# Enhanced Consistency Bi-directional GAN (CBiGAN) for Malware Anomaly Detection

Thesath Wijayasiri, Kar Wai Fok and Vrizlynn L. L. Thing
Cybersecurity Strategic Technology Centre
ST Engineering, Singapore



***Abstract*—Static analysis, a cornerstone technique in cybersecurity, offers a noninvasive method for detecting malware by analyzing dormant software without executing potentially harmful code. However, traditional static analysis often relies on biased or outdated datasets, leading to gaps in detection capabilities against emerging malware threats. To address this, our study focuses on the binary content of files as key features for malware detection. These binary contents are transformed and represented as images, which then serve as inputs to deep learning models. This method takes into account the visual patterns within the binary data, allowing the model to analyze potential malware effectively. This paper introduces the application of the CBiGAN [2] in the domain of malware anomaly detection. Our approach leverages the CBiGAN for its superior latent space mapping capabilities, critical for modeling complex malware patterns by utilizing a reconstruction error-based anomaly detection method. We utilized several datasets including both portable executable (PE) files as well as Object Linking and Embedding (OLE) files. We then evaluated our model against a diverse set of both PE and OLE files, including self-collected malicious executables from 214 malware families. Our findings demonstrate the robustness of this innovative approach, with the CBiGAN achieving high Area Under the Curve (AUC) results with good generalizability, thereby confirming its capability to distinguish between benign and diverse malicious files with reasonably high accuracy.**

***Index Terms*—Cybersecurity, Anomaly Detection, Generative Adversarial Networks, Malware Analysis, Feature Processing**

## I. Introduction

In the ever-evolving cybersecurity landscape, malware detection is a critical frontline defense, protecting systems against unauthorized access and potential harm. Among the techniques deployed, static analysis is particularly vital due to its ability to analyze software without executing the code. This method ensures safety and efficiency, as opposed to dynamic analysis techniques which require the program to be executed in order to analyze behavior, making it indispensable for environments where security cannot be compromised. The key downside being its overall higher consumption of time.

The reliance on biased or outdated datasets has traditionally challenged static malware analysis. These datasets often do not accurately represent the full spectrum of modern malware threats, leading to a gap in detection capabilities against newer or more sophisticated attacks. As malware continues to evolve in complexity and stealth, there is a pressing need for methodologies that not only detect known threats but can also adapt to identify new patterns of anomalies. Malware images created by visualizing binary sequences present an approach that has garnered significant research interest over time. Features used for the static analysis of malware range from entire byte sequences to more purpose driven features such as Operation code (opcode). The conversion of features to visual representations has been a popular method in deep learning solutions since its proposal by Nataraj et al [3]. This method involved transforming the higher dimension malware, to a lower dimension visual representation that can be more effectively processed by machine learning algorithms whilst still retaining all the global features. Global features are features derived from the entire program, providing a broader overview of its behavior or structure. In comparison, local features focus on specific sections such as API calls or specific code sections.

Generative Adversarial Networks (GANs), a model comprising a generator and a discriminator that learn in an adversarial manner, in order to effectively map data to a latent space and then reconstruct it, have been widely used in the field of image generation. GANs have been known for their exceptional ability to model and understand complex data distributions, a feature crucial for malware's complex nature. However, in the realm of malware detection, the potential of GANs extends beyond simple data generation to more sophisticated applications involving anomaly detection. Sabuhi et al. [4] outlines various GAN architectures and correlates them with the types of datasets they are typically employed in. For example, the BiGAN or Bi-directional GAN, which introduces an encoder that learns to encode real samples in the latent space concurrently with the generator, which decodes from the latent space to construct generated samples. Based on the higher successful usage of BiGANs with complex imagery, we selected the consistency Bidirectional GAN (CBiGAN), a variant of the BiGAN, as our foundational model. This decision was further informed by the CBiGAN's [2]demonstrated efficacy with texture-based images, a mean balanced accuracy of 0.836 for the texture dataset in the MVTec Adversarial Dataset, which aligns closely with our primary focus on analyzing the texture and layout of global malware images. This suggests it could be particularly adept at distinguishing subtle anomalies in data.

This paper explores the integration of CBiGANs into static malware analysis, utilizing a novel approach that employs comprehensive malware imaging techniques. Using the above mentioned techniques on both PE and OLE files, our model flags significant anomaly scores as potential malware by training exclusively on benign data.

Our method takes the existing CBiGAN architecture and

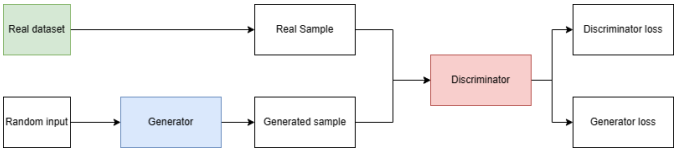

Fig. 1: Generative Adversarial Network

replaces the base encoder with a deep-learning model that would be better suited for the encoding of complex image data such as malware images.

We replaced the base encoder of the CBiGAN with several deep networks(ResNet, DenseNet, Inception), and observed improved performances on our selected datasets.

Our evaluations involve a curated dataset containing 6,330 benign PE files, 10,820 malicious executables, covering 214 families. We also included 9000 benign and 10,980 malicious PDF's from contagio [19]. We further tested the transferability of our model against 9 classes from the Microsoft Malware Challenge dataset. The findings highlight the capability of our approach to achieve a high Area Under the Curve (AUC) score. In this paper, we discuss the existing works such as the CBiGAN and how we have effectively altered it to our specifications, as well as image processing techniques which have not priorly been used for the anomaly detection of malware. We also compare our method with state-of-the-art works, in which our method being a simplified process provides comparable results to said works and shows better generalizability.

## II. Related work

Static analysis has been shown to have more research focus in recent times due to the lower risk, using local feature sets [6] [8] and global feature sets [3] [7]. We leverage

When considering existing work, majority of malware anomaly detection methods tend to lean towards the usage of support vector machines, which might be preferable for simpler or smaller datasets or when there are clear boundaries between normal and abnormal points, however with novel malware techniques this may not be sufficient. A potential solution was the usage of GANs, as they excel in more complex scenarios, where anomalies need to be detected in highly complex patterns.

Initially introduced by Ian Goodfellow [14], Generative Adversarial Networks (GANs) comprise of two key components as shown in figure 1, the generator and the discriminator, engaged in a zero-sum game. The generator generates data from a latent variable (z drawn from a prior distribution p(z)) aiming to produce images realistic enough to deceive the discriminator, while the discriminator's objective is to distinguish between authentic and generated images consistently and accurately.

