into a vector of 8-bit unsigned integers (Figure 12), where each vector represents a pixel value of the grayscale image within a range of 0 to 255. For this particular study, we have created square images of dimensions 84×84. To classify malware into specific families based on a few known samples, we employed visual similarity using well-studied few-shot models. Figure 13 showcases examples of images of specific malware families, highlighting their distinct visual characteristics. These visuals, sourced from the Figshare dataset, demonstrate that one can distinguish between the classes quite effortlessly through observation.

C. DATA AUGMENTATION

Our proposed MCOGAN was trained on the Figshare dataset, and we demonstrated its ability to enhance the performance of established classifiers by generating fresh images. During each iteration, the generator (G) received a Gaussian noise vector (zi) along with a batch of actual malware images. To create new data, we generated 5 random noise vectors for each input image to MCOGAN and obtained 5 new images from the generator. These images were then fed to the discriminator (D) to accurately classify them into their respective categories.

D. IMPLEMENTATION DETAILS

For our implementation task, we reviewed two previous studies that focused on malware classification. The first study, conducted by Choi et al. [76], proposed a deep Convolutional Neural Network (CNN) model to detect malware using image visualization. This model included two convolution layers with pooling and two fully connected layers, achieving an impressive accuracy of 96% in classifying benign and malicious samples. The second study, by Aslan et al. [77], designed a DNN hybrid model that classified malware variants using an optimized integration of two pre-trained network models. Their proposed model comprised five convolutional blocks with different sizes of convolution layers and combined the two pre-trained networks to create a feature vector through equal weighting. The authors collected data from Malimg and Malevis and visualized the malware binaries to improve the accuracy of their model.

For our proposed framework, we have utilized the classifier models during the training process to enhance the probability of correctly labeling both the real and generated instances. The initial step involves passing a collection of real malware samples from the malware repository through the classifiers, and the Discriminator Loss (DLoss) is then calculated based on the output. This approach builds on the strengths of the previous studies and provides an effective framework for

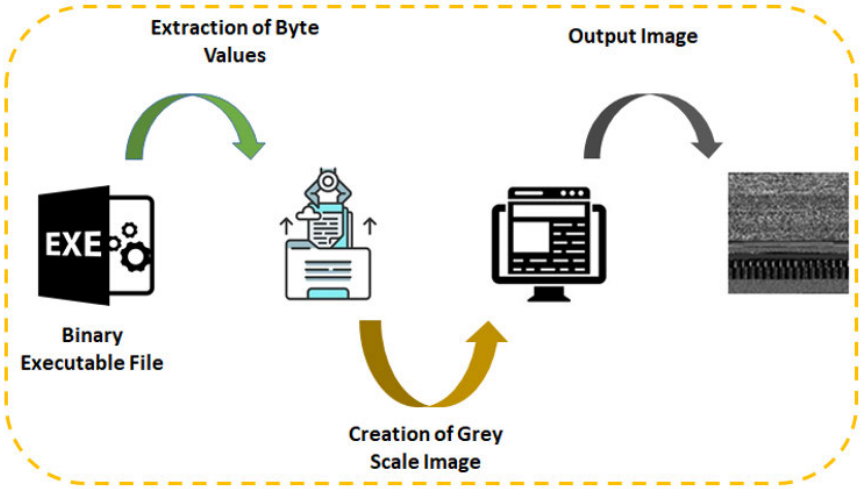


FIGURE 12. Malware conversion into image.

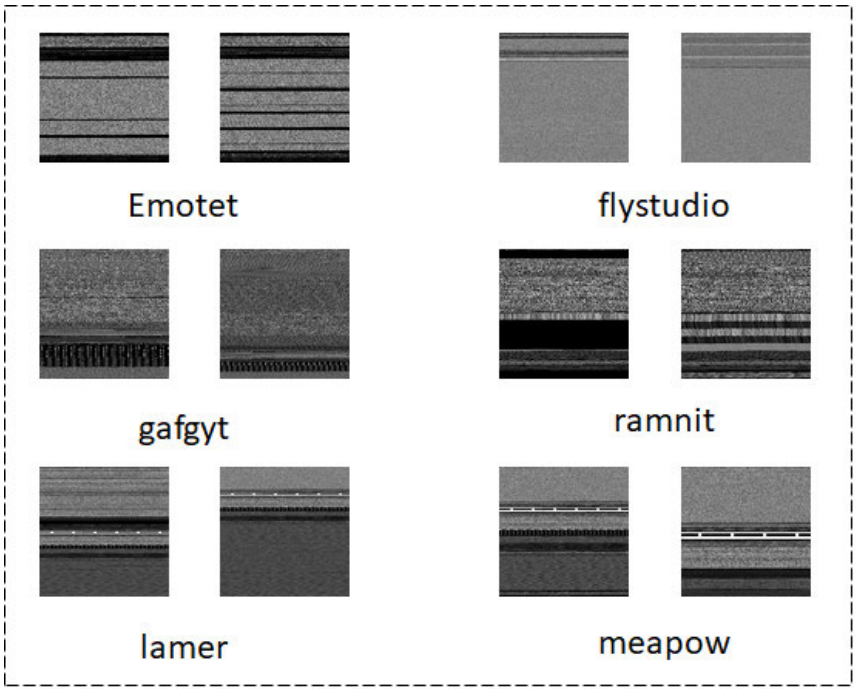


FIGURE 13. Malware visualization samples obtained from different categories.

improving malware classification accuracy.

$$DLoss = (log(D(x))) \tag{2}$$

where

DLoss represents the Discriminator Loss;

log(D(x)) represents log probability for real examples.

In a backward pass, the gradients are computed. Afterward, a batch of fake benign files is forward passed from the generator through D. Likewise, the Generator Loss (GLoss) is calculated as follows:

$$GLoss = \log(1 - D(G(z))) \tag{3}$$

where

log (1-D (G(z))) represents the log of the inverted probabilities of fake examples.

The backward pass is used to collect the gradients, which are then summed up for both the real and fake batches to optimize the discriminator. The objective of training the generator is to generate more accurate fake samples by minimizing GLoss. Conversely, the goal of the training

process is to maximize DLoss, which reduces the loss of the generator. The results of the generator are evaluated by passing them through the discriminator, and if the classification is incorrect, the MCO section is activated to retrain the classifier. The training process is performed not only in CSG but also in MCO, which enhances the robustness of the classifier by dual training.

