Detecting Anomalies in Network Systems by Leveraging Neural Networks

by

Mohammad J. Hashemi

B.S., Sharif University of Technology, 2014

M.S., University of Colorado Boulder, 2018

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2021

This thesis entitled:
Detecting Anomalies in Network Systems by Leveraging Neural Networks
written by Mohammad J. Hashemi
has been approved for the Department of Computer Science

Prof. Eric Keller

Prof. Rick Han

Prof. Qin Lv

Prof. Nikolaus Correll

Prof. Eric Wustrow

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Hashemi, Mohammad J. (Ph.D., Computer Science)

Detecting Anomalies in Network Systems by Leveraging Neural Networks

Thesis directed by Prof. Eric Keller

The growth of cyber attacks in both numbers and varieties in recent years demands building a more sophisticated network intrusion detection system (NIDS). Signature-based NIDS which were traditionally being used to detect malicious traffic, can't cope with today's variety of network attacks as they are incapable of detecting attacks that have not been seen before and, therefore, do not have a known signature to detect. As a result, Machine Learning-based NIDS built on neural networks are getting more attention due to their ability to detect such attacks.

While moving towards ML-based NIDS holds promise, such systems have their own drawbacks. The first problem is the vulnerability of neural networks to adversarial examples. It has been shown that an adversary can fool deep learning models used in tasks such as image classification by applying small and imperceptible modifications to the input images to force the models to predict them as their desired class. But the existing ML-based NIDS were not evaluated in such an adversarial setting, even though their potential of being vulnerable to the adversarial examples can face the network systems with significant threats. Secondly, training a model in a supervised manner on the network traffic has its own challenges; One major issue is that labeling a huge dataset consisting of billions of packets is expensive and time-consuming. Even if we can label such a dataset and train a model on it, such a model can hardly detect new attacks not included in the train set. For this reason, the existing work mostly trained their model in an unsupervised fashion, but in order to detect different network attacks, these models generate so many false alerts and any attempt to reduce their false positive rate significantly deteriorates their detection rate.

In this dissertation, we first identify the challenges to evaluate ML-based NIDS trained on network traffic in an adversarial setting. We demonstrate how an adversary can craft adversarial examples against such NIDS by applying a set of legitimate transformations to the malicious traffic to fool the NIDS without breaking the underlying network protocols. Our evaluation shows that existing work is vulnerable to the adversarial examples and our attack can reduce the detection rate of different network attacks by up to 70%.

From these findings, we can see there is a need for new a technique to make NIDS more robust to the adversarial examples. For this, we introduce Reconstruction from Partial Observation (RePO) as a new mechanism to build a more robust anomaly-based NIDS against the adversarial examples with the help of denoising autoencoders which can be trained in an unsupervised manner and is also capable of detecting different types of network attacks in a low false alert setting. Our evaluation conducted on a dataset with a variety of network attacks shows our method can improve detection of malicious traffic by up to 29% in a normal setting and by up to 45% in an adversarial setting compared to other recently proposed anomaly-based NIDS.

Finally, we complete this dissertation by addressing the data labeling issue that exists in traditional supervised learning methods. We demonstrate how a more accurate NIDS capable of detecting unseen attacks can be built while labeling a limited number of data points and by utilizing domain adaptation. We present Proportional Progressive Pseudo-Labeling (PPPL) as a new domain adaptation approach that generalizes better than other methods across different input types and we show how it can be used to train a model on a small portion of network traffic which is labeled and consists of some known attacks to detect new types of anomalies in the future. Our evaluation shows that PPPL can significantly increase the detection rate of unseen anomalies by up to 58% compared to other state-of-the-art methods.

Acknowledgements

First of all, I would like to thank my adviser, Prof. Eric Keller, for his help and guidance during my Ph.D. studies. His strong support provided me with what I needed to accomplish my academic goals, he was always available whenever I needed his help despite being extremely busy, and he was always supportive of my research interests. I appreciate all of his help during this journey.

I also would like to extend my thanks to my committee members Prof. Rick Han, Prof. Christine Lv, Prof. Nikolaus Correll, and Prof. Eric Wustrow for assigning their invaluable time to my dissertation and providing me with their constructive feedback.