Through adversarial training, these components iteratively improve their performance. Like an art counterfeiter and an art critic, the generator (counterfeiter) attempts to produce increasingly realistic counterfeits. In contrast, the discriminator (critic) aims to discern authentic artworks from counterfeits. The training process continues until the discriminator can no

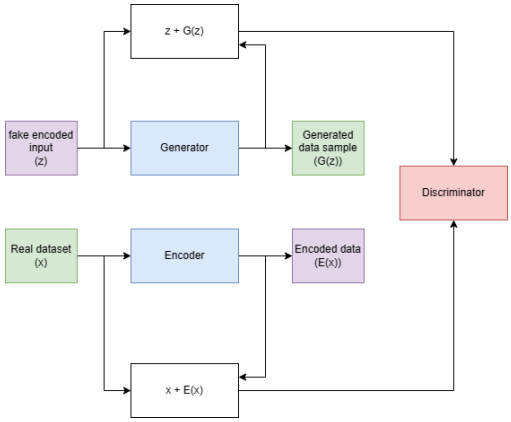

Fig. 2: Bi-directional GAN

longer effectively differentiate between real and fake images. A Generative adversarial network is considered trained once a Nash equilibrium is reached between the generator and the discriminator. In this state, neither party can improve their position unilaterally.

The Bi-directional GAN (BiGAN) [15], extends the GAN by incorporating an encoder as shown in figure 2, which maps the data to the latent space, enabling bidirectional translation between images and latent representations. The generator and the encoder are trained concurrently, with the generator serving as a decoder to reconstruct images from latent space. The discriminator retains its role of distinguishing real from generated images.

The Consistency Bi-GAN (CBiGAN) [2], introduces a consistency constraint as a regularization term in both the encoder and decoder components as shown in figure 3. Unlike traditional GANs focused on generative modeling, CBiGAN is tailored for anomaly detection as a one-class classifier. During inference, CBiGAN computes an anomaly score as a linear combination of pixel-based reconstruction error and feature-based discrimination error. For anomalous images, both the reconstruction error and the discrepancy between discriminator features increase, reflecting the failure of the generator to reconstruct the input faithfully. Significant deviations in the reconstruction of test samples indicate anomalies. This method exploits the model's inability to accurately reconstruct samples that differ from the benign training set, identifying them as anomalies.

The application of space-filling curves, originally proposed by Vu et al. [10], and color manipulations in image processing has been a focal area in various recent research studies, particularly in fields demanding image segmentation and enhancement for analytical purposes. One such study is by Saridou et al. [11]. Space-filling curves, such as the H-indexing technique, maintain the spatial locality of data points in an image when mapped from a high to a lower-dimensional space. This preservation is crucial, ensuring pixel-related anomalies remain detectable after the transformation. Although the method used by Saridou et al. was for classification, exploring space-filling curves and color manipulations in anomaly detection offers significant possibilities in image processing techniques, particularly in fields where identifying deviations from a norm is crucial, such as anomaly detection.

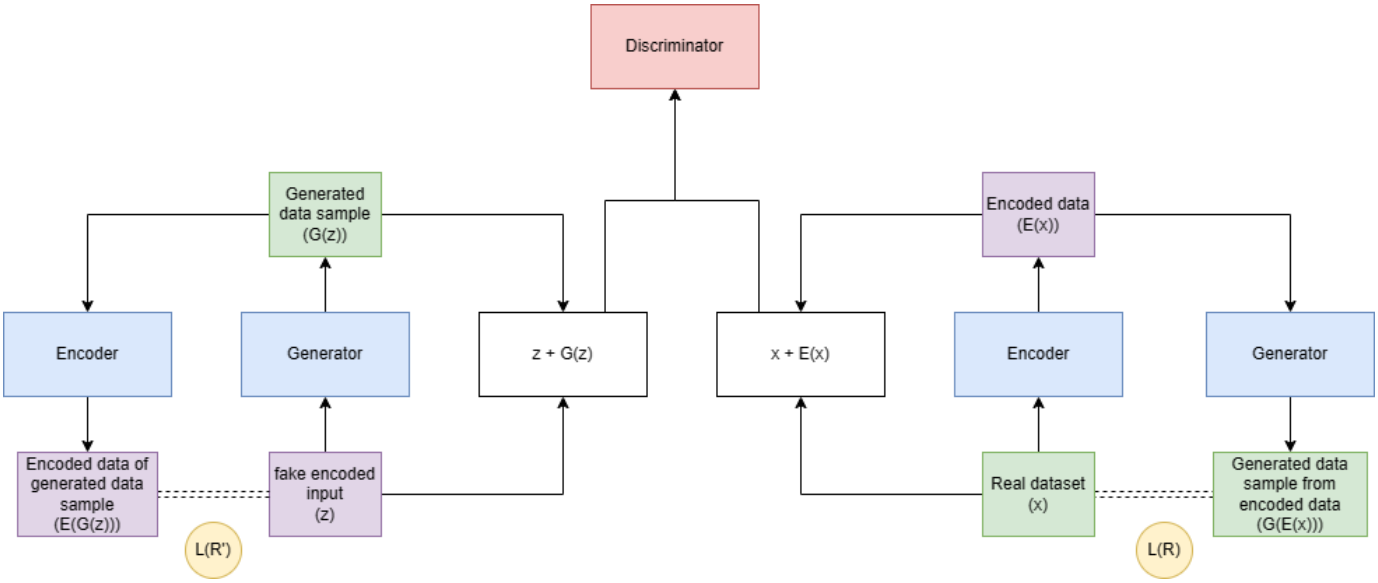


Fig. 3: Consistency Bi-GAN

Once the feature sets were considered, the next step was to identify how these features, namely the byte sequences and opcodes, should be processed before being fed as input to the machine learning models. Reviewing several survey papers, such as Sabuhi et al. [4] on GAN-based anomaly detection methods, it was shown that the most common method used as a dataset was in the form of images.