E. PERFORMANCE INDICES

We use the same performance metrics as those used by the previously proposed models. Specificity, Sensitivity, and F1 score are measured to determine the correctness of the existing and enhanced models. Accuracy is used to evaluate the performance of both models.

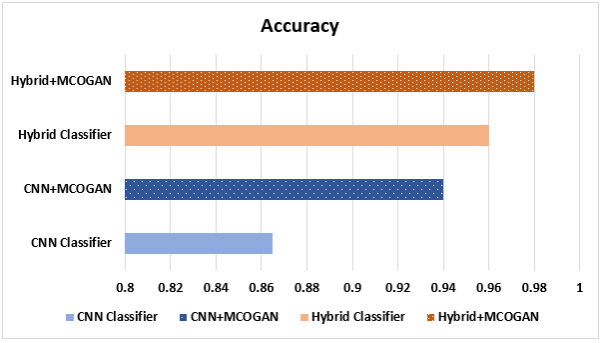


FIGURE 14. Accuracy-based comparison of the different models.

F. RESULTS ANALYSIS

The performance metrics of a model provide valuable insight into its quality. In this study, a CNN-based classifier achieved a malware classification accuracy of 87.5%, while a hybrid model reported an accuracy of 97%, claimed to be the best among all existing malware detection systems. To improve the accuracy further, we adopted the methodology proposed in MCOGAN. By continuously generating malicious, benign, and fake benign files and training the classifier, we enhanced its accuracy. The generator learned a generation distribution gd over data x and built a mapping function from a prior noise distribution to data space. The discriminator classified by calculating the probability that x came from training data rather than gd, using class labels as auxiliary information for both networks.

Initially, we trained both models on the Figshare dataset and evaluated their precision, sensitivity, and F-1 score. Then, we used each model as a discriminator with MCOGAN, training it on both real and fake data generated by the generator. Although the models showed good TPR, we iteratively trained them to reduce FPR by minimizing the loss through backpropagation after every misclassification. We repeated this process 500 times after the first misclassification, recording the FPR, which steadily declined over time (Figure 15). This iterative process improved the discriminator's performance and generated more benign instances that can be used for other models' evaluation, ultimately enhancing the overall quality of the system.

Our proposed model demonstrated an improvement in all performance metrics for both CNN-MCOGAN and Hybrid-MCOGAN, as presented in Figure 16. Figure 17 displays an overall performance gain in the two models after incorporating our proposed framework. Although the existing values for the performance metrics were good, our proposed model achieved even better results by extensively training the same models. Particularly for the Hybrid classifier, the F1-score improved and approached 1. Additionally, the proposed model had a positive impact on the accuracies of both models, as shown in Figure 14. CNN-MCOGAN accuracy increased by 10%, and the Hybrid classifier's accuracy improved by almost 9%. This indicates that increasing the effort in distinguishing generated images from real images during training leads to more rigorous training of the models. The impact of generated training images on accuracy for both hybrid and CNN models is presented in Figure 18.

GANs improved the results of malware classifiers by generating more training data. In the case of malware classification, the amount of data available for training can be limited, and in some cases, the data may not be diverse enough to capture the variations in malware instances. This limits the classifier's ability to generalize and identify new types of malware accurately. We used the ability to generate new, realistic malware instances that can be used to train the classifier. By generating synthetic instances, the classifiers are trained on a more comprehensive dataset that captured the full range of variations in malware instances. This improved the classifiers' ability to detect new types of malware, which in turn increased its overall accuracy. The iterative training process of the classifier also helped in reducing the false-positive rate, which is a critical factor in the performance of malware classifiers.

Overall, the use of GANs in malware classification helped us to overcome the limitations of traditional machine-learning techniques and enabled the performance enhancement of more accurate and effective models.

G. ANALYSIS OF PERFORMANCE ENHANCEMENT

The results indicated that training the malware classifier with GANs significantly improved its performance. Before implementing GANs, the classifier achieved a certain accuracy level, but after training with GANs, it performed much better. This suggests that the GANs played a crucial role in enhancing the model's ability to detect malware instances. It is also worth noting that the improvements may not have been possible without the use of GANs, highlighting the importance of such advanced techniques in enhancing machine learning models' performance.

Our proposed approach centers around the idea of an efficient AMS with the best combination of approaches to boost the performance of any detection model. We optimized various parameters such as loss function, training stopping criteria, and similarity measures to achieve an efficient and

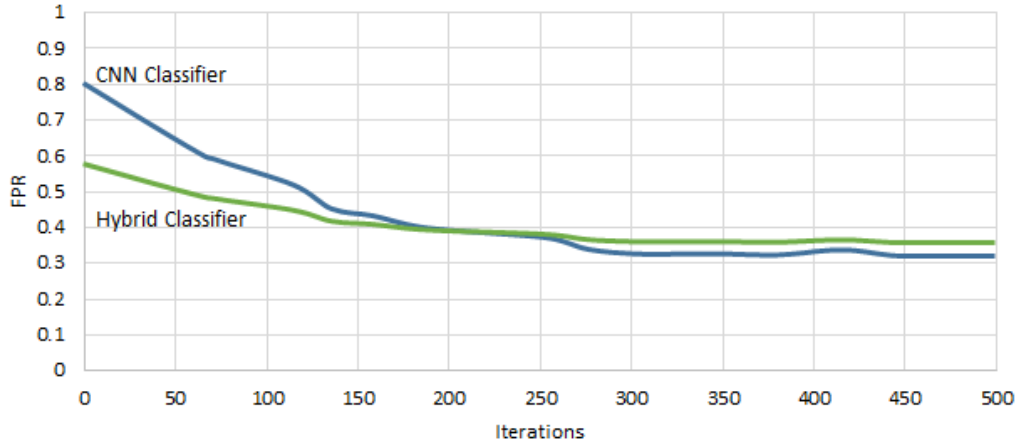


FIGURE 15. Decline in FPR of classifiers.