I am also thankful for working along with my lab mates Greg Cusack who assisted me with some of my publications, and Azzam Alsudais and Marcelo Abranches for their help and support.

In addition, I am grateful for all the support I have received from my friends while being in the United States. I would like to especially thank Saeid Tizpaz-Niari and Hadi Yazdi for all of their help during my graduate studies. Without having them around, life could have been more difficult especially because of being separated from the family by the inhuman travel ban which was in place during my Ph.D. years.

Finally, I want to thank my family. I am grateful for all the support I have received from my sister and my nephew. Moreover, I want to especially thank my mother and father for their unconditional love and support throughout my life and for being there for me whenever I needed them during this journey. I wouldn't have been where I am today without all the support I have received from them throughout my life.

Contents

Tables

Table

Figures

Chapter 1

Introduction

Computer networks play an important role in our everyday life. They get larger and more complex day by day and impact the performance of a wide range of services we use on a daily basis from accessing our emails to accessing the electricity. The dependency of such services to these network systems intrigues attackers to interfere with them to take advantage of their available resources, access their sensitive data unauthorizedly, deny their legitimate users to access a specific service and etc. The cost of these attacks on exploited businesses can be huge. Based on one estimation the average cost of a denial of service (DoS) attack on a company was $1.7 million in 2018 and the average cost of all cybercrimes on each affected company was $13 million [5]. Based on another estimation, the annual cost of cybercrime on the whole world will be $6 trillion by 2021 [68]. But the monetary cost for exploited businesses is not the only problem with these attacks. In fact, their impact can be much harsher such as a wide-scale power outage [40].

The severe adverse effects of these cyber attacks forced the companies to invest billions of dollars in designing and building better tools to early detect and potentially prevent them [63]. Network Intrusion Detection Systems (NIDS) are one line of defense against such attacks. Traditionally, they have been built in a way to detect cyber attacks by looking for specific signatures either a known sequence of bytes or fixed access patterns. While providing a degree of protection for networks, this approach has a problem in that it relies on signatures of **known** attacks. As a result, zero-day attacks will bypass these signature-based NIDS. Therefore, another type of solution is needed to detect the zero-day attacks where no known signature exists for them beforehand.

Because of the impressive results deep neural networks could achieve in other tasks such as image classification, object detection, speech recognition and etc. some researchers incorporated them into the NIDS to build ML-based NIDS which are capable of detecting unseen attacks [7, 17, 62, 65, 104, 107, 108, 110]. But these NIDS have their own drawbacks:

One major problem is the vulnerability of neural networks to adversarial examples. It has been shown that deep learning models used in tasks such as image classification can be fooled by an adversary who slightly and carefully modifies a given input to force the model to predict it as the adversary's desired class instead of the real label of the input. For example, an adversary can modify a few pixels of somebody's image to fool a face recognition system to detect it as anyone else. These inputs are called adversarial examples. Despite this vulnerability of neural networks to the adversarial examples, previously proposed ML-based NIDS have not been evaluated in an adversarial setting. Therefore we don't know to what degree they are vulnerable to this attack.

The second problem is that in many cases there is no easy and efficient way to label a huge dataset consisting of billions of packets to training a model on. In addition, since the characteristics of different network attacks differ a lot from each other, even when we manage to label a dataset containing some of the known attacks and train a model on, such a model can hardly detect unseen attacks. Because of such difficulties of training an NIDS in a supervised manner, previous work has utilized unsupervised training to build ML-based NIDS. However such models can only detect different network attacks while generating a high number of false alerts and any attempt to bring down their false-positive rates significantly deteriorate their ability to detect real malicious traffic. But a good NIDS should be able to detect a variety of network attacks while keeping false alerts at a low level. Because a high false-positive rate significantly increases the workload of security experts to sort through all the traffic labeled as malicious to detect the real attacks.

Therefore, in this dissertation, we want to address the aforementioned problems of the existing ML-based NIDS and demonstrate how a better NIDS can be built. More specifically, in this dissertation, we seek to answer the following three questions:

(1) How can we evaluate an NIDS trained on network traffic in an adversarial setting?