Malware has also been processed into images as proposed by Nataraj L. et al. [3] in 2011, where they visualize the malware by converting the malware binaries into greyscale images also known as byteplots. Low-level features such as intensity-based or texture-based features were extracted and used as the input for a support vector machine. Nataraj L. et al. observed that images generated by the same malware family often exhibit similar texture and layout, making them suitable for classification tasks. However, traditional methods face challenges when it comes to anomaly detection due to the diverse range of benign and malicious applications, each exhibiting unique visual patterns. To address this, we hypothesized that leveraging color information instead of grayscale, coupled with using space-filling curves as proposed by Vu et al. [10], could enhance the representation of meaningful features within a single image. Space-filling curves, such as the Hilbert curve, offer a computationally efficient approach for retaining locality while reducing dimensionality. It recursively subdivides a square, connecting smaller squares' centers in a continuous curve, thus preserving spatial relationships effectively.

Malware images act as visual representations of features extracted from malicious software programs. They are known for their varied methods of modifying existing code, thus impacting different feature sets. Historically, malware images have been generated by utilizing the entire byte sequence of these programs as a feature set, a precedent set by Nataraj L. et al. [3]. Additionally, Operation Codes (OpCodes) can achieve similar objectives, although this method focuses exclusively on the operational commands within the software.

In malware image anomaly detection, our methodology sets itself apart from prior works, such as Shaukat et al. [13], by adopting a different image processing technique. While Shaukat et al. utilize a pixel mapping method from Nataraj et al [3]. that processes images left to right, line by line, our approach leverages Vu et al.'s [10] technique using space-filling curves, specifically the Hilbert curve. This method preserves locality better, enhancing our ability to detect smaller variations in image anomalies. Notably, most current studies in this area utilize the MalIMG dataset [3]. While this dataset has been instrumental in advancing the field, it does not adequately reflect the evolving nature of malware threats. Since 2011, significant developments in malware technology have occurred, with new variants emerging that exhibit increasingly sophisticated characteristics designed to evade detection. These changes often result in malware samples that are less distinguishable from benign software, a challenge that our approach seeks to address by leveraging more current data and less intrusive preprocessing to potentially enhance detection accuracy in today's more complex threat landscape.

## III. Proposed Methodology

Our research adopted a methodology centered primarily on analyzing global features while minimizing the pre-processing steps involved. This approach was chosen to explore a more streamlined and practical processing pipeline, potentially suitable for deployment on local devices with limited computational resources. By focusing on global features, we aim to harness broad contextual information, which may prove crucial for the robustness and accuracy of our models, especially in real-world scenarios. Reducing pre-processing not only contributes to faster execution times but also decreases the complexity and potential for errors. This strategy is particularly relevant as the demand increases for real-time applications in mobile and edge computing environments, where efficiency and speed are paramount for detection.

Figure 4 presents the overall methodology regarding the PE file-to-image conversion. This same process is repeated for OLE files as well. The file is converted to an array of hexadecimal values, which then go through the image processing step to get converted into our final image. Figure 5 shows the image processing step, wherein the array of hexadecimal values are rearranged following the Hilbert transformation and then mapped to RGB vectors which are then used to create the final image. These images are then used to train our model which is a variation of the CBiGAN proposed by Carrera et al. [2] with a secondary deep learning model substituting for the encoder of the model.

Our model acts as a one-class classifier, using images from benign applications as the training data. The model would then learn to classify images made from malicious files as anomalous due to the higher reconstruction error.

### A. Datasets

*1) Primary Dataset:* The primary experiment employs a self-collected dataset consisting of 6,330 benign Portable Executable (PE) files. These files were sourced using a webcrawler and subsequently verified as benign through VirusTotal checks. The benign data was split with a 60/20/20 ratio, with 60% of the data being used for training: 3798 samples, 20% each used for testing and validation: 1266 samples each. Complementing this, the dataset for malicious files comprises 10,820 samples belonging to 214 families, sourced from MalwareBazaar [16]. MalwareBazaar is a public repository of malicious samples. A script was used to download PE samples with the .exe extension based on the upload date from the latest. The sample set was then further refined by limiting the size between 2MB to 50MB to reflect the size range in the benign data. The malicious samples were split evenly for testing and validation with 5,410 samples each. Furthermore, to assess the transferability and generalizability of our trained model, we utilize the Microsoft Malware Challenge Dataset [17]. The Microsoft dataset is a curated dataset from 2015 containing 10868 malicious samples from 9 key families. This dataset is instrumental in testing how well our model, trained on PE file types and malware families, adapts to a broader and previously unseen set of malware samples.

*2) Extended Primary dataset:* We further test the efficacy of increased benign dataset size and variation by adding more benign samples to the primary dataset, from Michael Lester's curated PE dataset [18]. We decided to extend only the training set of the primary dataset, in order to keep other variables constant, as the focus on this experiment was to test the increase in size and variation of the training set. Therefore, we extended the 3798 benign training samples of the primary dataset to form a training set of 15,000 benign samples for training. Michael Lester's curated dataset consists of 86,812 benign samples and 114,737 malicious samples. For our experiments we did not use malicious samples from this dataset and only selected 11,202 benign samples which were split from the dataset with a seed of 42 and combined with the training samples from the primary experiment to form the new training samples.

*3) Secondary Dataset (Contagio):* In a sub-experiment, we focus on Object Linking and Embedding (OLE) files, particularly PDFs, using the Contagio dataset. This includes 9,000 benign OLE files and 10,980 malicious OLE files, allowing us to test the model's effectiveness in a different binary context and evaluate its performance across varying file types.

### B. Conversion of Raw Files to Features

This involves identifying features to extract from OLE and PE files as the first step. Although our datasets are composed of PE and OLE files, this method is transferable to any type of file. Several literature surveys were considered, all within the past five years [7] [8] [9] [10], to look into which features were used most in malware anomaly detection and how those features were processed. These were then correlated to what datasets GANs used for anomaly detection and the format of those datasets (i.e., images, 1D arrays, etc.). The first step was converting the files to images. The process could essentially be broken down into two key steps. One was converting the file to a set of features selected through the literature surveys [5], such as bytecode and Op-code. The second step was converting these features into an image. Although the process itself was broken down into two steps, the two were interdependent.

### C. Feature Selection and Extraction

*1) Byte Sequences and Opcode Identification:* The primary features targeted for extraction were byte sequences and opcodes, which offer critical insights into the file's executable behavior. Byte sequences represent the raw hexadecimal data of the PE files. At the same time, opcodes (operation codes) indicate the instructions executed by the processor.

*2) Automation with Linux Terminal Commands :* To efficiently handle the vast quantity of PE files, we automated the extraction process using Linux terminal commands integrated within a Python script. We utilized the xdd command for byte sequences to generate hex dumps of the files. For extracting opcodes, the objdump command was employed, coupled with a custom Python script tailored to isolate just the opcodes from the output.

