2022

Few-Shot Malware Detection Using A Novel Adversarial Reprogramming Model

Ekula Praveen Kumar
Wright State University

FEW-SHOT MALWARE DETECTION USING A NOVEL ADVERSARIAL REPROGRAMMING MODEL

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Cyber Security

by

EKULA PRAVEEN KUMAR
B.Tech., National Institute of Science and Technology, 2017

2022
Wright State University

Wright State University
GRADUATE SCHOOL

December 20, 2022

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Ekula Praveen Kumar ENTITLED Few-Shot Malware Detection Using A Novel Adversarial Reprogramming Model BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Cyber Security.

Dr. Lingwei Chen, Ph.D.
Thesis Director

Thomas Wischgoll, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Dr. Lingwei Chen, Ph.D.

Dr. Tanvi Banerjee, Ph.D.

Dr. Junjie Zhang, Ph.D.

Dr. Shu Schiller, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Kumar, Ekula Praveen. M.S.C.S., Department of Computer Science and Engineering, Wright State University, 2022. *Few-Shot Malware Detection Using A Novel Adversarial Reprogramming Model.*

The increasing sophistication of malware has made detecting and defending against new strains a major challenge for cybersecurity. One promising approach to this problem is using machine learning techniques that extract representative features and train classification models to detect malware in an early stage. However, training such machine learning-based malware detection models represents a significant challenge that requires a large number of high-quality labeled data samples while it is very costly to obtain them in real-world scenarios. In other words, training machine learning models for malware detection requires the capability to learn from only a few labeled examples. To address this challenge, in this thesis, we propose a novel adversarial reprogramming model for few-shot malware detection. Our model is based on the idea to re-purpose high-performance ImageNet classification model to perform malware detection using the features of malicious and benign files. We first embed the features of software files and a small perturbation to a host image chosen randomly from ImageNet, and then create an image dataset to train and test the model; after that, the model transforms the output into malware and benign classes. We evaluate the effectiveness of our model on a dataset of real-world malware and show that it significantly outperforms baseline few-shot learning methods. Additionally, we evaluate the impact of different pre-trained models, different data sizes, and different parameter values. Overall, our results suggest that the proposed adversarial reprogramming model is a promising direction for improving few-shot malware detection.

Contents

List of Figures

List of Tables

Acknowledgment

I would like to thank my thesis advisor, Dr. Lingwei Chen, for his guidance, support, and encouragement throughout the entire process. His expertise and dedication to my academic development have been instrumental in shaping my research and helping me to reach this point. I would also like to thank my committee members, Dr. Tanvi Banerjee and Dr. Junjie Zhang for taking the time to evaluate my thesis; your discussion, ideas, and feedback have been absolutely invaluable. Finally, I would like to thank my family and loved ones for their unwavering support and encouragement. Their love and belief in me have sustained me through the highs and lows of this process and allowed me to persevere to this point. Thank you all again for your invaluable contributions to my thesis and my academic journey.

Introduction

In this thesis, we would like to propose a machine learning solution to detect malware from a more practical yet challenging perspective where the labeled data samples are limited, which is called few-shot malware detection. In this chapter, we first describe the background of malware and its detection, and our motivation for the proposed solution. After that, we specify our contribution and the organization of this thesis.

1.1 Background

Malware, short for malicious software, is any software that is designed to harm or exploit computer systems [24]. There are many different types of malware including viruses, worms, trojan horses, ransomware, and spyware. According to a recent report, which is illustrated in Figure 1.1, the total amount of malware as per recent findings surpassed 1.2 billion in 2022 [3]. The threat of malware is constantly evolving as criminals come up with new ways to gain access to systems and steal data. As more businesses move their operations to the cloud, the importance of protecting their data has never been greater. By identifying and removing malicious software before it can cause damage, you reduce the risk of your computer or network is compromised and the data within it being lost. This is the reason why malware detection is an important part of any security strategy. Different malware detection systems have been constantly designed and developed to ensure they

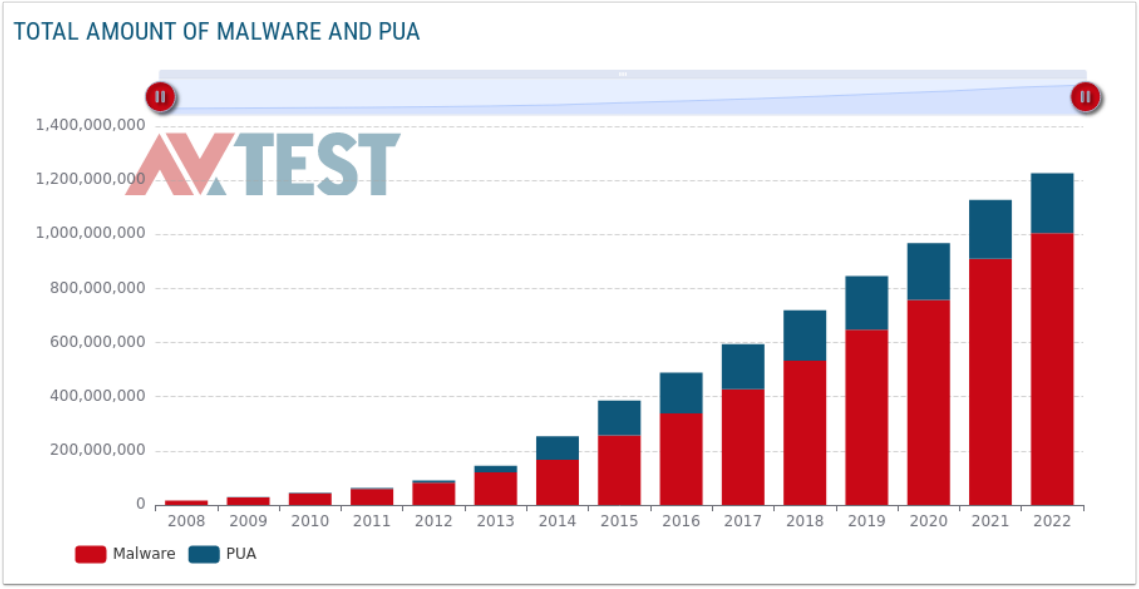

Figure 1.1: Statistics of malware over the years [3]

can detect threats and provide businesses with protection [5].

With the rapid development in machine learning, and especially the revolutionary learning structures and capabilities raised by deep learning [29], in recent years, the security industry and researchers have developed different these data-driven algorithms and frameworks into practical products and applications for more accurate malware detection, which improve the protection of computer systems against attacks. Here, we discuss some of the advantages of using machine learning to detect malware.

One of the first advantages of machine learning in malware detection is that it provides accuracy in identifying new and unknown threats. Unlike traditional malware detection methods that rely on pre-defined rules to identify known threats, machine learning algorithms can recognize previously unseen malware patterns. By analyzing large numbers of samples obtained from real malware infections, machine learning models can learn which features of a file or program are most indicative of malicious behavior [30]. Once trained, these models can then be used to automatically analyze new samples and detect previously unknown threats.

Another advantage of using machine learning for malware detection is that it has the

potential to scale to large datasets. Since machine learning algorithms are based on algorithms that can learn from data, they can incorporate features from a large set of malicious samples and train a model to identify similar features in new samples. Thus, they can handle large quantities of data and can effectively detect a wide range of malicious files without the need for extensive manual analysis [19, 42]. For example, deep learning models can produce highly accurate results even for complicated datasets containing thousands of features. This makes them particularly suitable for addressing large-scale malware detection and classification problems.

Utilizing machine learning for detecting malware can also enhance system security with minimal human input, providing an additional benefit. As machine learning models are trained on real-world samples rather than on rule-based specifications, they can detect previously unknown threats based on features extracted from actual malware code.

Even though using machine learning for malware detection has a lot of advantages, there are downsides to using machine learning for malware detection. The most significant one of them is the requirement for labeled data samples. Labeled data is the heart of machine learning. The data that is fed into the machine learning model should fit the algorithm and needs to be sufficiently large for the model to train correctly [53], such that the model can capture the feature patterns and accordingly improve its learning and detection ability. Unfortunately, collecting such a large number of high-quality labeled malware samples is extremely difficult, and labeling the malware samples is also costly [34, 19, 5].

1.2 Motivation

Based on the above discussions, we can summarize the challenge to use machine learning for malware detection is that we need to collect a large number of labeled data samples to facilitate model training. However, different from more straightforward annotations on

less complex data such as images, labeling malware data calls for solid domain knowledge and elaborate verification, which is time-consuming and labor-intensive [49, 12]; thus, it is often costly to acquire sufficient labeled malware data for detection model training. In other words, our built machine learning model for malware detection needs to have the ability to learn from a few samples. To address the data-limited learning challenge, various paradigms have been accordingly proposed, where meta learning [17, 59, 63] and transfer learning [41, 40] are some major solutions.