(2) How can we make an NIDS more robust against adversarial examples?

(3) How can we train an NIDS in a supervised manner more efficiently in a way to only label a limited number of inputs and still be able to detect new anomalies in the future?

In the remainder of this chapter, we first provide some background on how a deep neural network works. Then we demonstrate how an adversarial example can be crafted, in more detail. After that, we talk about the challenges that should be addressed to successfully create an adversarial example against an NIDS and briefly discuss our approach. We follow this section with another one in which we talk about the challenges of building a robust NIDS against the adversarial examples and briefly explain how our solution works. Finally, we refer to the challenges of training an NIDS in a supervised manner and how we solve them. We follow this chapter by the next three chapters where in each of them we focus on one of the above questions and thoroughly address it. Most of the contributions of these three chapters have first appeared as some publications in different venues (Chapter 2 in [34], Chapter 3 in [35] and Chapter 4 in [36]). Finally, we conclude this dissertation in Chapter 5.

1.1 Deep Neural Networks

This section wants to provide some background on how a deep neural network works, which helps to understand the crafting procedure of adversarial examples better.

A deep neural network (as a classifier) is a non-linear function that maps an input to a probability vector in which each of its elements corresponds to a class score. The one that has a larger score will be considered as the model's prediction. It consists of multiple sequential layers such that the output of one layer becomes the input of the next one. Each layer has a set of parameters that are initialized randomly, applies a non-linear transformation to its input and sends it to another space with a different dimension. By varying those parameters, the output of the classifier gets changed. The goal is to find those parameters such that for most of the inputs, it predicts their labels correctly, which means that the probability corresponding to their true label should be larger than the others. Figure 2 illustrates a neural network and its different components. In this figure,

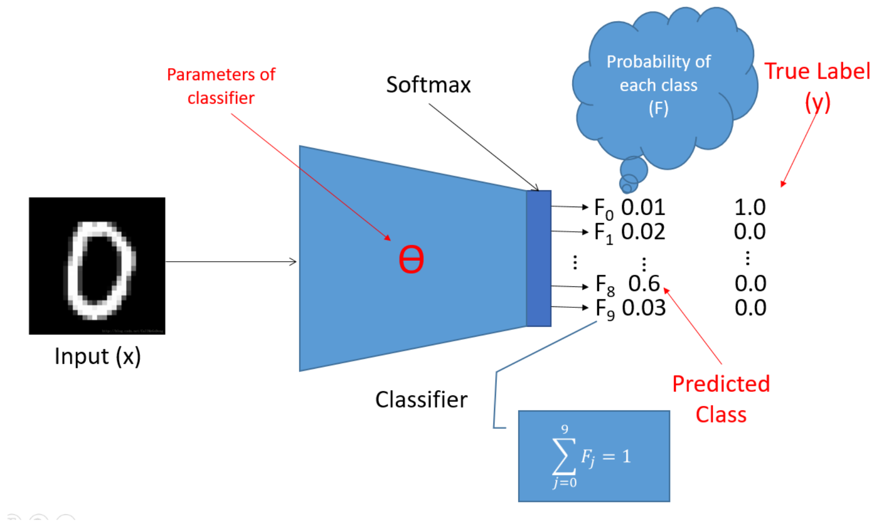

Figure 1.1: Illustration of a DNN classifier.

the input is a hand-written digit (number 0), and we want to train the model to classify different digits correctly. The classifier parameters have shown by θ, which are initialized randomly and should be tuned during the training phase. The output is shown by F which is a probability vector so $\sum_{i=0}^{9} F_j = 1$. The last layer of this classifier is a softmax ($\sigma(.)$) function which converts its inputs to probabilities as follows:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e_k^z} \text{ for } k = 1, 2, ..., K$$

Where K is the total number of classes. Since the parameters are initialized randomly, the classifier would output random probabilities for different inputs before training. For example, this classifier "thinks" that the input is 8 with 60% probability and it is 0 by 1% probability. So since F_8 is greater than the others, the input would be classified as 8 by this classifier. In this figure, you can also see the true label corresponding to image zero, which is a one-hot vector whose first element is 1 and other elements are 0. These labels are used in the training phase to tune the parameters of the network.