### D. Conversion of Extracted Features into Images

Integration of Features: The extracted features, namely the byte sequences and opcodes, were then integrated to form a coherent data set suitable for image conversion. This step is critical as it translates textual or numerical data into a visual format that image-based anomaly detection systems can process. Feature to image The conversion of features to images can also be considered to be comprised of two key facets: pixel color and pixel mapping. Images are made up of individual pixels. Each of these pixels are represented by 3 RGB values ranging from 0-255, for example a white pixel would have the RGB values 255,255,255. These pixels are then arranged on a grid of predetermined size. The way they are arranged would determine how the image finally looks and is referred to here as pixel mapping. Per the literature review, the most used

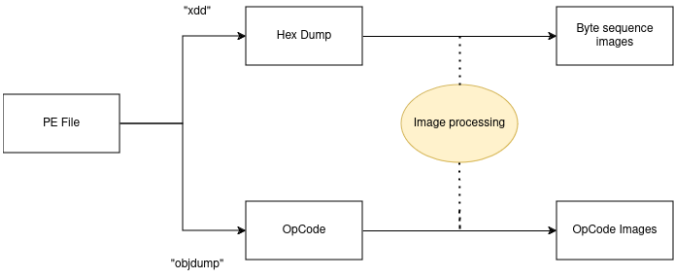


Fig. 4: PE to image conversion process

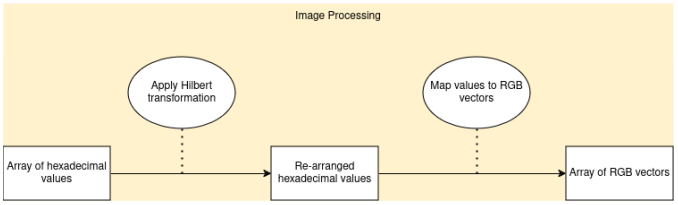


Fig. 5: Image Conversion

method was presented with the malware visualization paper proposed by Nataraj L. et al., which consists of converting the PE files into a series of binary values grouped into sets of 8 or a byte. This was then converted into a decimal value between 0 and 255, which would relate to a color on the greyscale. This method was used to create each pixel. The pixels were mapped from left to right until a set width was met, and then they continued from the next row. This method created images with a fixed width but variable length.

Vu et al. [10] proposed that this method did not capture local features well enough and suggested a different method, the usage of space-filling curves. Space-filling curves are a Hamiltonian path widely used in computer image generation for converting one-dimensional arrays to two-dimensional and vice versa while keeping the local features grouped. Saridou et al. [1] showed that in classification methods, this image processing method proved superior to the generic method proposed in Nataraj et al. [3] Therefore, to address the local feature retention issues for mapping that may arise when converting PE files to images, space-filling curves were considered, namely the Hilbert curve. The Hilbert curve is simple, with a repetitive shape change with each increment.

Furthermore, with the reconstruction error-based classification of our model, we integrated a novel approach combining geometrically distant colors with the 2D locality-preserving properties of the Hilbert curve. This integration aims to refine the color differentiation process, thus improving the accuracy of the model in classifying the instances based on their reconstruction characteristics.

*1) Pixel color allocation:* We made a strategic decision to standardize the color allocation to colors of high RGB value contrast. While Saridou et al. [11] proposed a color scheme for enhancing image representation, we took this concept further by standardizing colors based on geometric contrast. In our methodology, we envisioned the color space as represented by Red (R), Green (G), and Blue (B) values, each constrained to the interval [0, 256]. We conceptualized the RGB value as a vector in this color space, confined within a cube, and

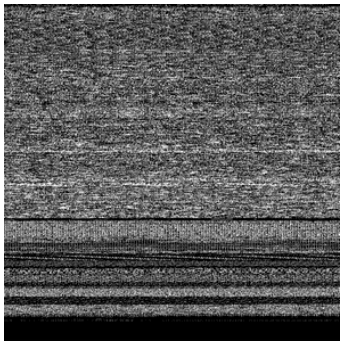

Fig. 6: Malware sample converted using Nataraj et al. method

sought to identify the 16 most contrasting values for RGB values within that space. Geometrically, this contrast is defined as the points farthest away from the given RGB value while still residing on the boundaries of the cube. By employing such standardized colors with maximal contrast, we aimed to enhance the discriminative power of our image representations, thereby improving anomaly detection performance. The conversion of hexadecimal values to RGB vectors is mentioned as follows in Table 1.

| Hexadecimal Character | R value | G value | B value |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 128 | 0 | 0 |
| 2 | 154 | 99 | 36 |
| 3 | 128 | 128 | 0 |
| 4 | 70 | 153 | 144 |
| 5 | 0 | 0 | 117 |
| 6 | 230 | 25 | 75 |
| 7 | 245 | 130 | 49 |
| 8 | 255 | 225 | 25 |
| 9 | 191 | 239 | 69 |
| A | 60 | 180 | 75 |
| B | 66 | 212 | 244 |
| C | 67 | 99 | 216 |
| D | 145 | 30 | 180 |
| E | 240 | 50 | 230 |
| F | 255 | 255 | 255 |

TABLE I: Standardized color allocation to hexadecimal chart

*2) pixel mapping:* Once the pixel colors were decided, they need to be mapped onto a grid in order to form an image. As stated before there are several methods such as Nataraj.et al. [3] which used a left to right mapping, wherein once the end of a row was reached, it would then begin mapping again from the next row in a similar fashion and that that was proposed by Vu et al. [10] which was the utilization of space filling curves such as the Hilbert curve. We follow the method proposed by Vu et al. due to the ability for space filling curves to better retain locality when being converted from 1d arrays to 2d arrays. This tends to better represent local patterns of the files as shown in the comparison between figure 6 and figure 7. We used python script to apply the Hilbert transformation taking approximately 6 seconds per 50MB sample

### *E. Model*

Our method incorporates a variation of the CBiGAN proposed by Carrera et al. [2] As stated before, the CBiGAN is a variation of the BiGAN [15] wherein a consistency constraint