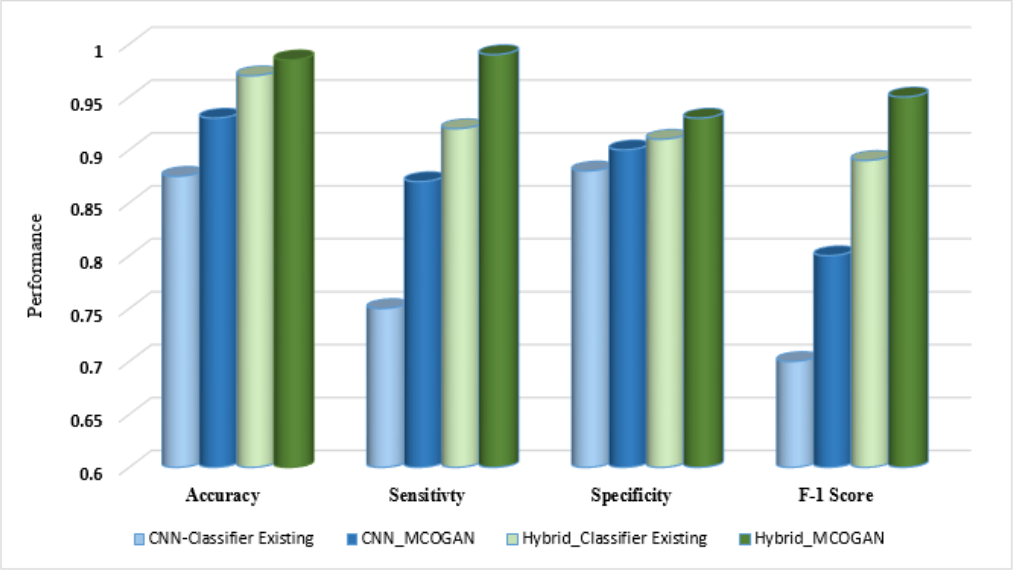


FIGURE 16. Performance metrics with and without MCOGAN.

optimal solution. While the convergence of the loss function is typically the stopping point for training DNN models, GANs benefit from the equilibrium between the generator and discriminator.

To assess the quality of our proposed model, we conducted an impartial evaluation using the ''nearest neighbors'' approach. We compared selected real samples with generated identical images and measured the Euclidean distance between the pixel information of these images to determine the degree of similarity. This similarity estimation helps us choose the generated sample most related to the real image, enabling us to evaluate how real the generated image appears. Our focus is on detecting new variants or zero-day attacks, and thus we set the threshold of the similarity index of two images to increase the accuracy of the classifier continually. We used the binary cross-entropy loss to optimize the discriminator's weights during training.

The relationship between GANs accuracy and the number of instances generated is directly proportional. This means that as the number of instances generated increases, the accuracy of the GAN-generated samples in the malware classification task also increases. During the later stages of the experiment, the model accuracy is observed to increase as the training approaches the advanced point. Once the GAN loss function converged, the accuracy reaches its maximum point, indicating the optimal efficacy of the trained classifier. Thus, generating more instances using GANs led to better accuracy in the malware classification task.

Considering both syntactic and semantic details of datasets used in malware detection is significant because it can lead to more accurate and robust malware detection systems. Syntactic information refers to the structural and behavioral aspects of the malware, such as the file size, byte sequence, or API calls made by the malware. This type of information is usually

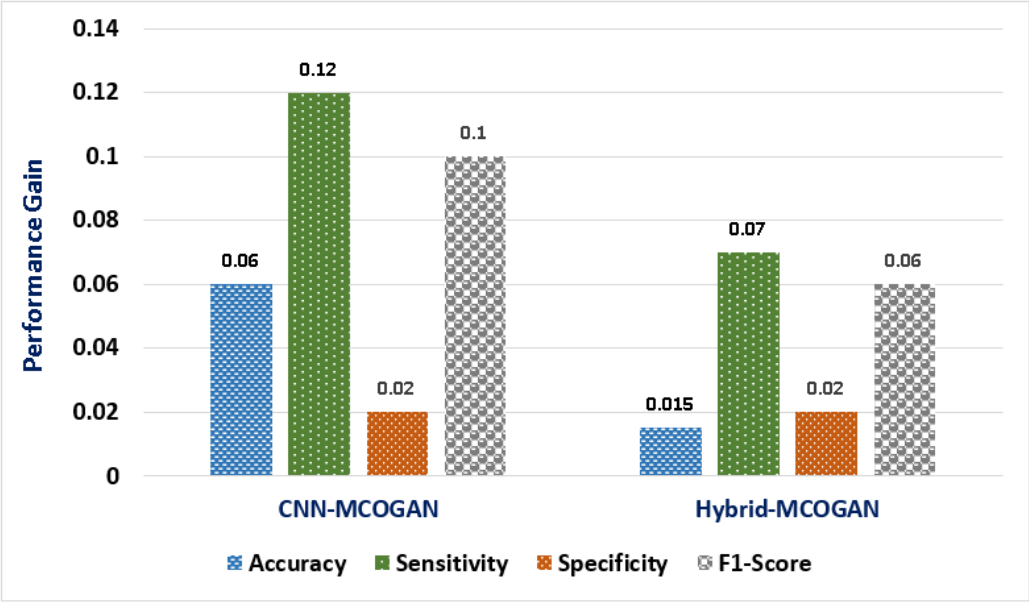


FIGURE 17. Performance gain of classifiers with MCOGAN.

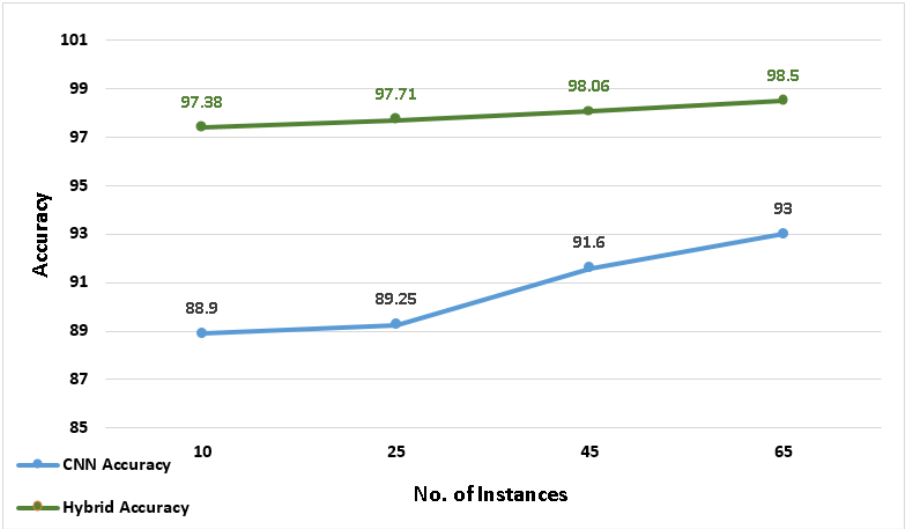


FIGURE 18. Impact of generated training images on accuracy.