`Meta Learning`: Meta learning, also known as "learning to learn," is a type of machine learning that involves learning how to learn new tasks quickly and efficiently [22]. It is based on the idea that a learning system can improve its learning ability by learning from past learning experiences. In meta learning, a model is trained on a set of related tasks, to learn how to learn new tasks within the same domain more quickly and effectively. The model is then tested on a new, unseen task and evaluated on its ability to learn and adapt to this task. Meta learning has been applied to a variety of domains, including natural language processing, computer vision, and reinforcement learning [23]. It has the potential to enable machine learning systems to learn more efficiently and adapt more quickly to new environments, making it an important area of research in the field of artificial intelligence.

However, meta learning has several limitations. Meta learning algorithms can be complex and computationally intensive, which can limit their practicality for certain applications. It is often sensitive to the quality of the training data, and may not perform well if the training data is noisy or biased [57]. These algorithms may struggle to generalize to tasks that are significantly different from those seen during training, due to a lack of sufficient inductive bias. The most significant limitation to meta learning is that the classes used for meta-training and meta-testing need to be disjoint, but the data are generally collected from the similar problem domain [25], which, however, is infeasible in few-shot malware detection, since annotations for any malware class are difficult and it is hard to collect such a distribution of malware detection tasks to learn a shared initialization.

`Transfer Learning`: Transfer learning is a machine learning technique in which a model trained on one task is re-purposed on a second related task. It involves the transfer of knowledge from a source task to a target task, to improve the performance of the model on the target task [39], which is useful in situations where it is difficult to collect sufficient data for training a model on a new task, or when there is a need to use a pre-trained model as a starting point for a new task. By using transfer learning, it is possible to leverage the knowledge gained from a related task to improve the performance of a model on a new task, without the need to start from scratch. There are several ways to perform transfer learning, including fine-tuning a pre-trained model, using a pre-trained model as a feature extractor, or using a pre-trained model as a fixed feature extractor [52]. As such, transfer learning has been applied to a variety of domains, including natural language processing [43], computer vision [20], and speech recognition [46].

However, transfer learning also has certain limitations. First, transfer learning is only effective when the source and target domains are similar enough that knowledge learned in the source domain applies to the target domain. If the domains are too different, transfer learning may not be effective. Second, it can be sensitive to the choice of the source domain and the model pre-trained on that domain, and may not always achieve the best performance compared to training a model from scratch on the target domain [52]. Third, it still requires a large amount of labeled data in the source domain, which may not always be available or may be difficult to obtain. In this way, transfer learning may not be a good solution to address few-shot malware detection either, since we do not have the large labeled samples for the model fine-tuning.

1.3 Contribution

Similar to transfer learning, adversarial reprogramming is a machine learning technique that involves altering the behavior of a machine learning model by making small, carefully

chosen changes to the input data that the model processes. These changes, known as adversarial perturbations, are designed to cause the model to produce a desired output, even if the original input data would have resulted in a different output [13]. That is, adversarial reprogramming can achieve the same objective as transfer learning that re-purposes a machine learning model pre-trained in a source domain to perform a target-domain task. By contrast, adversarial reprogramming yields an additional advantage that transfer learning does not have: adversarial reprogramming only learns a universal perturbation to be added to the input data, but the pre-trained model, with respect to structure and parameters, keeps unchanged. In this respect, adversarial reprogramming needs much less labeled data samples for model training and deals much better with few-shot machine learning application scenarios [10, 11].

With this in mind, to mitigate the limitations of meta learning and transfer learning, in this thesis, we propose to use adversarial reprogramming to deal with small labeled data for malware detection. More specifically, we present a novel adversarial reprogramming approach to detect malware in a few-shot setting, where we have only a small number of examples of malware and benign files to work with.

Our approach is to reprogram an ImageNet classification neural network to perform few-shot malware detection. Specifically, the model first obtains a set of features from data samples, injects these features into a host image randomly selected from ImageNet, and then adds a universal perturbation to the host image to create a set of new image inputs to represent the original data samples. After that, the adversarial reprogramming model will randomly select two ImageNet classes to indicate malware and benign classes for malware detection. At the end of model formulation, an optimization problem is constructed between the predictions and true labels to minimize the loss function to calculate the optimal perturbation. The reasons why we choose ImageNet classification neural networks in our proposed adversarial reprogramming model are that these pre-trained models have very deep and non-linear structures, which can cope better with feature learning and im-

prove the malware detection capability, and these neural networks are easily feasible. We evaluate our approach with different ImageNet classification neural networks and different parameter settings and compare it with different baseline models to demonstrate that it can outperform state-of-the-art few-shot learning methods for malware detection. We can summarize our contributions as follows:

- We deal with few-shot malware detection with small labeled data instead of regular data settings with a large number of labeled data.
- We propose a novel adversarial reprogramming idea to implement this few-shot malware detection model.
- We evaluate our approach with different ImageNet classification neural networks and different parameter settings, and compare with different baseline models.

1.4 Organization

In this thesis, section 1 refers to the background, motivation, and contribution to the thesis. In section 2, I explained the related works for few-shot malware detection that have been proposed and the introduction of adversarial reprogramming. In section 3, I explained more about adversarial reprogramming technically, how it works, and the difference between adversarial reprogramming and other few-shot models. In section 4, I explained how I implemented the model and the parameters with other settings. Then I evaluated the model with different pre-trained ImageNet classification models and with different train data sizes and different parameter values and compared it with the baseline models; random forest and neural network and transfer learning.

Literature Review

In this chapter, we would like to introduce the related works, including machine learning-based malware detection models, few-shot learning models, and few-shot malware detection models. Based on this literature review, we can better understand the challenges of malware detection and the differences between our proposed few-shot malware detection from the current works.

2.1 Malware Detection

Malware detection is the process of identifying and mitigating the presence of malware on a computer system or network. This can be done through a variety of means, including the use of antivirus software, manual inspection of system files and processes, and the use of machine learning algorithms [21]. In this section, we would like to present different machine learning detection methods.

2.1.1 Signature-based Malware detection

Signature-based methods are generally employed in anti-malware software products from different security companies to provide the major protections against malware attacks [16]. A signature is a specific pattern or characteristic that is unique to a particular type of malware [5]. We can easily find some research papers that use signature-based methods for

malware detection. Deepak et al. performed malware detection on mobile devices using a signature-based malware detection method [56]. Abbas et al. built a low complexity signature-based method for IoT devices that only identifies and stores a subset of signatures to detect a group of malware instead of storing a separate signature for every potential malware [1]. Savenko et al. proposed a method for signature generation of malware based on API call tracing [45]. But the limitation of signature-based malware detection lies in the fact that they are effective at detecting known malware, but they can be less effective at detecting new or unknown types of malware; also, these models can be easily get evaded by malware attacks using more sophisticated techniques, such as obfuscation, and also a large amount of malware generated every data.

2.1.2 Behavior-based Malware detection

To address the limitation of signature-based malware detection, another approach is proposed to use behavior-based information, which involves analyzing the behavior of a program or process to determine whether it is exhibiting characteristics that are indicative of malware [5]. Burguera et al. built a malware detection model using dynamic analysis of application behaviors in android [7], Liu et al. implemented a model using Malware Behavior Feature (MBF) based malware detection algorithm [31]. This approach can be more effective at detecting new or unknown types of malware, but it can also be more resource-intensive and may produce more false positives.

2.1.3 Machine learning-based Malware detection

In recent years, machine learning has become a popular approach for detecting malware, as it allows models to learn and adapt to new types of malware over time. The advantages of machine learning algorithms are that they can analyze large datasets of known malware and benign software to identify patterns and features that are indicative of malware, and

can then be used to classify new, unseen software as either benign or malicious.

There has been a significant amount of research using machine-learning techniques for malware detection. One common approach is to use traditional machine learning algorithms, such as decision trees [67], support vector machines [28], and ensemble learning [64], to classify executables as benign or malicious based on static features extracted from the executables. These static features may include characteristics of the executable file itself, such as its size, entropy, and the presence of certain strings or patterns [42, 15]. For example, Zulkifli et al. implemented a model based on network traffic using decision tree algorithm [67]. Zhao et al. proposed a behavior-based malware detection model using support vector machine algorithm [65]. Azzez et al. presented a malware detection method based on ensemble learning. A stacked ensemble of fully-connected, one-dimensional convolutional neural networks (CNNs) performs the base-stage classification, while a machine learning algorithm performs the end-stage classification [4].

In recent years, there has been a growing trend toward using deep learning techniques for malware detection. These techniques, such as convolutional neural networks (CNNs) [55] and long short-term memory (LSTM) [62] networks, are able to learn more complex patterns in the data and have achieved state-of-the-art results on many tasks. Deep learning approaches for malware detection often use dynamic features extracted from the execution of the executable, such as the system calls made by the executable or the network traffic generated by the executable [19, 2]. For example, Ganesh et al. suggested a technique, that utilizes a convolutional neural network to evaluate authorization patterns [18]. Based on behavioral data, McDole et al. carried out malware detection using process-level performance metrics, such as consumption of CPU, memory, and disk with convolutional neural network (CNN). [33]. Vinayakumar et al. used the long short-term memory (LSTM) architecture for Android malware detection [58].