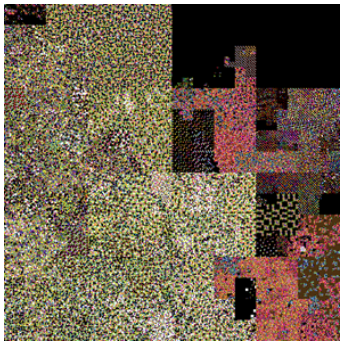

Fig. 7: Malware sample converted using our method

is introduced. As shown in figure 3, this ensures that when a real data sample and its corresponding latent representation (generated by the encoder) are fed back into the Generator, the output (reconstructed data) should closely resemble the original real data sample. Similarly, when a random latent vector (drawn from the latent space distribution) is used to generate synthetic data through the generator, and this synthetic data is encoded back to the latent space by the encoder, the resulting latent representation should be similar to the original random latent vector. This is implemented by adding the Wasserstein loss function (presented as L(R) in figure 3) to the training process that penalizes discrepancies between the original and reconstructed components. The CBiGAN had no prior history of processing images as complex as those generated from malware, raising concerns about its encoder's ability to map these images to the latent space effectively. Traditionally, most malware image detection methods have utilized CNN-type encoders, such as ResNet, configured as autoencoders. Yumoto et al. [5] demonstrated that combining GANs with CNN-type encoders could surpass the performance of autoencoders in detecting anomalies in complex images. Consequently, we replaced the CBiGAN's encoder with several CNN-type encoders ranging from lighter models such as IncpetionV3 to denser models such as DenseNet201. Namely ResNet50, ResNet101, ResNET152, DenseNet169 and InceptionV3. By doing so, we hope to explore the trade-offs between computational efficiency and anomaly detection accuracy. Lighter models, such as InceptionV3, offer faster processing times and lower computational costs, making them suitable for real-time applications. On the other hand, denser models like DenseNet201 provide higher accuracy in detecting subtle anomalies at the expense of increased computational requirements. This aims to determine the optimal balance between performance and efficiency.

## IV. Experimental setup

Our experiments were conducted using a Linux system on a system utilizing an intel i9-11900K processor and an NVIDIA RTX 3090 graphics card with 24GB of VRAM as well as 128GB of system RAM.

### A. Metrics

We utilized balanced accuracy and AUC (Area Under the Curve) for evaluation metrics. These metrics were chosen as they provide a more comprehensive view of model performance across imbalanced datasets than standard accuracy or F1 metrics. Balanced Accuracy (BalAcc) is calculated as follows:

$$\text{BalAcc} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Where; TP denotes True Positives, FN denotes False Negatives, FP denotes False Positives, and TN denotes True Negatives.

Balanced accuracy compensates for any bias introduced by the disproportionate ratio of classes by averaging the proportion of correct predictions in each class independently. AUC, on the other hand, measures the ability of the model to discriminate between classes at various threshold settings, making it a robust indicator of model effectiveness across different levels of classification threshold. The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) is a performance measurement for classification problems. It represents the degree or measure of separability. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

The AUC is calculated as the area under the ROC curve, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR})\, d(\text{FPR})$$

## V. Initial experiments

In the early stages of our experiment, we explored different feature extraction techniques to determine the most effective method for converting binary files into image representations for our CBiGAN model. Two primary methods were considered: the byte sequence method and the opcode method. These methods were evaluated based on their performance in distinguishing between benign and malicious software samples from our self-collected dataset. As stated before, the models were all trained on 3798 benign samples and tested and validated against 1266 benign samples and 5410 malicious samples each.

### A. Evaluation of byte sequence method vs Opcode method

*1) Byte Sequence Method:* This approach involves converting the binary files into images based on the raw byte sequences. The resulting images visually represent the structural patterns of the binaries, which are then used as input for a CBiGAN-ResNet50 model. The byte sequence method demonstrated a significant advantage in our experiments, achieving an Area Under the Curve (AUC) score of 0.816.

*2) Opcode Method:* Alternatively, the opcode method converts binary files into images based on the operation codes extracted from the executable files. This method aimed to capture the semantic information of the binaries more explicitly than the byte sequence approach. However, in our tests, the opcode method yielded an AUC of only 0.616, which is considerably lower than that of the byte sequence method. This result suggests that while the opcode method provides valuable insights into the behavioral patterns of the binaries, it is less

effective for the visual-based anomaly detection method used in our study.

The initial testing conclusively showed that the byte sequence method was substantially more effective for our purposes. This finding guided our decision to adopt the byte sequence approach for the detailed experiments and evaluations conducted in this study.

### *B. Evaluation of the Hilbert curve and RGB colour mapping*

The images created by applying the Hilbert curve and the standardized color allocation method were initially hypothesized to offer improvements. To validate these assumptions, we conducted further testing using identical training and testing data and the same GAN-encoder configuration to evaluate the differences in outcomes. For testing we used the contagio dataset as it provided a more focused scope for the initial testing. Given the structured nature of PDFs and the availability of comprehensive metadata within the Contagio dataset, initial testing on PDFs provides a more controlled environment. This focused scope helps in fine-tuning our conversion methods before applying them to the more diverse and unstructured PE files As indicated in Table 2 which shows a comparison of the results using identical datasets with different image processing approaches, this method significantly enhances results with a 0.795 AUC score compared to the 0.590 AUC score of the approach introduced by Nataraj L. et al. [3] in 2011, which remains widely adopted in current practices.

### *C. Evaluation of the encoder replacement for the CBiGAN*

One of the key hypotheses made in our works is the increased accuracy of a deep learning model substituted CBiGAN as opposed to a CBiGAN with its base encoder We conducted an initial test to evaluate this hypothesis by comparing experiments that used the same configurations and datasets, with the only difference being the encoder used. For this test, we switched to the primary dataset of PE files as the primary focus was testing on PE files. Table 3 shows the AUC scores for the trained models against the test datasets as well as the validation datasets. The test columns show the best result during training against the test dataset, which consequently is used to save the best model. The valid. columns show the validation results of the trained model against the validation dataset. the test results show both higher Area Under the ROC Curve and balanced accuracy results for the experiments where the deep learning model was substituted for the base encoder present in the CBiGAN with a 22.3% increase in AUC for the ResNet50 encoder as opposed to the base encoder. A potential hypothesis as to why the ResNet50 model outperformed the deeper models would be due to the way our model was designed to work. As the result would be based off of reconstruction errors, the unique color combinations would be easier to identify using a lighter model as there would be too many features if the entire image was considered. This however doesn't excuse the inceptionV3 model having poor results. A secondary hypothesis would be that the while Inception models focus on capturing multi-scale features using multiple convolutional filters of different sizes within the same layer, this complexity can sometimes lead to suboptimal performance if not tuned precisely. However, this would require further testing to identify.