used in traditional malware detection systems to identify and classify malware. However, malware creators can easily modify or obfuscate syntactic information to evade detection, making it necessary to consider semantic information as well. Semantic information refers to the meaning or intent of the malware's behavior, which can be inferred from the complete instruction set of the suspected file. By analyzing the semantic information of malware, detection systems can identify and detect variations of previously launched malware that have been modified to evade syntactic-based detection methods. Therefore, considering both syntactic and semantic details of datasets in malware detection can lead to more comprehensive and effective detection systems. Therefore, we emphasized the importance of syntactic as well as semantic details of the datasets used. It is noticed that most work has been done using the Malimg dataset as a benchmark, thus concentrating only on the syntactic information of the malware. In our work, we have employed semantic information also by preserving the complete instructions set of the suspected file.

H. COMPARATIVE ANALYSIS

The proposed architecture will have a great impact on modeling malware detection tasks. In this work, we have

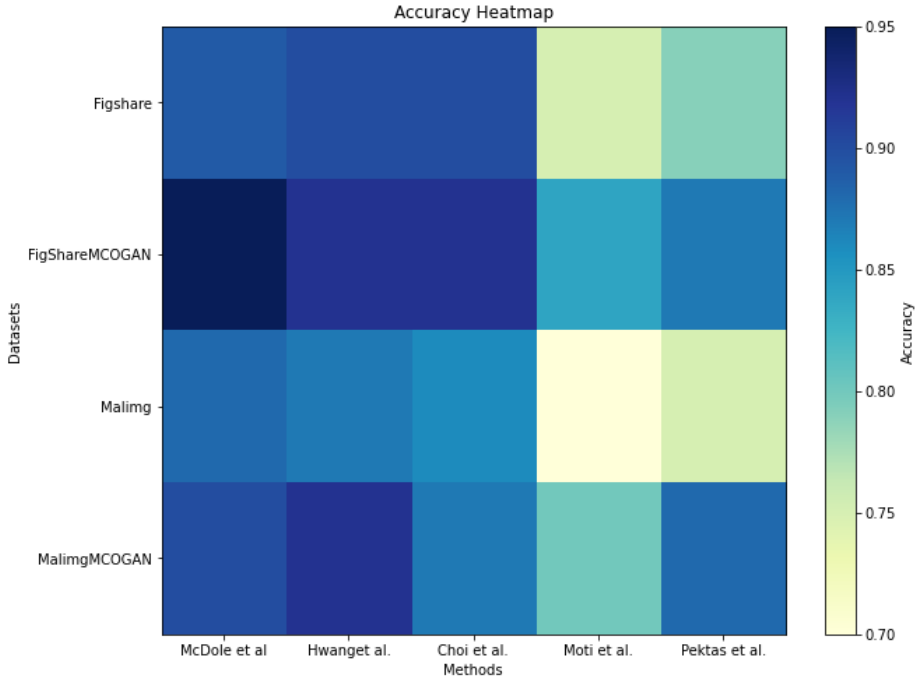


FIGURE 19. Accuracy comparison with different datasets using heatmap.

TABLE 5. Statistical comparative analysis.

Paper	Methodology	Performance	Limitation	Dataset	Analysis
McDole et al. [38]	Used malware's dynamic behavior for detecting malicious activity	Acc=90% F1=91.5%	FNR is high as no coverage for the detection of malware variants	VirusTotal	Quick but sacrifices accuracy. Skips to identify some malware due to low recall.
Hwang et al. [39]	Used API Sequence Call	accuracy=97%, FPR=4.83%, FNR=1.47%	The detection of malware variants is not covered	Custom	Not handling long API sequence calls
Choi et al. [49]	Employed Opcode sequence	Acc=75, ROC=0.98 %	Loss of information due to dropping off a long sequence	BIG2015	Misclassification of 25% cases is threatening. Only one dataset is used for experimentation.
Moti et al. [50]	Used PE Header	Performance increase of 2% for 3 classifiers	No coverage of semantic information	VXHeaven	The experiments are performed only on one dataset by using only headers. Generalization is missing for diverse malware families.
Pektas et al. [40]	Used API Call graphs	Acc=97.86%, F-measure=98.65%, Rec=98.47%, Prec=98.84%	Long API calls discarded	AndroZoo, AMD	Computationally expensive way for malware classification.
Proposed Work	**Our Proposed MCOGAN used GAN-based framework for the efficiency improvement of malware classifiers**	**Acc=98.85%, ROC=0.98%**	**-**	**Figshare, Malimg**	**Performance improvement achieved for various malware classifiers ranging from 10 to 12 points.**

proposed a GAN-based framework for playing our part in resolving the never-ending battle between attackers and the AMS. Due to its popularity, deep learning has always been a point of attention for researchers. Previously, researchers used different techniques to detect malware like data augmentation using VAE, ML-based algorithms, DNNs, etc. We have compared some of the related previous work with our proposed work (Table 5). We undertook this comparison with these studies because all of these prior papers utilized a similar dataset, specifically centered around malware. What unites them is their focus on conducting experiments within the domain of malware. We selected representative datasets based on their relevance to the specific malware detection scenarios addressed by the methods under comparison.

Moreover, the malware types and characteristics targeted by the compared methods remain consistent.