Overall, machine learning can be a powerful tool for detecting malware, but it is important to carefully consider the type of machine learning approach and the dataset being

used, as these factors can significantly impact the accuracy and effectiveness of the model. There are several limitations to using machine learning for malware detection [24], which are summarized as follows:

- Need for large, high-quality datasets. Machine learning algorithms require large datasets of known malware and benign software in order to learn and classify new, unseen software. If the dataset is small or of low quality, the model may not be able to accurately classify new software.

- Vulnerability to adversarial examples. Machine learning models can be vulnerable to adversarial examples, which are intentionally misclassified examples designed to trick the model into making incorrect predictions. This can be a concern in the context of malware detection, as attackers may try to create adversarial examples of malware in order to evade detection by the model.

- Need for ongoing training and evaluation. Machine learning models require ongoing training and evaluation to maintain their accuracy and effectiveness. If the model is not regularly updated and tested, it may become less effective at detecting malware over time.

- Limited ability to detect new or unknown types of malware. Machine learning algorithms are generally more effective at detecting known types of malware, as they are trained on specific patterns and features that are indicative of malware. However, they may be less effective at detecting new or unknown types of malware, as they have not been trained on these types of malware and may not recognize them as malicious.

In this thesis, we mainly focus on solving the limitation of machine learning-based malware detection with limited labeled data samples, and investigating how we can build a few-shot malware detection model to improve the detection performance.

2.2 Few-Shot Learning

As we would like to build a few-shot malware detection model, in this section, we list some major few-shot learning techniques to facilitate understanding data limitation and constructing our few-shot malware detection model. Few-shot learning is a machine learning technique that involves learning to perform a new task using only a small number of examples. This is an important problem in the context of not just malware detection but also any other cyber security attack detection, as it is often costly or time-consuming to obtain large amounts of labeled data for these types of detection tasks [57]. Some major few-shot learning methods with their own unique set of benefits and drawbacks are introduced in detail as follows.

2.2.1 Meta Learning

Meta learning is a type of machine learning that involves learning how to learn. In the context of malware detection, a meta-learning approach could involve training a model to learn how to classify new, previously unseen examples of malware by adapting its learning strategy based on the examples it has seen so far [23, 66]. There are several approaches to perform meta learning, including:

- `Model-based meta learning`: This approach involves training a neural network to learn a task by learning a model of the task. The model is then used to make predictions about the task, which can be used to improve the model's performance [50].

- `Optimization-based meta learning`: This approach involves training a model to optimize its own parameters, based on its experience with previous tasks. The model learns to optimize its own learning process, rather than learning the task directly [22, 60].

- `Memory-based meta learning`: This approach involves training a model to use its memory of previous tasks to learn new tasks more efficiently. The model stores information about past tasks in its memory and uses this information to guide its learning process for new tasks [60].

Meta learning has a wide range of potential applications, including natural language processing [27], computer vision [35], and robotics [26]. However, like any machine learning approach, it has certain limitations that should be considered. Some of the limitations of meta learning include [22]:

- Data requirements: Meta learning often requires a large amount of data in order to learn effectively. This can be a challenge in situations where it is difficult to collect sufficient data for training a model on a specific task.

- Limited generalization: Meta learning algorithms are typically designed to work well on a specific set of related tasks, but may not generalize well to tasks outside of this domain.

- Sensitivity to hyper-parameters: Meta learning algorithms can be sensitive to the choice of hyper-parameters, which can make it difficult to fine-tune the model for a specific task.

2.2.2 Transfer Learning

Transfer learning is a type of machine learning that involves using knowledge learned from one task to improve the performance of a model on a related task [39, 44]. The goal of transfer learning is to leverage the knowledge and experience gained from solving one problem to improve the performance of a model on a different, but related problem. For example, imagine that you have trained a deep-learning model to classify images of dogs and cats. If you wanted to use this model to classify images of birds, you could use transfer learning

by fine-tuning the model on a dataset of bird images. This would allow you to leverage the knowledge the model has already gained about image recognition and classification and fine-tune it to perform the specific task of bird classification [52].

Transfer learning can be a powerful tool in machine learning because it allows you to build on the work of others and take advantage of pre-trained models that have already been trained on large, high-quality datasets. This can save time and resources, and can often lead to better performance than training a model from scratch. There are several approaches to perform transfer learning, including fine-tuning [39], feature extraction [52], and multi-task learning [44]. Each approach involves using a pre-trained model as a starting point and adjusting it to perform a new task, with the specific approach depending on the details of the new task and the availability of data.

However, it has certain limitations that should be considered. Some of the limitations of transfer learning include [52]:

- Limited to related tasks: Transfer learning is most effective when the source and target tasks are closely related. If the tasks are too dissimilar, the knowledge learned from the source task may not be useful for the target task, resulting in poor performance.

- Lack of generalization: Transfer learning algorithms are typically designed to work well on a specific set of related tasks, but may not generalize well to tasks outside of this domain.

- Dependence on the quality of the pre-trained model: The performance of a transfer learning model is dependent on the quality of the pre-trained model. If the pre-trained model is not well-suited for the target task, the performance of the transfer learning model may be poor.

2.2.3 Other Techniques

In addition to meta-learning and transfer learning, there have also been efforts to design more effective feature representations and similarity measures for few-shot learning. These approaches aim to capture the important characteristics of input samples and measure the similarity between different types of samples in a way that is robust to variations in the input data [53].

2.3 Few-shot Malware Detection

Some approaches have been designed and developed to address few-shot malware detection using the aforementioned techniques, including transfer learning [44], and meta learning [27]. Transfer learning involves fine-tuning a pre-trained machine learning model on a malware detection task using a small amount of labeled data. Meta learning involves training a machine learning model to adapt to new malware detection tasks using a small amount of labeled data [53]. For example, Martin et al. proposed a model using meta-information from the Android Manifest to analyze and find fraudulent Android applications; its major goal is to offer a quick but accurate tool that can help users prevent infecting their devices without even needing to install the application to carry out the examination [32]. To more accurately capture the characteristics of the studied segments, Tahan et al. implemented the model employing a novel feature type called a meta-feature [51]. Bhodia et al. dealt with the issue of malware categorization and detection using image analysis. Bhodia et al. used deep learning (DL) models to apply image recognition after converting executable files to images [6]. Chen et al. proposed a model using computer vision's deep transfer learning to categorize static malware and apply knowledge from naturally occurring images or objects to the target domain of static malware detection in the transfer learning scheme [9].

Design of Few-Shot Malware Detection

Few-shot malware detection is to train a machine learning model to classify new, unseen types of malware based on a small number of examples, or "shots" [8]. As traditional machine learning approaches may require a larger dataset for model training [54], this approach is particularly useful in situations where there are few examples of a particular type of malware available for training [21]. In this chapter, we are using adversarial reprogramming to build a few-shot malware detection model. As the adversarial reprogramming technique uses machine learning vulnerability as a fundamental for the design to transfer the input with perturbation and change the output that leads to re-purposing the model for other tasks, we first explain adversarial vulnerability more technically and then dive into the technical steps in detail for our design for few-shot malware detection.

3.1 Adversarial Vulnerability

Adversarial vulnerability in machine learning refers to the susceptibility of a machine learning model to being deceived or fooled by malicious or unintended inputs [48]. These inputs, called adversarial examples, have been specifically crafted to cause the model to make mistakes or incorrect predictions, which can be in the form of images, texts, or other types of data and designed to be similar to normal, legitimate inputs but with subtle differences that can be difficult for humans to detect [14].

One example of such an adversarial vulnerability is that an attacker crafts an image

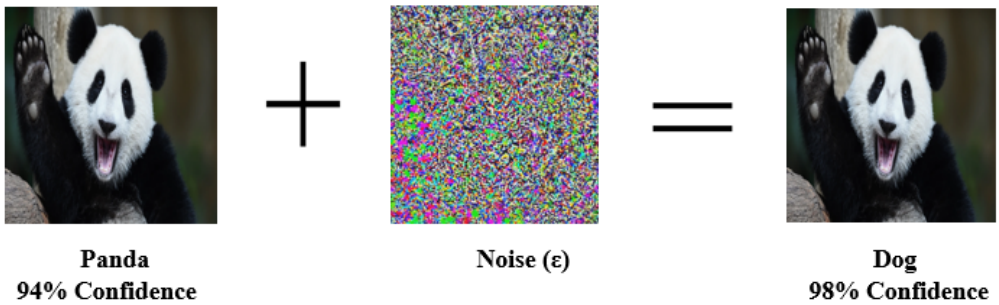

Figure 3.1: An example of adversarial vulnerability

that is intentionally designed to mislead a machine learning model trained for image classification. For instance, consider a model that has been trained to recognize animals in images [38]. As illustrated in Figure 3.1, an attacker could generate an adversarial image of a panda in an easy way that the model incorrectly classifies it as a dog. This could be done, for example, by adding small, imperceptible perturbations to the image that are specifically designed to fool the model [48].