In order to train this network, we want to find the set of parameters that make it predict most of the inputs correctly. So we have to maximize the score corresponding to the true label of each input. For example, if the input is number 7, we have to maximize $F_7(x)$. If it is number 5, we have

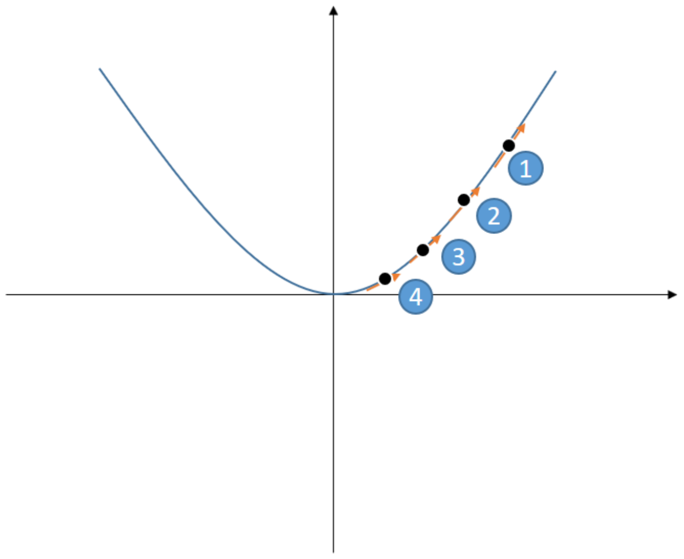

Figure 1.2: Illustration of gradient descent. The horizontal axis shows θ. The vertical axis shows $G(\theta) = \theta^2$. By moving in the opposite direction of the slope at each step, we can minimize function G.

to maximize $F_5(x)$ and so on. So, in general, we can say that we need to find a θ that maximizes $\sum_{j=0}^{K} y_j F_j$ for every input. Note that y is a vector and only one of its elements is 1 and others are 0. Instead of maximizing $\sum_{j=0}^{K} y_j F_j$ we can minimize $-\sum_{j=0}^{K} y_j log(F_j)$. So mathematically, we need to solve the following problem to train a model:

$$argmin_\theta \left(-\frac{1}{N} \sum_{i=0}^{N} \sum_{j=0}^{K} y_{ij} log(F_j(x_i)) \right)$$

Where N is the total number of samples in our training set and K is the total number of classes. This is called a cross-entropy loss function, which is a function of θ. A lower value of this function means better predictions over the training set. In order to solve this minimization problem, we use a technique called gradient descent. Figure 3 illustrates this technique for a simple function $G(\theta) = \theta^2$. As can be seen, in this figure, at any point, if we move the input θ in the opposite direction of the slope (the derivative of the function with respect to its input), the value of the function decreases. So in order to minimize the function $G(\theta)$ with respect to θ we can take a small step in the opposite direction of the slope and then update the value of theta and do it iteratively until we reach the minimum value as follows:

$$\theta = \theta - \alpha \frac{\partial G}{\partial \theta}$$

Where α shows the magnitude of each step and $\frac{\partial G}{\partial \theta}$ corresponds to the slope and minus sign means that we want to move in the opposite direction of the slope.

This technique also works for the loss function of a DNN. The only difference is that instead of one variable, there are multiple parameters. So we need to take partial derivatives with respect to each of the parameters and update them separately.

1.2 Adversarial Examples

It has been shown that deep neural networks like traditional machine learning models are vulnerable to the adversarial example (evasion) attack [12, 95]. For the first time, Szegedy et al. in [95] and Biggio et al. in [12] showed that image classifiers are vulnerable to this type of attack. The adversarial example attack is an attack during the inference phase. For this attack, the attacker doesn't change model parameters but modifies their own inputs to make the model predict them as the desired class and the perturbation added to input images is imperceptible in many cases to a human observer. Figure 1. illustrates this problem better. In this figure, the adversarial example is crafted by adding the perturbation to the "clean" image. In the rest of this section, we use some notations that we define here:

- x: the legitimate (clean) input. $x \in [0, 1]^m$, where m is the number of pixels in an image.
- y: the label corresponding to the legitimate input.
- x': the adversarial input. $x' \in [0, 1]^m$.
- y': the label corresponding to the adversarial input, which is different from its original label.
- y_{target}: the label which an adversary wants to make the classifier output.
- $F(.)$: the classifier which maps an image to a label. For correctly predicted inputs we have $F(x) = y$.
- θ: the parameters of the classifier.