In [49], authors amplified the opcode part of a PE file and used it for malware detection. PE file was dissected for segregation of different parts, i.e., PE header, data, text, etc. They used GANs to amplify the malware using Opcode but they succeeded to gain 75% accuracy as they faced data loss due to dropping off the other parts of the suspected file. In [50], researchers came up with the concept of only using the suspected file's header. They generated the new data by using header information but they succeeded to enhance the accuracy by only 1%. Most of the early work has been done on malware features like opcodes, API call graphs [40], API calls sequencing [39], etc. These API calls and opcodes-based images deteriorate the absolute excellence of image-based methodologies. The authors in [38] proposed a model for malware detection by just covering the dynamic behavior of the malware. In [39], the authors emphasized the importance of API call sequence patterns of the malware. They described their results only on a specific dataset ignoring the fact that how well the model will be reusable for emerging malware. When compared with our proposed work, our proposed architecture focuses on the enhancement of the generalization capability of the malware classifiers. Moreover, MCOGAN is sustainable as it does not need retraining of the model for any new malware as training through adversarial examples has prepared the framework to deal with new variant scenarios. The statistical analysis reveals that our approach outperforms other methods in terms of accuracy and ROC. Our findings can be summarized as follows:

1) Performance Across Various Datasets:
   In prior studies, when evaluating their models with different datasets, such as Malimg and Figshare, the reported accuracies closely resembled their original findings.
   In the case of MCOGAN, an interesting and noteworthy trend emerges. When applying MCOGAN to these diverse datasets, a significant improvement of nearly 10% is observed in the performance of each model. In the case of Figshare, the accuracy of McDole et al. [38] initially stood at 0.85 but saw an improvement to 0.92. Similar observations were made for the Malimg dataset. When the approaches of Hwang et al. [39] and Choi et al. [49] were evaluated using Figshare and Malimg datasets, the accuracy increased by approximately 6 to 7%. This enhancement across various datasets is visually represented in Figure 19, highlighting the substantial impact of the MCOGAN framework on the models' capabilities.
2) Enhanced Data Augmentation:
   In previous comparative studies, we can observe various data augmentation strategies. One approach, for instance, employed GANs to amplify the opcode segment of a PE file, achieving a reasonable 75% accuracy in malware detection. However, this strategy incurred data loss, as it focused exclusively on the opcode segment while omitting other vital parts of the suspected file. Another study took a slightly different route by relying solely on the header information of suspected files, resulting in a marginal 1% boost in accuracy.
   Now, in the context of MCOGAN, we witness a transformative approach to data augmentation. MCOGAN's primary strength lies in its proactive stance against data loss. Instead of tunnel vision on a single aspect, MCOGAN prioritizes enhancing the generalization capabilities of malware classifiers. This strategic shift enables MCOGAN to outshine the limitations of data loss and positions it as a sustainable, adaptable solution.
3) Exploration of Malware Features:
   The majority of the mentioned studies primarily emphasized different aspects of malware features, including opcodes, API call graphs, API call sequencing, and more. While these features have proven to be effective, they might not fully leverage the potential of image-based methodologies.
   Our proposed MCOGAN architecture addresses these limitations and enhances the generalization capability of malware classifiers. It provides a sustainable solution that can adapt to emerging malware scenarios without retraining.

In comparison to previous studies, our approach of using GANs with malware images has improved the detection rate efficiently. Due to the similarity of features, it was easier for the attackers to generate adversarial samples related to these features and attacks on different but similar models with the same technique. In our work, we have focused on utilizing the whole EXE file by converting it into a greyscale image. The percentage of added noise quantized by the number of bits changed is minimized. Moreover, the proposed MCOGAN carried out adversarial training by combining the generated samples with the real training samples to increase the model's robustness. The performance is improved by 10%, which is a good indicator of the efficacy of our proposed work.

VIII. CONCLUSION AND FUTURE WORK

Malicious software (Malware) is one of the foremost threats on the Internet today. Many problems of data security arise due to the unstopping propagation of malware. In the past few years, several techniques have been developed for the in-depth inspection of malware, ranging from static code review to dynamic exploration of malware behavior. However, the previous work lacked a complete system for the timely detection of malware combining analysis and detection along with signature and machine learning-based detection engines. Therefore, it was much required to present an architecture to detect both existing as well as novel malware in the network. Keeping that in view, we presented a well-integrated hybrid architecture of an Anti-Malware System (AMS) comprising conventional signature-based detection along with AI-based detection modules. Moreover, we proposed a robust framework for enhancing the optimality

of malware classifiers. We presented a GAN-based Malware Classifier Optimizer (MCOGAN) with the capability of generating fake benign files to test any state-of-the-art malware discriminator, which can act as a trainer of the training module for enhancing the optimality of that particular discriminator. We conducted the experiments using the Figshare dataset. Two state-of-the-art models were used as discriminators. The accuracies were enhanced by a good percentage for both the models.

In the proposed work, the complete architecture of an AMS is presented, which is effective in detecting malware with increased efficiency. In the future, we aim to measure the resilience of the proposed architecture by using several unknown variants of malware. We aim to improve the training part of the MCO component along with improvement in the generator of the samples. We plan to conduct experiments with more datasets, such as Malimg and Malevis. Expanding our research endeavors in this direction involves delving into several potential research areas. A crucial aspect is the comprehensive assessment of the AMS's performance across diverse datasets to guarantee its adaptability to various malware families and their distinct variations. This evaluation should encompass datasets featuring a spectrum of file formats, sizes, and obfuscation techniques.

Additionally, an avenue for further exploration lies in the refinement and extension of the Generative Adversarial Network (GAN) employed within the AMS. This entails a thorough examination of advanced GAN variants or customized modifications explicitly designed for the generation of both malicious and benign samples. Such enhancements aim to fortify the AMS's ability to discern and effectively counter a broader array of malware scenarios, thereby contributing to the system's overall robustness.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia for funding this research through the project no. (IFKSUOR3-507-2).