Different from crafting perturbation for individual input, the universal perturbation is a small change added to all inputs that can cause a classifier to misclassify most inputs, which is generally drawn from a given distribution [36]. The goal is to find a perturbation vector v that satisfies two constraints:

- It has a small norm (as measured by the p-norm, with $p \in [1, \infty)$).

$$||v||_p <= \epsilon \tag{3.1}$$

- It causes the classifier to misclassify at least 1-δ fraction of inputs drawn from the distribution.

where the parameters ϵ and δ are used to control the magnitude of the perturbation vector and the desired fooling rate, respectively [36, 48].

Adversarial vulnerability is a concern in machine learning because it can potentially

lead to serious consequences, such as incorrect medical diagnoses or financial fraud. Therefore, it is important for machine learning researchers and practitioners to consider these vulnerabilities when developing and deploying machine learning models. However, these vulnerabilities are also good opportunities to be applied for social good. The focus of this thesis is one of such cases, which attempts to craft a universal perturbation to be added to all inputs that can change the behaviors of the pre-trained neural networks, and accordingly reprograms their functions to perform malware detection.

3.2 Adversarial Reprogramming

Inspired by adversarial vulnerability, adversarial reprogramming is proposed to re-purpose a pre-trained model in a source domain to perform a new task in a target domain by adding universal perturbation into the inputs [37]. Here, we reprogram an ImageNet classification neural network to address few-shot malware detection problem. In more detail, the model first extracts a collection of features from data samples, adds these characteristics to a host image drawn at random from ImageNet, and then adds a universal perturbation to the host image to produce a set of new image inputs that reflect the original data samples. Next, two ImageNet classes will be randomly chosen using the adversarial reprogramming model to represent malware and benign classes for malware detection. The best perturbation is determined by minimizing the loss function after the model construction process, which involves creating an optimization problem between the predictions and true labels. The advantages yielded by adversarial reprogramming are [11]:

- Only the perturbation is trainable and requires less training time and labeled data than transfer learning and other models that are created from scratch.
- The ImageNet model can be used for a range of malware detection tasks without modifying the model's structure or computations because software features must always be converted to images due to the model's fixed input format.

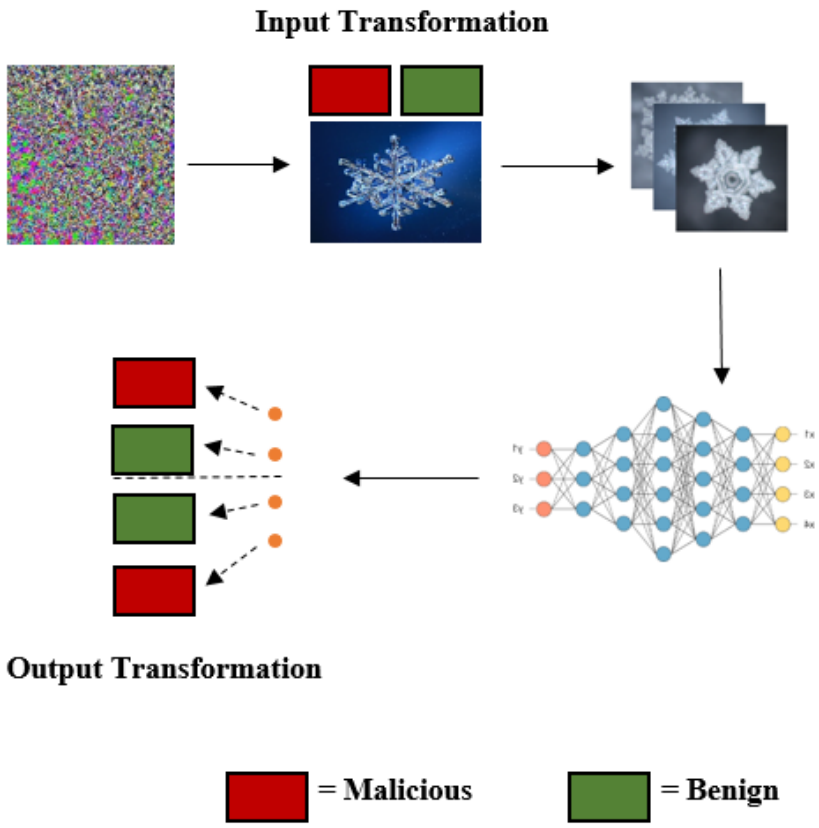

Figure 3.2: Overview of the designed adversarial reprogramming model

- With the help of pre-trained weights and layers on ImageNet neural networks, it is possible to identify expressive patterns from subtle software features and perform well at malware detection.

Based on the aforementioned advantages of adversarial reprogramming, to deal with machine learning-based malware detection with few labeled data samples, we build a novel adversarial reprogramming model based on a pre-trained ImageNet classification neural network, which is illustrated in Figure 3.2. Overall, our adversarial reprogramming model proceeds with three steps: (1) input transformation, (2) output transformation, and (3) optimization. This leads to three major goals:

- Input transformation to embed the software features with the perturbation into the host image to create new image data.
- Output transformation to map the ImageNet classes to malware detection classes.
- Optimization to obtain the optimal perturbation.

We detail all these three major steps as follows.

3.2.1 Input Transformation

To perform input transformation, there were two important steps. One is software feature embedding and the other is applying perturbation. For feature embedding, I assigned features of my dataset to a Numpy array and embedded them to randomly selected pixels of the host image and stored the index values of the pixels so that it can generate a whole new image dataset for training and testing. After feature embedding, I added a universal perturbation function to all the inputs. I also introduced a parameter epsilon ϵ, to analyze the magnitude of the perturbation [13, 10]. The perturbation function is formulated to be added to all inputs, which can be defined as:

$$\tilde{\theta} = \epsilon \cdot tanh(\theta \odot M) \tag{3.2}$$

where $\theta \in \mathrm{R}^{n \times n \times 3}$ refers to perturbation, M is the matrix of mask values to specify the pixels used to store the software features, $\odot$ refers to element-wise product, $\tanh(\cdot)$ sets the bounds of the perturbation to be in $(-1,1)$. To control the amount of the perturbation, we introduce an adjustable hyper-parameter ϵ. The transformed new input data sample can thus be presented as a transformation function $h_s(x;\theta)$ defined as follows:

$$h_s(x;\theta) = \tilde{X} = clip(X + \epsilon \cdot tanh(\theta \odot M)) \tag{3.3}$$

where X is input data in a format of host image with feature embedding x, clip $(\cdot)$ limits the values of each pixel in the image by clipping them to a specified range $(0,1)$ and $\tilde{X}$ represents the new image input after input transformation. In our scenario, both ϵ and clip $(\cdot)$ result in the perturbation imperceptibility.

3.2.2 Output Transformation

The goal of the output transformation process is to map the output classes of the ImageNet classification neural network back to the classes used for malware detection (i.e., benign and malicious). To do this, we implement a hard-coded mapping to perform this operation for detection [47].

More specifically, the hard-coded mapping involves simply assigning two randomly-selected class outputs out of 1,000 ImageNet classes to predict benign and malicious classes, respectively. This is done by using a function $h_t(\cdot)$ that takes an output $\tilde{y}$ of the ImageNet model and maps it to a pair of classes y_i and y_j, where i is not equal to j.

$$h_t(\tilde{y}) = \langle \tilde{y}_i, \tilde{y}_j \rangle, i \neq j. \tag{3.4}$$

Overall, the output transformation process is an important step in adversarial reprogramming, as it allows the results of the ImageNet model to be translated into the classes used for malware detection [12].

3.2.3 Optimization

To develop a cost-efficient method for detecting malware, the pre-trained ImageNet classification neural network needs to be self-deployed for complete access, such that no additional expenses on large model queries will be incurred and we can perform gradient-based method to optimize the adversarial reprogramming model and obtain the ideal perturbation.

In this respect, the optimization of our adversarial reprogramming model is formulated as an optimization problem where the goal is to minimize a loss function that takes into account the difference between the probability that an input image, transformed from software, will be classified as malware and its ground-truth label, where a regularization term is added to help prevent overfitting [11]. The adversarial reprogramming model optimization can be formally written out as:

$$\theta^* = \text{argmin } (-\log(p(h_t(\tilde{y})|h_s(x;\theta))) + \lambda||\theta||2_F) \tag{3.5}$$

where λ is a regularization parameter, and $p(h_t(\tilde{y})|h_s(x;\theta))$ shows the likelihood that an image input X that was altered by features x will be categorized as $\tilde{y}$, which can be mapped to malware. The single parameter of the problem, θ, is updated using Adam, an optimization algorithm that makes use of first and second-moment estimates of the gradient to adapt the learning rate. The optimization problem described in this subsection is a key component of the adversarial model, which is critical for achieving the goal of cost-efficient malware detection using self-deployed ImageNet classification neural networks.