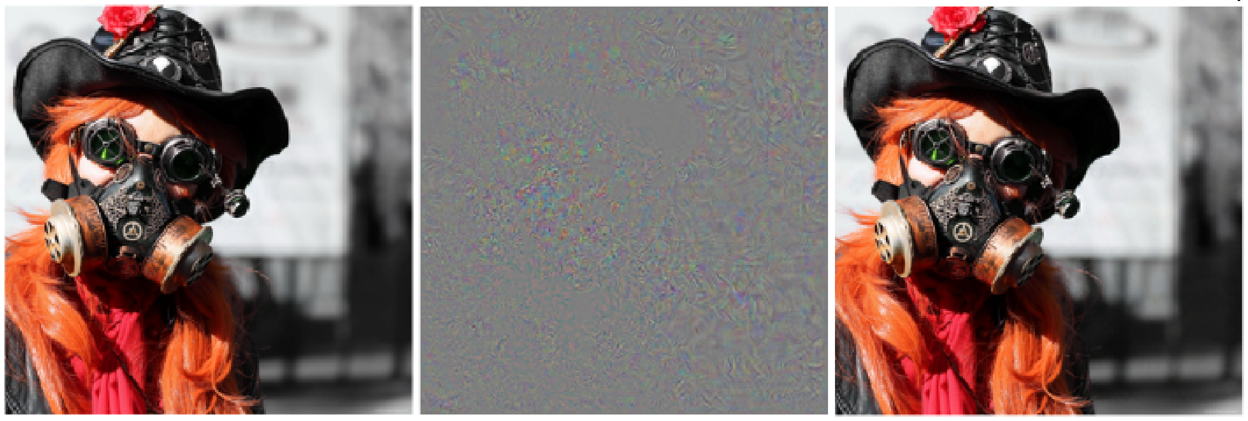

Figure 1.3: Demonstration of an adversarial example: left: a legitimate example classified as a gas mask; middle: the carefully generated perturbation magnified by 10x; right: an adversarial example classified as a French bulldog.

- $Z(.)$: the logits, which are the inputs of the Softmax layer. So, $Softmax(Z(x)) = F(x)$.
- δ: the perturbation which is added to a legitimate example to make it adversarial. So, $x' = x + \delta$.

The procedure to craft an adversarial example against an image classifier can be formulated as a box-constraint optimization problem as follows:

$$argmin_{\delta}||\delta||_p \text{ s.t. } (x+\delta) \in [0,1]^m \text{ and } F(x+\delta) = y_{target}$$

By solving this minimization problem, the attacker can find the minimum perturbation that, if added to a given image, the result will be predicted as the attacker's desired class instead of its true label. Since neural networks are not convex, solving this optimization problem is intractable. Thus, researchers came up with different heuristics to solve it [9, 15, 30, 67, 74, 95].

One of the best heuristics in a white-box setting in which the adversary has access to all of the model parameters has been designed by Carlini et al. [15]. They designed their attack by introducing a new objective function. The objective function that they used is as follows:

$$\text{minimize } c.||\delta||_p + f(x+\delta) \text{ s.t. } x+\delta \in [0,1]^m$$

in which p can be 0,2 or ∞. One of their choices for function f is:

$$f(x') = (max_{i \neq t}(Z(x')_i) - Z(x')_t)^+$$

in which, t is the target label, $(e)^+$ is short-hand for $max(e, 0)$, and c is a hyperparameter that determines the trade-off between the amount of distortion and the growth of the target score. By decreasing c, the amount of distortion and the success probability grows. They showed that they could craft adversarial examples for multiple different image datasets with less distortion compared to other white-box attacks by using this function. This minimization basically says that we want to find a δ such that its magnitude is minimal (with respect to $l0$, $l2$ or $l\infty$ norm) and the logit value corresponding to the target label is larger than other logits, which makes the classifier predict the input as the target class. This optimization problem is solved with the help of gradient descent, which we mentioned earlier.