REFERENCES

[1] S. Irum, A. Ali, F. A. Khan, and H. Abbas, "A hybrid security mechanism for intra-WBAN and inter-WBAN communications," *Int. J. Distrib. Sensor Netw.*, vol. 9, no. 8, Aug. 2013, Art. no. 842608.
[2] *Microsoft Security Intelligence Report*. Accessed: Jun. 30, 2023. [Online]. Available: http://www.microsoft.com/security/about/sir.aspx
[3] U.-E.-H. Tayyab, F. B. Khan, M. H. Durad, A. Khan, and Y. S. Lee, "A survey of the recent trends in deep learning based malware detection," *J. Cybersecur. Privacy*, vol. 2, no. 4, pp. 800–829, Sep. 2022.
[4] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *Proc. 12th Conf. USENIX Secur. Symp.*, vol. 12, Berkeley, CA, USA, 2003, p. 12.
[5] M. Belaoued, A. Boukellal, M. A. Koalal, A. Derhab, S. Mazouzi, and F. A. Khan, "Combined dynamic multi-feature and rule-based behavior for accurate malware detection," *Int. J. Distrib. Sensor Netw.*, vol. 15, no. 11, Nov. 2019, Art. no. 155014771988990.
[6] M. Belaoued, A. Derhab, S. Mazouzi, and F. A. Khan, "MACoMal: A multi-agent based collaborative mechanism for anti-malware assistance," *IEEE Access*, vol. 8, pp. 14329–14343, 2020.
[7] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 646–656, Mar. 2013.
[8] N. Amjad, H. Afzal, M. F. Amjad, and F. A. Khan, "A multi-classifier framework for open source malware forensics," in *Proc. IEEE 27th Int. Conf. Enabling Technol., Infrastructure Collaborative Enterprises (WETICE)*, Jun. 2018, pp. 106–111.
[9] F. B. Khan, M. H. Durad, A. Khan, F. A. Khan, S. H. Chauhdary, and M. Alqarni, "Detection of data scarce malware using one-shot learning with relation network," *IEEE Access*, vol. 11, pp. 74438–74457, 2023.
[10] A. Khan, Z. Rauf, A. Sohail, A. Rehman, H. Asif, A. Asif, and U. Farooq, "A survey of the vision transformers and its CNN-transformer based variants," 2023, *arXiv:2305.09880*.
[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020.
[12] S. Treadwell and M. Zhou, "A heuristic approach for detection of obfuscated malware," in *Proc. IEEE Int. Conf. Intell. Secur. Informat.*, Jun. 2009, pp. 291–299.
[13] G. Berger-Sabbatel, M. Korczyński, and A. Duda, "Architecture of a platform for malware analysis and confinement," in *Proc. MCSS*, 2010, pp. 1–7.
[14] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, "Idea: Opcode-sequence-based malware detection," in *Proc. Int. Symp. Eng. Secure Softw. Syst. (ESSoS)*. Pisa, Italy: Springer, Feb. 2010, pp. 35–43.
[15] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *J. Mach. Learn. Res.*, vol. 2721, no. 2744, Dec. 2006.
[16] I. Firdausi, C. Lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *Proc. 2nd Int. Conf. Adv. Comput., Control, Telecommun. Technol.*, Dec. 2010, pp. 201–203.
[17] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in API calls with machine learning algorithms for malware detection," in *Proc. 2nd ACM workshop Secur. Artif. Intell.*, Nov. 2009, pp. 55–62.
[18] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Comput. Sci. Rev.*, vol. 34, Nov. 2019, Art. no. 100199.
[19] Y. Li, K. Xiong, T. Chin, and C. Hu, "A machine learning framework for domain generation algorithm-based malware detection," *IEEE Access*, vol. 7, pp. 32765–32782, 2019.
[20] I. M. M. Matin and B. Rahardjo, "Malware detection using honeypot and machine learning," in *Proc. 7th Int. Conf. Cyber IT Service Manage. (CITSM)*, vol. 7, Nov. 2019, pp. 1–4.
[21] Sudhakar and S. Kumar, "An emerging threat fileless malware: A survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, Dec. 2020.
[22] I. K. Dutta, B. Ghosh, A. Carlson, M. Totaro, and M. Bayoumi, "Generative adversarial networks in security: A survey," in *Proc. 11th IEEE Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2020, pp. 399–405.
[23] H. Navidan, P. F. Moshiri, M. Nabati, R. Shahbazian, S. A. Ghorashi, V. Shah-Mansouri, and D. Windridge, "Generative adversarial networks (GANs) in networking: A comprehensive survey & evaluation," *Comput. Netw.*, vol. 194, Jul. 2021, Art. no. 108149.
[24] C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, and Q. Wu, "AVOIDIT: A cyber attack taxonomy," in *Proc. 9th Annu. Symp. Inf. Assurance (ASIA)*, 2014, pp. 2–12.
[25] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in *Proc. Int. Symp. Distrib. Comput. Artif. Intell.*, A. Abraham, J. M. Corchado, S. R. González, and J. F. D. Paz Santana, Eds. Berlin, Germany: Springer, 2011, pp. 415–422.
[26] A. K. Sood and S. Zeadally, "A taxonomy of domain-generation algorithms," *IEEE Secur. Privacy*, vol. 14, no. 4, pp. 46–53, Jul. 2016.
[27] F. Ullah, M. Edwards, R. Ramdhany, R. Chitchyan, M. A. Babar, and A. Rashid, "Data exfiltration: A review of external attack vectors and countermeasures," *J. Netw. Comput. Appl.*, vol. 101, pp. 18–54, Jan. 2018.
[28] S. Singh. *Breaking the Sandbox*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.exploit-db.com/docs/english/34591-breaking-the-sandbox.pdf
[29] E. Nasi. *Bypass Antivirus Dynamic Analysis*. Accessed: Jan. 15, 2024. [Online]. Available: http://www.sevagas.com/
[30] (2015). *Just-in-Time Malware Assembly: Advanced Evasion Techniques*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.ten-Inc.com/presentations/invincea3.pdf

[31] R. R. Branco and U. Shamir, "Architecture for automation of malware analysis," in *Proc. 5th Int. Conf. Malicious Unwanted Softw.*, Oct. 2010, pp. 106–112.

[32] A. A. E. Elhadi, M. A. Maarof, and A. H. Osman, "Malware detection based on hybrid signature behaviour application programming interface call graph," *Amer. J. Appl. Sci.*, vol. 9, no. 3, pp. 283–288, Mar. 2012.

[33] J. Lee, K. Jeong, and H. Lee, "Detecting metamorphic malwares using code graphs," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2010, pp. 1970–1977.

[34] L. Li, Y. Ding, B. Li, M. Qiao, and B. Ye, "Malware classification based on double byte feature encoding," *Alexandria Eng. J.*, vol. 61, no. 1, pp. 91–99, Jan. 2022.

[35] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 38–49.

[36] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining and inference for malware detection," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2011, pp. 1–8.

[37] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *J. Comput. Virol.*, vol. 7, no. 4, pp. 247–258, Nov. 2011.

[38] A. McDole, M. Abdelsalam, M. Gupta, and S. Mittal, "Analyzing CNN based behavioural malware detection techniques on cloud IaaS," in *Proc. 13th Int. Conf. Cloud Comput., Services Conf. Fed. (CLOUD/SCF)*. Honolulu, HI, USA: Springer, 2020, pp. 64–79.

[39] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Pers. Commun.*, vol. 112, no. 4, pp. 2597–2609, Jun. 2020.