## VI. Experiments

Our study aimed to evaluate the impact of varying encoder complexities, from lighter to deeper models, on the initial model performance, including validation accuracy and the model's generalizability across unseen malware classes through zero-shot testing. This comprehensive testing strategy allows us to understand how different levels of model depth influence our system's immediate effectiveness and broader applicability in identifying diverse malware threats. We assess the model's ability to generalize beyond its training dataset by conducting zero-shot tests, where we test the model against previously unseen classes of malware such as the Microsoft Malware Challenge dataset [17], which is crucial for cybersecurity applications with new malware variants. The results show that the deeper models tended to have better overall results, albeit with a longer training time. Our methodology involves training the model using the benign dataset, with periodic evaluations against a test set of malware to assess both balanced accuracy and the best AUC. Throughout the training process, the model is configured to save two checkpoints: one for the checkpoint that achieves the highest AUC, updated each time an improved AUC is recorded, and another for the final trained model after completing all training iterations. This approach ensures that we retain the model that performs best in terms of area under the curve and the model at the end of the training cycle for further analysis and comparison. Our methodology also incorporates an Exponential Moving Average (EMA) to smooth out the fluctuations in the training metrics. Using EMA helps stabilize the learning updates by averaging the model parameters over time, which can lead to more robust generalization performance. This approach is beneficial for mitigating the effects of erratic changes in model performance due to the nature of the training process, ensuring that our evaluations of AUC and balanced accuracy reflect a more consistent and reliable estimate of the model's capabilities.

### *A. Primary Experiment*

Our primary experiment was based on PE files utilizing the aforementioned primary dataset which is a self-collected dataset of benign and malicious PE files. We chose to include as many varying malware families as possible to get a more generalized result which should translate to better transferability of the model.

### *B. Dataset Size Experiment*

This experiment was to see the effect of increased benign samples in the training data that would allow the model to generalize better and bring in more variation to the benign samples as well. We tested this hypothesis by using the extended primary dataset mentioned in section 4.1.2 supplementing our existing dataset using benign samples from Michael

| Method | AUC | Balanced Accuracy |
|---|---|---|
| Traditional method [3] | 0.59 | 0.627 |
| Greyscale + Hilbert | 0.786 | 0.734 |
| **Our method (RGB + Hilbert)** | **0.795** | **0.782** |

TABLE II: comparison of image processing methods between Nataraj et al. and ours using the contagio dataset

| Encoder model used | Test AUC | Test Bal. Acc. | Valid. AUC | Valid. Bal. Acc. |
|---|---|---|---|---|
| CBiGAN base encoder | 0.667 | 0.662 | 0.661 | 0.659 |
| **ResNet50** | **0.816** | **0.781** | **0.818** | **0.787** |
| ResNet101 | 0.766 | 0.731 | 0.768 | 0.730 |
| ResNet152 | 0.791 | 0.737 | 0.778 | 0.724 |
| DenseNet169 | 0.801 | 0.730 | 0.806 | 0.736 |
| InceptionV3 | 0.833 | 0.762 | 0.620 | 0.599 |

TABLE III: Test and Validation results using different encoders with against the self-collected malware dataset of 214 classes

| Training Dataset Size | Test AUC | Test Bal. Acc. | Valid. AUC | Valid. Bal. Acc. |
|---|---|---|---|---|
| 3,798 | 0.816 | 0.781 | 0.818 | 0.787 |
| **15,000** | **0.864** | **0.831** | **0.857** | **0.822** |

TABLE IV: Test and Validation results using different training dataset sizes for the CBiGAN - ResNet50 combination

Lester's curated dataset [18]. The final training set was 15,000 benign samples. We tested using the model configuration with ResNet50 encoder and all hyperparameters the same as our main experiments.

### C. Transferability Experiment

This experiment is to test the transferability of the model resulting from the primary experiment, by testing it against 9 previously unseen malware classes from the Microsoft dataset.

### D. Secondary Experiment

Our secondary experiment was based on using OLE files, specifically PDF files, sourced from the contagio dataset. This evaluates our methods effectiveness in a different binary context.

## VII. Results

### A. Primary Experiment Results

The comparative analysis of the performance across ResNet50, ResNet152, and DenseNet169 models as Shown in table 3 of section 7 provides a nuanced view of the effectiveness of different network architectures in our malware detection framework. While ResNet50 achieves a higher AUC in both test and validation sets and reaches its performance peaks earlier in the training process, this does not necessarily translate to superior generalizability across all scenarios, notably, the ResNet models reach their peak AUC considerably earlier than the DenseNet model as observable in figure 8 which shows the comparison of AUC over time for the ResNet50 an DenseNet169 models. Initially, this might suggest a preference for ResNet models; however, a deeper examination of model transferability reveals a different perspective.

ResNet50's earlier peaking in AUC might indicate faster convergence, potentially due to its shallower architecture compared to DenseNet169. This can be advantageous in scenarios

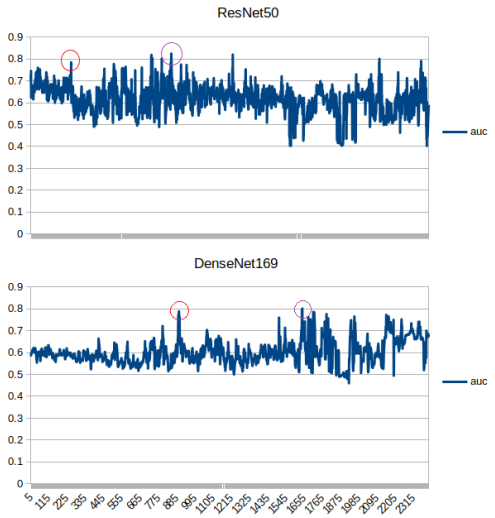


Fig. 8: Comparison of AUC over time for ResNet50 and DenseNet169 encoders

where quick deployment of updated models is crucial. However, the lower peak AUCs of ResNet152 and DenseNet169 suggest these models may capture more complex patterns that are not immediately evident early in training.