Accessing the model parameters is not a strict requirement to craft an adversarial example against it. Biggio et al. in [12] and Papernot et al. in [73] showed that an adversary can still craft adversarial examples against a model in a black-box setting by querying the target model and training a substitute model using the labels predicted by the target model. In this case, after training the substitute model, the adversary can craft adversarial examples against the substitute model in order to transfer them to the target model.

It has also been shown that adversarial examples exist in areas beyond digital images. Kurakin et al. in [49] showed that this attack is applicable in the physical world as well. Sharif et al. in [89] showed how an adversary can print a sticker to add it to glasses to fool face recognition systems. Later, Eykholt et al. in [25] showed that an adversary can place a few stickers on a stop sign to fool a classifier potentially deployed on a moving vehicle, *e.g.,* causing it to predict the stop sign as a speed limit sign. Carlini et al. in [16] showed that an adversary can craft adversarial examples against speech-to-text systems. Grosse et al. in [32] also showed that it is possible to craft adversarial examples against malware detectors. Crafting adversarial examples in each of these domains introduced its own problems that need to be tackled to fool the model successfully. For example, as shown in Sharif et al. [89] using the attacks introduced for digital images, one cannot fool a face recognition system in the physical world because of some new constraints. For example, in this case, the limitation in the number of colors that a printer can print was a constraint, and this constraint prevents many pixels from being printed with their exact value as the crafting procedure found them.

1.3 NIDS in an Adversarial Setting

As we mentioned earlier, existing ML-based NIDS were not evaluated in an adversarial setting. However, as we saw, deep learning models are vulnerable to adversarial examples. That is to say, even though the existing NIDS might be able to detect a network attack, the attacker still might be able to bypass them by generating an adversarial version of that network attack. Therefore, to understand better how robust the existing work is against adversarial examples, we need to evaluate them in such a setting.

1.3.1 Challenge

The adversarial example attacks designed against image classifiers cannot be directly applied to the NIDS as the images and network traffic have some intrinsic differences. For example, an attack designed against an image classifier might modify all the pixels of an image to successfully carry out the attack. This is allowable for the images as the pixel values in an image are independent of each other. However, the features extracted from network traffic depend on each other and therefore, such modifications are not allowed. So, a new method should be designed to craft adversarial examples against NIDS trained on network traffic. Such a method should be designed in a way to satisfy some constraints. First, we have to be sure that the adversarial version of a given network attack doesn't break down the underlying network protocols. Second, the attacker still should be able to carry out the original malicious intent of the attack.

1.3.2 Towards evaluation of NIDS in an adversarial setting

In order to evaluate the existing NIDS, in Chapter 2, we identify a few legitimate transformations such as modifying the delay between packets, splitting large packets into multiple smaller packets, etc. that can be applied on the malicious network traffic and can modify the feature values fed into an NIDS in a way to preserve the constraints we mentioned above. We show how a series of these transformations can be applied against three different recently proposed NIDS. Our evaluation shows that our attack is effective against all of them and can drop the detection rate of different attack categories by up to 70%.

1.4 Building a better NIDS without data labeling

As we mentioned, existing NIDS can hardly detect different network attacks in a low false alert setting. In addition, they are vulnerable to the adversarial example attack. Because of such issues that existing work deals with, a new NIDS should be designed in a way to be able to detect a variety of network attacks in a low false alert setting while demonstrating more robustness against adversarial examples.

1.4.1 Problems of the existing work

The first problem with the existing work is that they don't have any mechanism to remove the adversarial perturbations added by an attacker to the malicious inputs. Therefore, such perturbations completely remain effective against them. The second problem is that some of the existing work build hand-engineered features from the network traffic, which are tailored towards the detection of specific types of network attacks. Therefore while they might detect those attacks, they hardly can detect other network attacks and especially zero-day attacks. Finally, a big problem of the existing work is that their anomaly detection mechanism can't distinguish well enough between benign and malicious inputs. Therefore, a new method for training the anomaly detector component is required.