[40] A. Pektaş and T. Acarman, "Deep learning for effective Android malware detection using API call graph embeddings," *Soft Comput.*, vol. 24, no. 2, pp. 1027–1043, Jan. 2020.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[42] H. S. Anderson, J. Woodbridge, and B. Filar, "DeepDGA: Adversarially-tuned domain generation and detection," in *Proc. ACM Workshop Artif. Intell. Secur.*, 2016, pp. 13–21.

[43] H. Chen and L. Jiang, "Efficient GAN-based method for cyber-intrusion detection," 2019, *arXiv:1904.02426*.

[44] R. Burks, K. A. Islam, Y. Lu, and J. Li, "Data augmentation with generative models for improved malware detection: A comparative study," in *Proc. IEEE 10th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2019, pp. 660–665.

[45] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," 2017, *arXiv:1710.11342*.

[46] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting classifiers against adversarial attacks using generative models," 2018, *arXiv:1805.06605*.

[47] F. Yu, L. Wang, X. Fang, and Y. Zhang, "The defense of adversarial example with conditional generative adversarial networks," *Secur. Commun. Netw.*, vol. 2020, pp. 1–12, Aug. 2020.

[48] J. Yang, W. Zhang, J. Liu, J. Wu, and J. Yang, "Generating de-identification facial images based on the attention models and adversarial examples," *Alexandria Eng. J.*, vol. 61, no. 11, pp. 8417–8429, Nov. 2022.

[49] C. Choi, S. Shin, and I. Lee, "Opcode sequence amplifier using sequence generative adversarial networks," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2019, pp. 968–970.

[50] Z. Moti, S. Hashemi, and A. Namavar, "Discovering future malware variants by generating new malware samples using generative adversarial network," in *Proc. 9th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2019, pp. 319–324.

[51] S. L. Hansman, "A taxonomy of network and computer attack methodologies," Tech. Rep., 2003.

[52] F. A. Khan and A. Gumaei, "A comparative study of machine learning classifiers for network intrusion detection," in *Proc. 5th Int. Conf. Artif. Intell. Secur. (ICAIS)*. New York, NY, USA: Springer, Jul. 2019, pp. 75–86.

[53] K. Ivaturi and L. Janczewski, "A taxonomy for social engineering attacks," in *Proc. Int. Conf. Inf. Resour. Manag.* Cuttack, Odisha: Centre For Information Technology Organization, 2011, pp. 1–12.

[54] S. Akarsh, S. Sriram, P. Poornachandran, V. K. Menon, and K. P. Soman, "Deep learning framework for domain generation algorithms prediction using long short-term memory," in *Proc. 5th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Mar. 2019, pp. 666–671.

[55] E. Bergenholtz, E. Casalicchio, D. Ilie, and A. Moss, "Detection of metamorphic malware packers using multilayered LSTM networks," in *Proc. 22nd Int. Conf. Inf. Commun. Secur.(ICICS)*. Copenhagen, Denmark: Springer, Aug. 2020, pp. 36–53.

[56] B. N. Sanjay, D. C. Rakshith, R. B. Akash, and D. V. V. Hegde, "An approach to detect fileless malware and defend its evasive mechanisms," in *Proc. 3rd Int. Conf. Comput. Syst. Inf. Technol. Sustain. Solutions (CSITSS)*, Dec. 2018, pp. 234–239.

[57] Logix Computer Consulting. (2022). *Single vs Two-Stage Malware: What You Should Know*. [Online]. Available: https://logixconsulting.com/2022/02/09/single-vs-two-stage-malware-what-you-should-know/

[58] F. Suthar, N. Patel, and S. V. O. Khanna, "A signature-based botnet (emotet) detection mechanism," *Int. J. Eng. Trends Technol.*, vol. 70, no. 5, pp. 185–193, May 2022.

[59] S. P. Niraj and A. K. Tiwari, "Performance analysis of signature based and behavior based malware detection," *Res. J. Eng. Technol. Med. Sci.*, vol. 5, no. 4, pp. 193–199, 2022.

[60] T. Muralidharan, A. Cohen, N. Gerson, and N. Nissim, "File packing from the malware perspective: Techniques, analysis approaches, and directions for enhancements," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–45, May 2023.

[61] H. E. Merabet and A. Hajraoui, "A survey of malware detection techniques based on machine learning," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 1, 2019.

[62] N. Rani, S. V. Dhavale, A. Singh, and A. Mehra, "A survey on machine learning-based ransomware detection," in *Proc. 7th Int. Conf. Math. Comput.*, D. Giri, K.-K. R. Choo, S. Ponnusamy, W. Meng, S. Akleylek, and S. P. Maity, Eds. Singapore: Springer, 2022, pp. 171–186.

[63] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "OPEM: A static-dynamic approach for machine-learning-based malware detection," in *Proc. Int. Joint Conf. (CISIS/ICEUTE/SOCO)*. Berlin, Germany: Springer, 2013, pp. 271–280.

[64] K. Sethi, R. Kumar, L. Sethi, P. Bera, and P. K. Patra, "A novel machine learning based malware detection and classification framework," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Security)*, Jun. 2019, pp. 1–4.

[65] G. Pitolli, G. Laurenza, L. Aniello, L. Querzoni, and R. Baldoni, "MalFamAware: Automatic family identification and malware classification through online clustering," *Int. J. Inf. Secur.*, vol. 20, no. 3, pp. 371–386, Jun. 2021.

[66] M. Asam, S. J. Hussain, M. Mohatram, S. H. Khan, T. Jamal, A. Zafar, A. Khan, M. U. Ali, and U. Zahoora, "Detection of exceptional malware variants using deep boosted feature spaces and machine learning," *Appl. Sci.*, vol. 11, no. 21, p. 10464, Nov. 2021.

[67] F. Fahimi, S. Dosen, K. K. Ang, N. Mrachacz-Kersting, and C. Guan, "Generative adversarial networks-based data augmentation for brain–computer interface," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 9, pp. 4039–4051, Sep. 2021.

[68] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on OpCode patterns," *Secur. Informat.*, vol. 1, no. 1, pp. 1–22, Dec. 2012.

[69] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proc. ACM SIGKDD Workshop CyberSecurity Intell. Informat.*, Jun. 2009, pp. 23–31.

[70] L. Liu, X. He, L. Liu, L. Qing, Y. Fang, and J. Liu, "Capturing the symptoms of malicious code in electronic documents by file's entropy signal combined with machine learning," *Appl. Soft Comput.*, vol. 82, Sep. 2019, Art. no. 105598.