Experimental Results and Analysis

In this chapter, we evaluate the effectiveness of our proposed few-shot malware detection model over a real malware dataset. First, we introduce the dataset in detail and then specify the experimental settings. After that, we evaluate the impacts of different pre-trained models, parameters, and data sizes, and also compare with some baselines to demonstrate the advantages yielded by our adversarial reprogramming model.

4.1 Experiment Setup

4.1.1 Dataset

For this experiment, I used the MalData dataset from the GitHub open source repository[1]. This dataset consists of 138,047 entries, with 41,323 benign files and 96,724 malware files. The data statistics are illustrated in Table 4.1. Each file in the dataset is characterized by 56 features, including the number of resources, the size of initialized data, the size of uninitialized data, the major and minor linker versions, the MD5 hash, the size of code, the machine type, the size of the optional header, and the characteristics. These features provide a wealth of information about each file, allowing for thorough analysis and comparison. Considering that the feature scales are very different, we normalize these feature values before embedding them into the host image. The MalData dataset is a valuable

[1] https://github.com/PacktPublishing/Mastering-Machine-Learning-for-Penetration-Testing

Table 4.1: Statistics of the dataset used in this thesis

Dataset	No. of Entries	No. of Benigns	No. of Malwares	No. of Features
MalData	138,047	41,323	96,724	56

resource for researchers studying malware and seeking to develop effective methods for detecting and mitigating its effects.

The dataset includes the following features, where the detailed feature value examples are given in Table 4.2:

- `Name` is the name of the file.
- `md5` is the MD5 hash of the file, which is a unique identification code that can be used to verify the integrity of the file.
- `Machine` is the machine type the file is intended to run on, such as x86 or x64.
- `SizeOfOptionalHeader` is the size of the optional header in the file, which contains information about the file's layout and execution.
- `Characteristics` is a set of flags indicating the characteristics of the file, such as whether it is a dynamic-link library (DLL) or whether it is intended to be run on a system with high entropic content.
- `MajorLinkerVersion` and `MinorLinkerVersion` are the major and minor version numbers of the linker used to create the file. The linker is a program that combines object files and libraries to create an executable file.
- `SizeOfCode` is the size of the code section in the file, which contains the instructions that the processor will execute.
- `SizeOfInitializedData` is the size of the initialized data section in the file, which contains data that is initialized when the file is loaded into memory.

Table 4.2: Features of files from dataset

	0	2
Name	mshtml.dll	VirusShare_cf329dab9d0e2b9129b27771a2862394
md5	d20f7eea01f00190902462c0b69ec6c8	cf329dab9d0e2b9129b27771a2862394
Machine	34404	332
SizeOfOptionalHeader	240	224
Characteristics	8226	783
MajorLinkerVersion	9	2
MinorLinkerVersion	0	56
SizeOfCode	6752256	29184
SizeOfInitializedData	2310656	14848
SizeOfUninitializedData	0	110592
AddressOfEntryPoint	21520	14764
BaseOfCode	4096	4096
BaseOfData	0	36864
ImageBase	8793907789824.0	4194304.0
SectionAlignment	4096	4096
FileAlignment	512	512
MajorOperatingSystemVersion	6	4
MinorOperatingSystemVersion	1	0
MajorImageVersion	6	6
MinorImageVersion	1	0
MajorSubsystemVersion	5	4
MinorSubsystemVersion	2	0
SizeOfImage	9076736	221184
SizeOfHeaders	1536	1024
CheckSum	9120035	0
Subsystem	2	2
DllCharacteristics	64	32768
SizeOfStackReserve	262144	2097152
SizeOfStackCommit	4096	4096
SizeOfHeapReserve	1048576	1048576
SizeOfHeapCommit	4096	4096
LoaderFlags	0	0
NumberOfRvaAndSizes	16	16
SectionsNb	6	7
SectionsMeanEntropy	5.41745943392	3.34528001443
SectionsMinEntropy	2.76177448394	0.0
SectionsMaxEntropy	6.76452138512	6.30323831549
SectionsMeanRawsize	1509290.66667	8118.85714286
SectionsMinRawsize	53760	0
SectionMaxRawsize	6752256	29184
SectionsMeanVirtualsize	1510175.16667	28755.4285714
SectionsMinVirtualsize	60704	140
SectionMaxVirtualsize	6751839	110088
ImportsNbDLL	13	8
ImportsNb	935	155
ImportsNbOrdinal	147	0
ExportNb	20	0
ResourcesNb	134	8
ResourcesMeanEntropy	3.28753758854	4.0912689525
ResourcesMinEntropy	1.11239546515	2.45849222582
ResourcesMaxEntropy	7.99092475232	5.86996923095
ResourcesMeanSize	1965.05223881	2088.75
ResourcesMinSize	20	48
ResourcesMaxSize	67585	9640
LoadConfigurationSize	0	0
VersionInformationSize	17	0
legitimate	1	0

- `SizeOfUninitializedData` is the size of the uninitialized data section in the file, which contains data that is not initialized when the file is loaded into memory.

- `AddressOfEntryPoint` is the address of the entry point of the file, which is the location in memory where the execution of the file begins.
- `BaseOfCode` is the base address of the code section in the file, which is the starting address of the code section when the file is loaded into memory.
- `BaseOfData` is the base address of the data section in the file, which is the starting address of the data section when the file is loaded into memory.
- `ImageBase` is the preferred address of the first byte of the file when it is loaded into memory.
- `SectionAlignment` is the alignment of the sections in the file, which determines how the sections are arranged in memory.
- `FileAlignment` is the alignment of the file on disk, which determines how the file is arranged on the disk.
- `MajorOperatingSystemVersion` and `MinorOperatingSystemVersion` are the major and minor version numbers of the required operating system.
- `MajorImageVersion` and `MinorImageVersion` are the major and minor version numbers of the image.
- `MajorSubsystemVersion` and `MinorSubsystemVersion` are the major and minor version numbers of the subsystem.
- `SizeOfImage` is the size of the image, including all headers, in memory.
- `SizeOfHeaders` is the size of the headers in the file.
- `CheckSum` is the checksum of the file, which is a value calculated from the contents of the file that can be used to verify the integrity of the file.

- `Subsystem` is the subsystem required to run the file, such as a graphical user interface (GUI) or a command-line interface (CLI).
- `DllCharacteristics` is a set of flags indicating the DLL characteristics of the file, such as whether it is re-locatable or whether it supports high-entropy 64-bit virtual address space.
- `SizeOfStackReserve` is the size of the stack to reserve for the file. The stack is a portion of memory used for storing temporary data, such as function parameters and local variables.
- `SizeOfStackCommit` is the size of the stack to commit for the file. When a memory page is committed, it is reserved for the program's use and can be accessed.
- `SizeOfHeapReserve` is the size of the heap to reserve for the file. The heap is a portion of memory used for dynamic memory allocation.
- `SizeOfHeapCommit` is the size of the heap to commit for the file. When a memory page is committed on the heap, it is reserved for the program's use and can be accessed.
- `LoaderFlags` is a set of flags indicating the state of the file.
- `NumberOfRvaAndSizes` is the number of data directories in the file. Data directories contain information about the file's resources, such as its import and export tables.
- `SectionsNb` is the number of sections in the file. A file is divided into sections, each of which has a specific purpose, such as containing code or data.
- `SectionsMeanEntropy` is the mean entropy of the sections in the file. Entropy is a measure of the randomness or disorder of a system, and it can be used to assess the complexity of the file's sections.

- `SectionsMinEntropy` is the minimum entropy of the sections in the file.
- `SectionsMaxEntropy` is the maximum entropy of the sections in the file.
- `SectionsMeanRawsize` is the mean size of the sections in the file.
- `SectionsMinRawsize` is the minimum size of the sections in the file.
- `SectionMaxRawsize` is the maximum size of the sections in the file.
- `SectionsMeanVirtualsize` is the mean virtual size of the sections in the file. The virtual size of a section is the size it occupies in memory, which may be larger than its actual size on disk.
- `SectionsMinVirtualsize` is the minimum virtual size of the sections in the file.
- `SectionMaxVirtualsize` is the maximum virtual size of the sections in the file.
- `ImportsNbDLL` is the number of dynamic-link libraries (DLLs) imported by the file.
- `ImportsNb` is the number of imported functions in the file.
- `ImportsNbOrdinal` is the number of imported functions with ordinals in the file. An ordinal is a number that identifies a function in a DLL, and it can be used to import the function more quickly than by using its name.
- `ExportNb` is the number of exported functions in the file. An exported function is a function that can be called by other programs.
- `ResourcesNb` is the number of resources in the file. Resources are data that is stored in the file and can be accessed at runtime, such as images and strings.