1.4.2 Enhancing robustness against adversarial examples in NIDS

To build an NIDS capable of detecting a wide range of network attacks with an enhanced robustness against adversarial examples, in Chapter 3, we present Reconstruction from Partial Observation (RePO). RePO is a technique that forces a model to reconstruct a whole input by only observing some parts of it by utilizing denoising autoencoders [101] and combining the inputs with multiple random masks. We show that using this technique, the model learns to distinguish better between malicious and benign inputs, which leads to a higher detection rate of different network attacks. Moreover, we show that the random masks we use in our NIDS make the model more robust against the adversarial examples because of making the model non-deterministic and also by removing some of the adversarial perturbations added to the adversarial inputs. Our evaluation conducted on a dataset with various network attacks shows our NIDS can improve detection of malicious traffic by up to 29% in a normal setting and by up to 45% in an adversarial setting compared to the existing work.

1.5 Building a better NIDS with data labeling

So far, we have mentioned how a better NIDS can be built in an unsupervised manner when we train a model without labeling any data. However, deep learning models demonstrated their full potential when trained in a supervised manner on a labeled dataset. Therefore, we complete this dissertation, by answering this question that how a better NIDS can be trained while labeling some of the network traffic in an efficient way and in a way to be still able to detect unseen attacks. Training a model in this setting has its own challenges. In the rest of this section, we first explain why it is challenging to do so and then we briefly describe our solution that can be used to address this problem.

1.5.1 Challenge

Labeling a dataset is expensive and time-consuming as it needs a human in the loop to label each instance of the data in the dataset. Furthermore, in a task such as anomaly detection in the network systems, there might be billions of packets or flows and we can't label everything. Therefore, at most, a small portion of the network traffic can be labeled and the model should be trained on it. While such a model might perform well in detecting network attacks included in the train set, it will struggle to detect other attacks, including zero-day attacks. This is because the zero-day attacks' characteristics might be different from those we trained a model on. However, this contradicts the initial goal we mentioned for using ML-based NIDS, which was to detect zero-day attacks.

1.5.2 General Domain Adaptation Through Proportional Progressive Pseudo Labeling

We propose to leverage domain adaptation for training a model in this setting. Domain adaptation is a technique that lets us transfer the knowledge gained from a labeled dataset to an unlabeled one. Several domain adaptation techniques are proposed for tasks such as image classification. Given a model trained on the labeled dataset, these methods increase its accuracy on the unlabeled one by reducing the gap between the source (labeled) and target (unlabeled) domains in the input space or some intermediate representation of the inputs. While showing promising results for image classification, as we will show, their performance is very limited when applied to other input types, including network traffic. As a result, in Chapter 4, we present Proportional Progressive Pseudo-Labeling (PPPL), a new domain adaptation method that generalizes better than existing ones across different input types. In a nutshell, PPPL mitigates the domain gap between the source and target domains while reducing the chance of alignment of target samples with a wrong class of source domain samples by assigning pseudo-labels to the target samples and prioritizing training on those, which are more likely to have correct pseudo-labels. Our evaluation shows that PPPL outperforms existing work in detecting unseen anomalies in network traffic by up to 58% based on the average F1 score.

Chapter 2

Towards Evaluation of NIDS in an Adversarial Setting

Signature-based Network Intrusion Detection Systems (NIDS) have traditionally been used to detect malicious traffic, but they cannot detect new threats. As a result, anomaly-based NIDS, built on neural networks, are beginning to receive attention due to their ability to seek out new attacks. However, it has been shown that neural networks are vulnerable to adversarial example attacks in other domains. Nevertheless, previously proposed anomaly-based NIDS have not been evaluated in such an adversarial setting. In this chapter, we show how to evaluate an anomaly-based NIDS trained on network traffic in the face of adversarial inputs. We show how to craft adversarial inputs in the highly constrained network domain, and we evaluate three recently proposed NIDS in an adversarial setting.

2.1 Introduction

Network attacks continue to grow in complexity, scale, and number [92]. The impacts of these attacks range from high monetary costs for exploited businesses and individuals to more serious issues, such as wide-scale power outages [40]. Due to the growing number of attacks, and their severe, adverse effects, companies are expected to invest billions of dollars by 2021 to find effective tools that detect and eliminate network intrusions [47].