### B. Dataset Size Experiment Results

Table 4 shows the results of training an identical model to the CBiGAN-ResNet50 model, which showed the best results in the prior experiment, with an increased training dataset. We tested by increasing the dataset size from 3,798 to 15,000 by supplementing with benign samples from Michael Lester's PE machine learning dataset [18]. Samples were selected based on a seed of 42. As we can see there is a positive correlation in increasing the dataset size and variation of samples in this context with an improved AUC of 0.864 as opposed to the 0.816 of the smaller training set. This passes on to the validation set as well with an AUC of 0.857 as opposed to 0.818 of the smaller training set.

### C. Transferability Experiment Results

Table 5. further provides a critical insight into the transferability of the models, by testing the models trained on our self-collected dataset against the 9 classes of the Microsoft malware challenge dataset [17]. Despite their slower ascent to peak performance, both ResNet152 and DenseNet169 demonstrate superior capability in generalizing to the Microsoft malware classification challenge dataset, including various malware types unseen during training. This indicates that the depth of the model plays a reasonable role in learning generalized features rather than fitting them to the specificities of the training data.

| Name | Best AUC | MS1 - ramnit | MS2 - Lollipop | MS3 – kelihos_ver3 | MS4 - Vundo | MS5 - Simda | MS6 - Tracur | MS7 – Kelihos_ver1 | MS8 – Obfuscator.ACY | MS9 - Gatak |
|---|---|---|---|---|---|---|---|---|---|---|
| CBiGAN-D169 | 0.801 | 0.555 | 0.448 | 0.252 | 0.353 | **0.771** | 0.456 | 0.916 | 0.478 | 0.568 |
| CBiGAN – R152 | 0.791 | 0.629 | **0.493** | 0.149 | 0.279 | 0.097 | 0.41 | 0.887 | 0.823 | 0.539 |
| CBiGAN - R50 | 0.816 | 0.603 | 0.419 | **0.357** | 0.218 | 0.328 | **0.513** | 0.852 | 0.842 | 0.396 |
| CBiGAN - R50 (larger training set) | 0.864 | **0.683** | 0.413 | 0.314 | **0.724** | 0.618 | 0.475 | **0.925** | **0.886** | **0.665** |

TABLE V: Transferability of model to unseen data

| Model Used | AUC |
|---|---|
| **ResNet50** | **0.837** |
| ResNet101 | 0.74 |
| ResNet152 | 0.767 |
| DenseNet169 | 0.772 |
| InceptionV3 | 0.766 |

TABLE VI: Results using different encoders against Contagio dataset

The ability of deeper models to perform better on a comprehensive challenge such as the Microsoft dataset, despite lower AUCs initially, underscores the importance of architectural depth for complex image recognition tasks like malware detection. While ResNet50 may be efficient for quicker tasks and performs well on known test and validation sets, models like ResNet152 and DenseNet169 are more robust when dealing with diverse and previously unseen malware classes. Therefore, while ResNet50 provides a good balance between performance and speed, for applications where robustness against diverse threats is critical, deeper networks might offer more reliable protection, albeit at the cost of increased computational resources and training time.

However, the most influential factor for transferability results was the increased dataset size. One could conclude that the increased variation in the training set would affect the transferability results positively. Although the results were not great through each of the 9 classes in the Microsoft dataset, we do see a clear increase in the results of the transferability test, with the increased dataset size test showing the highest result for 5 of 9 classes tested on. It also shows a relatively positive result for MS5 - Simda, which for the other ResNet results did not show good results. We can also see a correlation in the encoder model used and results for certain classes, such as for MS8 - Obfuscator.ACY showing subpar results for the DenseNet experiment and ¿0.8 AUC results for the ResNet models.

### *D. Secondary Experiment Results*

For the secondary experiment we tested our model against the Contagio PDF dataset [19]. Table 6 shows the results of using different encoder variants with the Contagio dataset, with the ResNet50 encoder showing the best results by far with 0.837 AUC score. We also test the usage of limiting the PDF files to only ones containing JavaScript components to see the effect on the AUC score, based on the findings of Gu et al. [12]. For this we follow the same method as Gu et al. by using PeePDF to separate the samples with JavaScript components.

| Model Used | Feature Reduction Step | AUC |
|---|---|---|
| OSVM + RegNetY320 | No | 0.89 |
| OSVM + RegNetY320 | Yes | 0.921 |
| OSVM + ResNet152 | No | 0.84 |
| OSVM + ResNet152 | Yes | 0.87 |
| OSVM + VGG19 | No | 0.81 |
| OSVM + VGG19 | Yes | 0.891 |

TABLE VII: Results of shaukat et al.

## VIII. Analysis

### *A. Comparison with Shaukat et al. [13]*

Our principal comparison for the PE dataset is with the model proposed by Shaukat et al., which employs a one-class SVM trained solely on benign PE files converted to RGB images using a method adapted from Nataraj et al. Shaukat et al.'s study is one of the few utilizing malware images for anomaly detection rather than classification. They test their initial model against a combination of the Microsoft malware challenge dataset and VirusShare. They show their results for their model alone as well as with an extra feature extraction process using PCA and a deep learning model. Table 7 is an extract from shaukat et al.'s paper which shows their key results which include the combination of an OSVM with three different deep learning models used for feature extraction as well as well as results for using each with and without the feature reduction step

Our CBiGAN-ResNet50 model demonstrates comparable results with an AUC of 0.816 on the PE dataset which contains over 214 malware classes as opposed to the 9 classes used by shaukat et al. (they do not specify how many samples were taken off virus share so it is hard to approximate), using a simpler and more integrated approach than the multi-step method employed by Shaukat et al. Furthermore, their transferability tests on the MalIMG dataset, which include different data types (RGB for their method and greyscale for MalIMG), are not considered comparable in our analysis due to the discrepancies in data representation. Furthermore, we tested their feature reduction technique with our method as well, which showed an increase of the CBiGAN-ResNet50 models AUC to 0.837 however that negatively affected both the transfer-ability results (reduced AUC for Microsoft's 5,7 and 8 classes) which further polarizes us from their transferability findings.

### *B. Comparison with Gu et al. [12]*

Our analysis also compares our methodology against the JavaScript-based detection approach proposed by Gu et al.,

which employs a unique strategy focusing on JavaScript code within PDF documents. We propose a comparison with the work of Gu et al. as they utilized the same publicly available OLE dataset [19]. The limited number of research studies employing publicly available OLE datasets restricts the scope of comparative analyses that can be conducted. Gu et al. use PJscan and peepdf parsers to extract and tokenize JavaScript from PDF files, applying a one-class SVM classifier to differentiate between benign and malicious documents using the Contagio OLE dataset—a dataset we also utilize in our experiments.