- `ResourcesMeanEntropy` is the mean entropy of the resources in the file. Entropy is a measure of the randomness or disorder of a system, and it can be used to assess the complexity of the file's resources.
- `ResourcesMinEntropy` is the minimum entropy of the resources in the file.
- `ResourcesMaxEntropy` is the maximum entropy of the resources in the file.
- `ResourcesMeanSize` is the mean size of the resources in the file.
- `ResourcesMinSize` is the minimum size of the resources in the file.
- `ResourcesMaxSize` is the maximum size of the resources in the file.
- `LoadConfigurationSize` is the size of the load configuration data in the file. The load configuration data contains information about how the file should be loaded into memory.
- `VersionInformationSize` is the size of the version information in the file. The version information contains details about the version of the file, such as its copyright and company name.

4.1.2 Parameters

In the experiment, several parameters were used to train and test the adversarial reprogramming model. The train parameters include the number of epochs to train 12, the initial learning rate as 0.01, the batch size as 32, the weight decay as 5e-4, and the epsilon ϵ value used for rounding while updating the weights as 0.3. The test parameters include the step size at which weights were updated is set to 4 and the gamma value is 0.9.

4.1.3 Settings

The settings for the experiment involve using different portions of the training dataset to test the model, ranging in $\{80\%, 5\%, 0.1\%, 0.05\%, 0.01\%\}$. The hyper-parameter epsilon is set to a constant value of 0.3, where its impact on the model performance is also analyzed by changing the value of epsilon from 0.1 to 0.5. Four different pre-trained ImageNet Classification models (resnet50, resnet101, densenet121, and densenet161) with pre-trained weights from the PyTorch and Torch-vision modules were used as the basis for the adversarial task. The baseline models used for comparison are random forest, neural network, and transfer learning.

4.2 Evaluation

4.2.1 Impact of Pre-Trained Model

Adversarial reprogramming is a type of attack on machine learning models where an attacker attempts to cause the model to perform a task that it was not designed for by adding small, carefully crafted perturbations to the input data [61]. In this experiment, we tested the effectiveness of this attack on different ImageNet models using a small training dataset and a larger test dataset, while keeping the perturbation constant at $\epsilon = 0.3$.

Table 4.3: Evaluation on the impact of pre-trained models

Pre-trained Models	**Accuracy**	**F1-score**
resnet50	97.73	96.21
resnet101	97.50	95.89
densenet121	98.05	96.74
densenet161	97.66	96.08

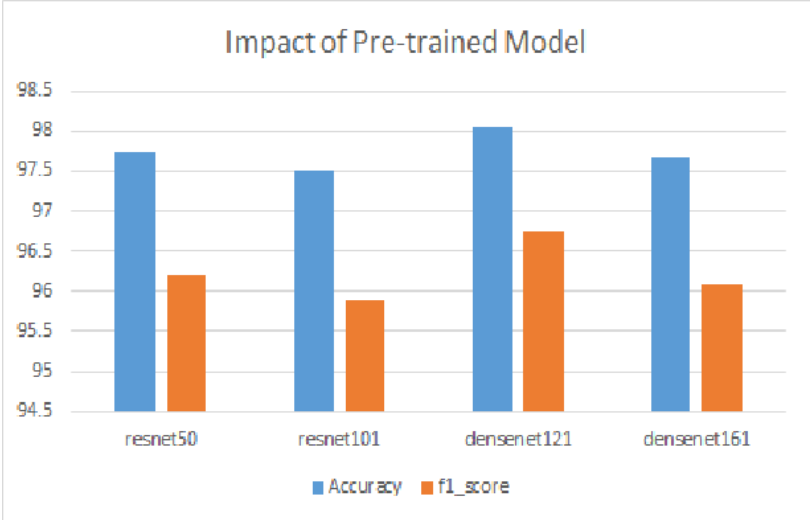

Figure 4.1: Evacuation on the impact of pre-trained model

We conducted an experiment where we tested the performance of adversarial reprogramming on four different ImageNet models: resnet50, resnet101, densenet121, and densenet161. We presented the results of the experiment in Figure 4.1 and Table 4.3, and found that the densenet121 model has the best performance, with an accuracy of more than 98%. This suggests that the densenet121 model was the most resistant to the adversarial reprogramming attack among the four models tested.

4.2.2 Impact of Data Size

In this experiment, we tested the performance of adversarial reprogramming on a dataset that was split into five different categories based on size. The categories included 80% of the dataset (with 110,437 inputs), 5% of the dataset (with 6,902 inputs), 0.1% of the dataset (with 138 inputs), 0.05% of the dataset (with 69 inputs), and 0.01% of the dataset (with 13 inputs). We kept the perturbation constant at $\epsilon = 0.3$ throughout the experiment. This experiment is significant because we are testing the performance of adversarial reprogramming on very small datasets, which may have different characteristics compared to larger datasets. It would be interesting to see how the size of the dataset affects the performance

of the adversarial reprogramming attack.

Table 4.4: Evaluation on the impact of data size

Data Size	Accuracy	F1-score
80%	98.10	96.87
5%	97.99	96.64
0.1%	94.62	90.06
0.05%	93.02	87.32
0.01%	92.08	85.37

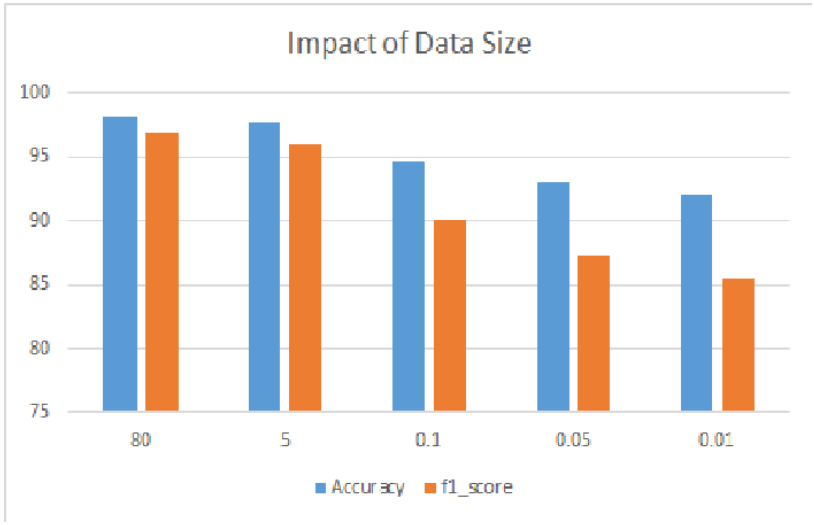

Figure 4.2: Evaluation on the impact of data size

As illustrated in Figure 4.2 and Table 4.4, we can observe that the attack had a very high efficiency of more than 98% when using 80% of the dataset (with 110,437 inputs) as the training data. We also found that the attack had an accuracy of more than 91% even when using a very small dataset (0.01% of the total dataset, with 13 inputs) as the training data. These results suggest that adversarial reprogramming is a highly effective method that is able to achieve good performance even when using a very small amount of training data, which demonstrate our proposed idea to address few-shot malware detection.

4.2.3 Impact of Perturbation

In this experiment, we tested the effect of the perturbation function on the performance of adversarial reprogramming. We did this by varying the parameter ϵ, which is used to control the magnitude of the perturbations applied to the input data, and observing the performance of the attack with 5% of the dataset as the training data. This is also an interesting experiment because the perturbation function is a key component of adversarial reprogramming, and understanding how it affects the attack can provide insights into how the attack works and how to defend against it. It would be interesting to see how the performance of the attack changes as the value of ϵ is varied

Table 4.5: Evaluation on the impact of perturbation magnitude

Perturbation ϵ	**Accuracy**	**F1-score**
0.1	97.60	95.98
0.2	97.80	96.34
0.3	97.99	96.64
0.4	97.66	96.45
0.5	98.09	97.60

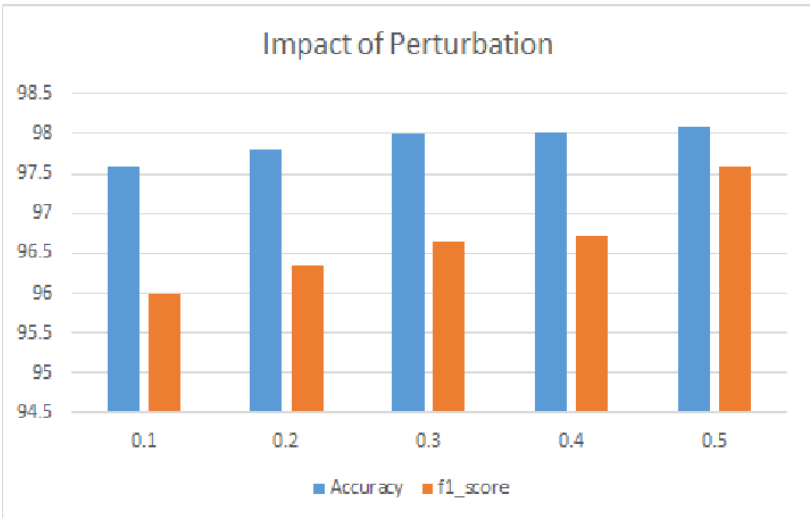

Figure 4.3: Evaluation on the impact of perturbation magnitude

From figure 4.3 and table 4.5, we can see the experimental results where we tested the performance of adversarial reprogramming with 5% of the dataset as the training data, and varied the parameter $\epsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ that controls the magnitude of the perturbations applied to the input data. We recorded the accuracy and f1-score for each value of ϵ that we tested. The results show that as the value of ϵ increases, the accuracy and f1-score of the attack also generally increase. However, there is a noticeable drop in accuracy and f1-score when ϵ is increased from 0.3 to 0.4. This suggests that there may be an optimal value of ϵ for this particular dataset that falls in the range between 0.3 and 0.4.