[71] N. A. S. Mirza, H. Abbas, F. A. Khan, and J. Al Muhtadi, "Anticipating advanced persistent threat (APT) countermeasures using collaborative security mechanisms," in *Proc. Int. Symp. Biometrics Secur. Technol. (ISBAST)*, Aug. 2014, pp. 129–132.

[72] *Figshare*. Accessed: Jan. 15, 2024. [Online]. Available: https:www.figshare.com

[73] *MalShare*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.malshare.com

[74] *Virusign*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.virusign.com

[75] *DasMalwerk*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.dasmalwerk.com

[76] S. Choi, S. Jang, Y. Kim, and J. Kim, "Malware detection using malware image and deep learning," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2017, pp. 1193–1195.

[77] Ö. Aslan and A. A. Yilmaz, "A new malware classification framework based on deep learning algorithms," *IEEE Access*, vol. 9, pp. 87936–87951, 2021.

FAIZA BABAR KHAN received the M.S. degree in computer science with machine learning as a major subject. She is currently pursuing the Ph.D. degree in computer science with the Pakistan Institute of Engineering and Applied Sciences, Islamabad. Her research interest includes novel malware detection using few-shot learning techniques.

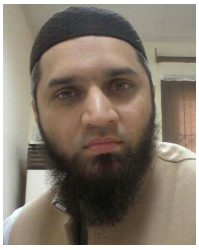
MUHAMMAD HANIF DURAD received the M.Sc. degree in physics from the University of the Punjab, Pakistan, the M.Sc. degree in systems engineering from Quaid-i-Azam University, Pakistan, and the Ph.D. degree in computer engineering from the Beijing Institute of Technology (BIT), China. He is currently a Professor with the Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan. He has more than 40 publications in refereed international journals and conferences. He has received a few research grants for ICT-related projects from various national funding agencies. His research interests include computer networks, cyber security, cloud computing, parallel computing, grid computing, the Internet of Things (IoT), and computer architecture.

ASIFULLAH KHAN received the M.S. and Ph.D. degrees in computer systems engineering from the GIK Institute, Pakistan.

He is currently leading the PIEAS Artificial Intelligence Center (PAIC), Pakistan Institute of Engineering and Applied Sciences (PIEAS). He is a highly accomplished professor with over 24 years of teaching and research experience. He has made significant contributions to the field of artificial intelligence with more than 120 international journal publications with more than 8,000 citations. He has also published 55 conference papers and eight book chapters. He has a proven track record of supervising Ph.D. scholars, with 24 successful Ph.D. supervisions to his credit. He is a highly respected researcher and has won eight research grants as a Principal Investigator. He has been actively involved in deep learning research. From 2016 to 2020, he was the Head of the Department of Computer and Information Sciences, PIEAS.

Dr. Khan has been awarded the President's Award for Pride of Performance for the year 2018 and has received four HEC's Outstanding Research Awards and one Best University Teachers Award. He also received the PAS-COMSTECH Prize 2011 in computer science and I.T. and Research Productivity Awards from the Pakistan Council for Science and Technology (PCST), from 2012 to 2016. In addition, he won the Youm-e-Takbir Performance Gold Medal by PAEC, in 28th May 2017. He has been listed in the World's Top 2% Scientists by Stanford University in both Career-Long and last-year categories, in 2020 and 2021.

FARRUKH ASLAM KHAN (Senior Member, IEEE) received the M.S. degree in computer system engineering from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Pakistan, in 2003, and the Ph.D. degree in computer engineering from Jeju National University, South Korea, in 2007. He also received professional training from the Massachusetts Institute of Technology, New York University, IBM, and other professional institutions. He is currently a Professor of cybersecurity with the Center of Excellence in Information Assurance, King Saud University, Riyadh, Saudi Arabia. He is also a Visiting Professor with Universiti Tenaga Nasional (UNITEN), Malaysia. He has more than 135 publications in refereed international journals and conferences. He has co-organized several international conferences and workshops. He has successfully supervised/co-supervised six Ph.D. students and more than 20 M.S. thesis students. His research interests include cybersecurity, body sensor networks and e-health, bio-inspired and evolutionary computation, smart grid, and the Internet of Things. He is a fellow of the British Computer Society (BCS). He has won several research grants from various funding agencies. His name has been listed in the World's Top 2% Scientists in a study conducted by Stanford University, in 2022 and 2023. He is on the panel of reviewers of more than 50 reputed international journals and numerous international conferences. He serves/served as an Associate Editor for prestigious international journals, including IEEE Access, *PLOS ONE*, *Neurocomputing* (Elsevier), *Ad Hoc and Sensor Wireless Networks*, *KSII Transactions on Internet and Information Systems*, *Human-Centric Computing and Information Sciences* (Springer), *PeerJ Computer Science*, and *Complex and Intelligent Systems* (Springer).



MUHAMMAD RIZWAN received the M.Sc. degree in software engineering from Riphah International University, Islamabad, and the Ph.D. degree in computer sciences from the Capital University of Science and Technology (CUST), Islamabad, Pakistan. His master's thesis focused on the crucial area of software fault tolerance, demonstrating his commitment to ensuring reliable software systems. He is currently a Distinguished Researcher and a Developer specializing in the fields of machine learning and software testing. With a keen interest in machine learning, software engineering, and software fault tolerance, he has made significant contributions to the field through his research endeavors. He is actively engaged in exploring the efficacy of AI approaches in the realm of cybersecurity. His research interests include immense potential for advancing the field and addressing the evolving challenges of cybersecurity.

AFTAB ALI is currently a Lecturer with the School of Computing, Ulster University, Belfast, U.K. With over a decade of experience in academia, he was a Lecturer and a Researcher at various universities. He has made significant contributions to his field through publications in scholarly international journals and conferences. In addition, his work has led to the filing of seven patents that focus on enhancing software productivity and cybersecurity using AI. He serves as the lead academic on two projects, namely Software Bug Prediction and Future IoT Security, at BT Ireland Innovation Centre (BTIIC). He leads a team of graduate students, supervising both M.Sc. and Ph.D. candidates. He is a Co-I on the BT Ireland Innovation Centre (BTIIC) and the PwC Advanced Engineering and Research Centre (ARC £3.13 million). His expertise is also sought after in the publishing domain, where he serves as a reviewer and a guest editor for several international journals. His research interests include cybersecurity, trust management in the Internet of Things (IoT), blockchains, digital twins, artificial intelligence, and machine learning applications.