Key findings from Gu et al. reveal that 97.51% of the malicious PDF files in their dataset contain JavaScript, compared to only 4.26% of benign files. Their approach achieved an impressive detection accuracy of 96.9% with a relatively low false positive rate of 3.2%. Our results fall short of theirs with our experiment only yielding a 0.837 AUC with the ResNet50 encoder. We further test by limiting our dataset to the one used by Gu et al. by using PeePDF to isolate the PDFs that have JavaScript components. However, this only yielded an increased AUC result of 0.855, which was only a marginal increase over the prior result.

While their results are notable, it's important to consider the practical limitations of their method:

The dependency on JavaScript presence may not generalize well across all types of PDF-based malware, particularly as malware creators may shift away from using JavaScript due to increased detection. The method requires the use of specific parsing tools, which introduces potential vulnerabilities, particularly if malware obfuscation techniques interfere with the parsers' ability to detect JavaScript code. In contrast, our CBiGAN-based approach does not rely on the presence of specific features such as JavaScript within the files. This broader approach avoids the limitations associated with parsers and does not confine the detection capability to specific malware characteristics. Furthermore, our method has demonstrated robustness against obfuscation techniques, achieving an AUC of 82.3% in detecting the obfuscater.acy malware class during our transferability tests.

## IX. Conclusion

In this study, we introduced a novel approach for malware detection using the consistency Bi-directional Generative Adversarial Network (CBiGAN) combined with deep learning models such as ResNet and DenseNet, specifically tailored for anomaly detection. Our approach used visual representations of binary content to effectively capture intricate patterns within malware binaries, employing deep learning to enhance detection capabilities.

Our experimental results showed that the CBiGAN achieved strong predictive performance and generalizability across diverse datasets including PE and OLE files. This method's efficacy is highlighted by its ability to handle a diverse set of malicious executables from 214 malware families with reasonable accuracy. Our CBiGAN-based method offers a streamlined, single-model approach that contrasts with the multiple, separate steps required in other studies. This simplicity is advantageous for practical applications, where ease of deployment and maintenance are critical. Additionally, our method's ability to detect sophisticated obfuscation techniques in malware, such as those used in the obfuscater.acy malware class, underscores its effectiveness and advanced capability in handling real-world malware threats.

## References


[1] Saridou, B., Moulas, I., Shiaeles, S., & Papadopoulos, B. K. (2023). Image-Based malware detection using Ă-Cuts and binary visualisation. Applied Sciences, 13(7), 4624. https://doi.org/10.3390/app13074624

[2] Carrara, F., Amato, G., Brombin, L., Falchi, F., & Gennaro, C. (2021, January 10). Combining GANs and AutoEncoders for efficient anomaly detection. International Conference on Pattern Recognition. https://doi.org/10.1109/icpr48806.2021.9412253

[3] Nataraj, K., Jacob, G., & Manjunath, B. S. (2011). Malware images: visualization and automatic classification. Proceedings of the 8th International Symposium on Visualization for Cyber Security.

[4] Sabuhi, M., Zhou, M., Bezemer, C., & Musilek, P. (2021). Applications of Generative Adversarial Networks in anomaly Detection: A Systematic Literature review. IEEE Access, 9, 161003–161029. https://doi.org/10.1109/access.2021.3131949

[5] Yumoto, S., Kitsukawa, T., Moro, A., Pathak, S., Nakamura, T., & Umeda, K. (2023). Anomaly detection from images in pipes using GAN. ROBOMECH Journal, 10(1). https://doi.org/10.1186/s40648-023-00246-y

[6] Wu, Q., Zhu, X., & Liu, B. (2021). A survey of Android malware static detection technology based on machine learning. Journal of Mobile Information Systems, 2021, 1–18. https://doi.org/10.1155/2021/8896013

[7] Ngo, Q., Nguyen, H., Le, V., & Nguyen, D. (2020). A survey of IoT malware and detection methods based on static features. ICT Express, 6(4), 280–286. https://doi.org/10.1016/j.icte.2020.04.005

[8] Sihwail, R., Omar, K., & Ariffin, K. A. Z. (2018). A survey on malware analysis techniques: static, dynamic, hybrid and memory analysis. International Journal on Advanced Science, Engineering and Information Technology, 8(4–2), 1662. https://doi.org/10.18517/ijaseit.8.4-2.6827

[9] Pan, Y., Ge, X., Fang, C., & Yi, F. (2020). A Systematic Literature Review of Android Malware Detection using Static Analysis. IEEE Access, 8, 116363–116379. https://doi.org/10.1109/access.2020.3002842

[10] Vu, D., Nguyen, T., Nguyen, T. V., Nguyen, T. N., Massacci, F., & Phung, P. H. (2019). HIT4Mal: Hybrid image transformation for malware classification. Transactions on Emerging Telecommunications Technologies, 31(11). https://doi.org/10.1002/ett.3789

[11] Saridou, B., Rose, J. R., Shiaeles, S., & Papadopoulos, B. (2022). SAGMAD–A signature agnostic malware detection system based on binary visualisation and fuzzy sets. Electronics, 11(7), 1044. https://doi.org/10.3390/electronics11071044

[12] Gu, J., Kong, R., Sun, H., Zhuang, H., Pan, F., & Lin, Z. (2023). A novel detection technique based on benign samples and one-class algorithm for malicious PDF documents containing JavaScript. International Conference on Computer Application and Information Security. https://doi.org/10.1117/12.2637518

[13] Shaukat, K., Luo, S., & Varadharajan, V. (2024). A novel machine learning approach for detecting first-time-appeared malware. Engineering Applications of Artificial Intelligence, 131, 107801. https://doi.org/10.1016/j.engappai.2023.107801

[14] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. https://arxiv.org/abs/1406.2661

[15] Donahue, J., Krähenbühl, P., & Darrell, T. (2016). Adversarial feature learning. arXiv (Cornell University). https://arxiv.org/pdf/1605.09782

[16] MalwareBazaar - Malware sample exchange. (n.d.). https://bazaar.abuse.ch/

[17] Microsoft Malware Classification Challenge (BIG 2015) — Kaggle. (n.d.). https://www.kaggle.com/c/malware-classification

[18] Lester, M. (2021, June 8). PE Malware Machine Learning Dataset. Practical Security Analytics LLC. https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/

[19] Mila. (2013, March 16). 16,800 clean and 11,960 malicious files for signature testing and research. https://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html