4.2.4 Comparison with Baselines

In this experiment, we compared the performance of our proposed model with three baseline models (random forest, neural network, and transfer learning) on small data settings. We used three different train datasets with sizes of 0.1%, 0.05%, and 0.01% of the total data samples, and trained all four models on each of these datasets. Here, transfer learning is implemented in a way similar to adversarial reprogramming, with only one difference that transfer learning fine-tunes the parameters for all layers. The goal of the experiment was to see how well each of the models performs on small datasets, and compare the performance of the proposed model with the baseline models. This is an interesting experiment because it can provide insights into how well these types of models can generalize to small datasets, and whether there are any specific characteristics of the dataset or the models that contribute to their performance. It would be also interesting to see how the performance of the models changes as the size of the train dataset is decreased, and whether there are any notable advantages yielded by the proposed model compared to the baseline models.

As illustrated in Table 4.6 and Figure 4.4, we can see the comparative results of four different models (random forest, neural network, transfer learning, and adversarial reprogramming) on a small datasets with sizes of 0.1%, 0.05%, and 0.01% of the total dataset

Table 4.6: Comparison with baselines

Baseline Models		**0.1%**	**0.05%**	**0.01%**
Random Forest	Accuracy	95.10	91.33	88.92
	F1-score	92.23	82.91	80.22
Neural Network	Accuracy	93.79	89.58	80.87
	F1-score	90.43	83.70	54.09
Transfer Learning	Accuracy	91.39	86.55	64.09
	F1-score	83.95	71.07	60.60
Adversarial Reprogramming	Accuracy	95.62	92.81	91.59
	F1-score	90.79	87.03	84.12

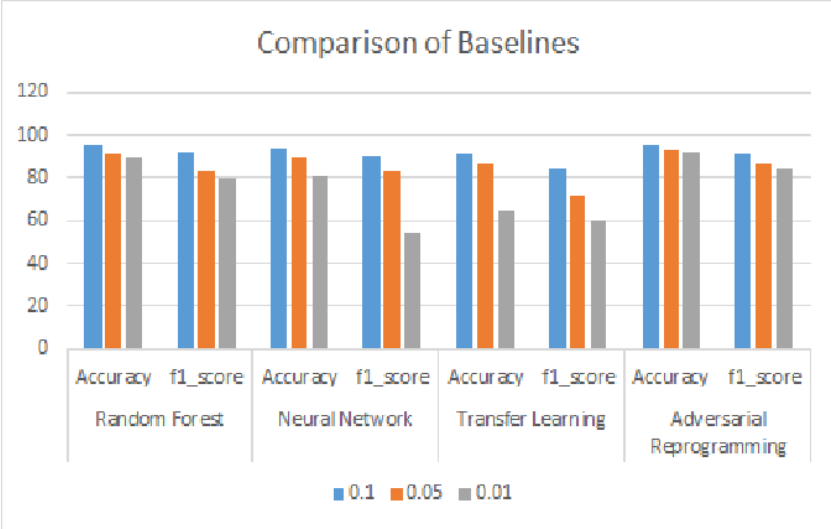

Figure 4.4: Comparison with baselines

in terms of the accuracy and f1-score. The results show that the performance of all four models decreases as the size of the train dataset is decreased. However, the adversarial reprogramming model generally performs better than the baseline models across all train dataset sizes. This suggests that the adversarial reprogramming model may have better generalization performance compared to the other models when working with small datasets.

Challenges

One challenge that was faced while building the model for adversarial reprogramming was related to input transformation and embedding features into the host image, as well as saving the indexes of the host image to create a new image dataset. This process can be complex and time-consuming, and can require a lot of trial and error to get right.

Another challenge was finding suitable datasets to use for adversarial reprogramming. we tried using two popular datasets, MNIST and CIFAR10, for adversarial reprogramming, but found that they were not well suited for use with pre-trained models. MNIST is a dataset of gray-scale images of handwritten digits, while CIFAR10 is a dataset of colored images of objects in 10 classes. Both datasets are widely used for machine learning research, but they have some limitations that made them not well suited for use with pre-trained models in this context. One limitation of MNIST is that it has only one channel for gray-scale images, which can limit the types of models that can be used with it. CIFAR10, on the other hand, is a relatively small dataset, which can make it difficult to achieve good performance with more complex models. This highlights the importance of selecting appropriate datasets for a given task, and the challenges that can arise when working with datasets that are not well suited for building a pre-trained model.

In addition to these challenges, adversarial reprogramming can also be computationally intensive, particularly when working with large target models or when the perturbations require many iterations to find. This can be a challenge for users with limited computational resources, and may require the use of specialized hardware or techniques to manage

the computational demands of the task. Finally, adversarial reprogramming often involves fine-tuning a pre-trained model on a small number of examples, which can lead to overfitting and reduced generalization and accuracy on new and unseen data. This highlights the need to carefully balance the trade-off between model performance and generalization, and to carefully monitor model performance on new data to ensure that it is not overfitting.

Future Work

One direction for future work is to explore the effect of different hyper-parameters on the performance of the adversarial reprogramming model. This could involve testing different values for the perturbation parameter epsilon, or trying different optimization algorithms or learning rates. By varying these and other hyper-parameters, it may be possible to find configurations that result in improved performance or faster convergence.

Another direction for future work is to test the model using different pre-trained models as the target model. This could involve using models trained on different datasets or using models with different architectures or hyper-parameters. By comparing the performance of the adversarial reprogramming model across different target models, it may be possible to identify any characteristics of the target model that affect its vulnerability to the attack or to identify strategies for making the target model more resistant to adversarial reprogramming.

One potential application of adversarial reprogramming is improving machine learning models' robustness. By altering the behavior of the model in a way that makes it more resistant to noise or perturbations in the input data, it may be possible to improve the generalization performance of the model and make it more robust to changes in the data distribution. This could be an important area of future work, as robustness and generalization are critical concerns in many machine-learning applications.

Conclusion

Malware is software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system. To protect against malware, effective techniques for detecting and mitigating malware are needed. One approach to detecting malware is to use machine learning models, which are algorithms that can learn patterns in data and make predictions or decisions based on those patterns. Machine learning models can be effective for detecting malware, but they often require large datasets to achieve good performance. This can be a challenge in the context of malware detection, as the datasets used for training and evaluation are often quite small.

To address this challenge, in this study, we aimed to develop a machine learning model that can achieve good performance even when using small datasets. To do this, we used an adversarial reprogramming technique, which involves modifying the behavior of a pre-trained model by adding small perturbations to the input data. Adversarial reprogramming has been shown to be effective in improving the performance of machine learning models on small datasets.

To test this hypothesis, we compared the performance of the adversarial reprogramming model with traditional baseline models on a dataset of malware and benign samples. We found that the adversarial reprogramming model was able to produce efficient results even when using very small datasets, outperforming the traditional baseline models. We also evaluated the performance of the adversarial reprogramming model with different pre-trained ImageNet classification models, with different parameter settings, and with differ-

ent data sizes.

Overall, the results of this study suggest that adversarial reprogramming is a promising approach for improving the performance of machine learning models on small datasets, and could be a valuable tool for improving malware detection in situations where the available datasets are limited. It would be interesting to see how the approach performs on other types of small datasets, and whether it is generally applicable to a wide range of machine learning tasks.

Bibliography

[1] Muhamed Fauzi Bin Abbas and Thambipillai Srikanthan. Low-complexity signature-based malware detection for iot devices. In *International Conference on Applications and Techniques in Information Security*, pages 181–189. Springer, 2017.

[2] Muhammad Shoaib Akhtar and Tao Feng. Detection of malware by deep learning as cnn-lstm machine learning techniques in real time. *Symmetry*, 14(11):2308, 2022.

[3] Malware AV-TEST The Independent IT-Security Institute. Atlas - malware amp; pua, 2023.

[4] Nureni Ayofe Azeez, Oluwanifise Ebunoluwa Odufuwa, Sanjay Misra, Jonathan Oluranti, and Robertas Damaševičius. Windows pe malware detection using ensemble learning. In *Informatics*, volume 8, page 10. MDPI, 2021.

[5] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*, pages 113–120. IEEE, 2013.

[6] Niket Bhodia, Pratikkumar Prajapati, Fabio Di Troia, and Mark Stamp. Transfer learning for image-based malware classification. *arXiv preprint arXiv:1903.11551*, 2019.

[7] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26, 2011.

[8] Yuhan Chai, Lei Du, Jing Qiu, Lihua Yin, and Zhihong Tian. Dynamic prototype network based on sample adaptation for few-shot malware detection. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[9] Li Chen. Deep transfer learning for static malware classification. *arXiv preprint arXiv:1812.07606*, 2018.

[10] Lingwei Chen, Yujie Fan, and Yanfang Ye. Adversarial reprogramming of pretrained neural networks for fraud detection. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2935–2939, 2021.

[11] Lingwei Chen, Xiaoting Li, and Dinghao Wu. Adversarially reprogramming pretrained neural networks for data-limited and cost-efficient malware detection. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 693–701. SIAM, 2022.

[12] Luca Demetrio, Scott E Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security (TOPS)*, 24(4):1–31, 2021.

[13] Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial reprogramming of neural networks. *arXiv preprint arXiv:1806.11146*, 2018.

[14] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[15] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6:30996–31011, 2018.

[16] Eric Filiol, Grégoire Jacob, and Mickaël Le Liard. Evaluation methodology and theoretical model for antiviral behavioural detection strategies. *Journal in Computer Virology*, 3(1):23–37, 2007.

[17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[18] Meenu Ganesh, Priyanka Pednekar, Pooja Prabhuswamy, Divyashri Sreedharan Nair, Younghee Park, and Hyeran Jeon. Cnn-based android malware detection. In *2017 international conference on software security and assurance (ICSSA)*, pages 60–65. IEEE, 2017.

[19] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[20] Kasthurirangan Gopalakrishnan, Siddhartha K Khaitan, Alok Choudhary, and Ankit Agrawal. Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and building materials*, 157:322–330, 2017.

[21] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. Dl4md: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Science (ICDATA)*, page 61. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[22] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[23] Mike Huisman, Jan N Van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541, 2021.

[24] Nwokedi Idika and Aditya P Mathur. A survey of malware detection techniques. *Purdue University*, 48(2):32–46, 2007.

[25] Ashraful Islam, Chun-Fu Richard Chen, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, and Richard J Radke. Dynamic distillation network for cross-domain few-shot recognition with unlabeled data. *Advances in Neural Information Processing Systems*, 34:3584–3595, 2021.

[26] Rituraj Kaushik, Timothée Anne, and Jean-Baptiste Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5269–5276. IEEE, 2020.

[27] Hung-yi Lee, Shang-Wen Li, and Ngoc Thang Vu. Meta learning for natural language processing: A survey. *arXiv preprint arXiv:2205.01500*, 2022.

[28] Wenjia Li, Jigang Ge, and Guqian Dai. Detecting malware for android platform: An svm-based approach. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, pages 464–469. IEEE, 2015.

[29] Xiaoting Li, Lingwei Chen, and Dinghao Wu. Turning attacks into protection: Social media privacy protection using adversarial attacks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 208–216. SIAM, 2021.

[30] Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, 18(9):1336–1347, 2017.

[31] Wu Liu, Ping Ren, Ke Liu, and Hai-xin Duan. Behavior-based malware analysis and detection. In *2011 first international workshop on complexity and data mining*, pages 39–42. IEEE, 2011.

[32] Alejandro Martín, Alejandro Calleja, Héctor D Menéndez, Juan Tapiador, and David Camacho. Adroit: Android malware detection using meta-information. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[33] Andrew McDole, Mahmoud Abdelsalam, Maanak Gupta, and Sudip Mittal. Analyzing cnn based behavioural malware detection techniques on cloud iaas. In *International Conference on Cloud Computing*, pages 64–79. Springer, 2020.

[34] Grégoire Mesnil, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, et al. Unsupervised and transfer learning challenge: a deep learning approach. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 97–110. JMLR Workshop and Conference Proceedings, 2012.

[35] Farid Ghareh Mohammadi, Hamid R Arabnia, and M Hadi Amini. On parameter tuning in meta-learning for computer vision. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 300–305. IEEE, 2019.

[36] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[37] Paarth Neekhara, Shehzeen Hussain, Shlomo Dubnov, and Farinaz Koushanfar. Adversarial reprogramming of text classification neural networks. *arXiv preprint arXiv:1809.01829*, 2018.

[38] Kb Pachauri. Adversarial machine learning, Aug 2020.

[39] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[40] Matthew E Peters, Sebastian Ruder, and Noah A Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*, 2019.

[41] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[42] Hemant Rathore, Swati Agarwal, Sanjay K Sahay, and Mohit Sewak. Malware detection using machine learning and deep learning. In *International Conference on Big Data Analytics*, pages 402–411. Springer, 2018.

[43] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, pages 15–18, 2019.

[44] Ravi K Samala, Heang-Ping Chan, Lubomir M Hadjiiski, Mark A Helvie, Kenny H Cha, and Caleb D Richter. Multi-task transfer learning deep convolutional neural network: application to computer-aided diagnosis of breast cancer on mammograms. *Physics in Medicine & Biology*, 62(23):8894, 2017.

[45] Oleg Savenko, Andrii Nicheporuk, Ivan Hurman, and Sergii Lysenko. Dynamic signature-based malware detection technique based on api call tracing. In *ICTERI Workshops*, pages 633–643, 2019.

[46] Prashanth Gurunath Shivakumar and Panayiotis Georgiou. Transfer learning from adult to children for speech recognition: Evaluation, analysis and recommendations. *Computer speech & language*, 63:101077, 2020.

[47] Arif Siddiqi. Adversarial security attacks and perturbations on machine learning and deep learning methods. *arXiv preprint arXiv:1907.07291*, 2019.

[48] Carl-Johann Simon-Gabriel, Yann Ollivier, Léon Bottou, Bernhard Schölkopf, and David Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension, 2019.

[49] Ankush Singla, Elisa Bertino, and Dinesh Verma. Overcoming the lack of labeled data: Training intrusion detection models using transfer learning. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 69–74. IEEE, 2019.

[50] Steven A Stahl and Marilyn M Fairbanks. The effects of vocabulary instruction: A model-based meta-analysis. *Review of educational research*, 56(1):72–110, 1986.

[51] Gil Tahan, Lior Rokach, and Yuval Shahar. Mal-id: Automatic malware detection using common segment analysis and meta-features. *Journal of Machine Learning Research*, 13(4), 2012.

[52] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[53] Umm-e-Hani Tayyab, Faiza Babar Khan, Muhammad Hanif Durad, Asifullah Khan, and Yeon Soo Lee. A survey of the recent trends in deep learning based malware detection. *Journal of Cybersecurity and Privacy*, 2(4):800–829, 2022.

[54] Trung Kien Tran, Hiroshi Sato, and Masao Kubo. Image-based unknown malware classification with few-shot learning models. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pages 401–407. IEEE, 2019.

[55] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Qin Zheng. Image-based malware classification using ensemble of cnn architectures (imcec). *Computers & Security*, 92:101748, 2020.

[56] Deepak Venugopal and Guoning Hu. Efficient signature based malware detection on mobile devices. *Mobile Information Systems*, 4(1):33–49, 2008.

[57] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[58] R Vinayakumar, KP Soman, Prabaharan Poornachandran, and S Sachin Kumar. Detecting android malware using long short-term memory (lstm). *Journal of Intelligent & Fuzzy Systems*, 34(3):1277–1288, 2018.

[59] Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in Neural Information Processing Systems*, 32, 2019.

[60] Duo Wang, Yu Cheng, Mo Yu, Xiaoxiao Guo, and Tao Zhang. A hybrid approach with optimization-based and metric-based meta-learner for few-shot learning. *Neurocomputing*, 349:202–211, 2019.

[61] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G Ororbia, Xinyu Xing, Xue Liu, and C Lee Giles. Adversary resistant deep neural networks with an application to malware detection. In *Proceedings of the 23rd ACM sigkdd international conference on knowledge discovery and data mining*, pages 1145–1153, 2017.

[62] Xi Xiao, Shaofeng Zhang, Francesco Mercaldo, Guangwu Hu, and Arun Kumar Sangaiah. Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications*, 78(4):3979–3999, 2019.

[63] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *International Conference on Machine Learning*, pages 7045–7054. PMLR, 2019.

[64] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. High accuracy android malware detection using ensemble learning. *IET Information Security*, 9(6):313–320, 2015.

[65] Min Zhao, Fangbin Ge, Tao Zhang, and Zhijian Yuan. Antimaldroid: An efficient svm-based malware detection framework for android. In *International conference on information computing and applications*, pages 158–166. Springer, 2011.

[66] Jinting Zhu, Julian Jang-Jaccard, Amardeep Singh, Ian Welch, AI-Sahaf Harith, and Seyit Camtepe. A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. *Computers & Security*, 117:102691, 2022.

[67] Aqil Zulkifli, Isredza Rahmi A Hamid, Wahidah Md Shah, and Zubaile Abdullah. Android malware detection based on network traffic using decision tree algorithm. In *International Conference on Soft Computing and Data Mining*, pages 485–494. Springer, 2018.