Network intrusion detection systems (NIDS) are one part in the line of defense against network attacks. Historically, these are based on signatures – whether it be a known sequence of bytes (with deep packet inspection) or known and fixed access patterns. While providing a degree of protection for networks, this approach has a problem in that it relies on the signatures of **known** attacks.

In response, there has been a great deal of research. and even commercial offerings, that leverage machine learning (with deep neural networks) to augment the detection capabilities [7, 17, 62, 65, 104, 107, 108, 110]. These anomaly-based NIDS have been introduced due to their ability to detect zero-day attacks (for which there is no pre-existing signature) by looking for deviations from typical, benign network traffic. To do so, these NIDS are trained only on benign traffic. Then, during inference time, the NIDS measures how similar the new traffic is to the traffic seen during training time. Each packet or flow seen by the NIDS is given a similarity score and compared to a predefined threshold. If the packet or flow score exceeds the threshold, the traffic is considered malicious.

While moving toward deep neural networks for NIDS holds great promise, there is an underlying problem that has yet to be addressed, their vulnerability to adversarial examples. Previous work in other domains (e.g., image classification) has shown that neural networks are vulnerable to adversarial example attacks [12, 95], small perturbations of the input that can bypass or purposely alter the classification. In the case of images, this might be changing a few pixels (imperceptible to the human eye) such that the classifier misclassifies a specific person as a different person or hides that person all together. Unfortunately, we don't fully understand the implications in the context of NIDS because previously proposed, anomaly-based NIDS have not been evaluated in adversarial settings [7, 17, 62, 65, 104, 107, 108, 110]. The other downside of these anomaly-based NIDS is that they are evaluated on outdated datasets [17].

In order to address these issues, we introduce a technique to evaluate anomaly-based NIDS in an adversarial setting. We also perform an evaluation of previously proposed NIDS with this technique on a new dataset that contains 12 different network attacks. To do so, we needed to overcome some challenges not seen in other domains. When generating adversarial examples, we are constrained by two key factors: (i) we must retain the network protocol correctness, and (ii) we must retain the attack's semantics. Therefore, in this chapter, we illustrate how to craft adversarial examples for networks by identifying traffic manipulations that can change the network features but remain within the constraints above. In summary, we make the following contributions:

- For the first time, we explain how an adversary can legitimately modify network traffic in order to fool an anomaly-based NIDS and not break underlying network protocols.
- We show how these transformations can be tailored towards a packet-based NIDS, which predicts the malicious traffic in real-time by extracting features from each packet.
- We demonstrate how an adversary can fool a flow-based NIDS that detects malicious traffic based on the high-level features extracted from the whole flow by considering the legitimate transformations we introduce.
- We evaluate the NIDS mentioned above on a new network traffic dataset, which contains a wide range of attacks, to show how each of these attacks can be maliciously modified to fool an NIDS.

2.2 Anomaly-based NIDS

Anomaly-based NIDS can be built in many different ways. In general, for each input, they calculate a score and if that score is higher than a pre-defined threshold, they consider it as malicious. We categorize these anomaly-based NIDS into two different groups: packet-based NIDS and flow-based NIDS. A packet-based NIDS outputs a score for each packet that it receives. This decision is not necessarily based solely on the current packet; it can also consider the history of packets it has seen earlier. In contrast, flow-based NIDS make a decision based on features extracted from a whole flow and mark the whole flow as malicious or benign. In this section, we provide an overview of one previously proposed packet-based NIDS [65] and two flow-based NIDS [108, 110] and explain them in detail.

2.2.1 Kitsune

Mirsky et al. in [65] developed a packet-based NIDS, Kitsune. Kitsune has two main components: a feature extractor and an anomaly detector. On each packet arrival, the feature extractor extracts a behavioral snapshot of the hosts and protocols which communicated the packet. More specifically, upon arrival of a packet, the feature extractor extracts 115 features from 5 different temporal windows (100ms, 500ms, 1.5sec, 10sec, and 1min) that summarize all of the traffic

- originating from that packet's source MAC and IP address.
- originating from that packet's source IP.
- sent between that packet's source and destination IPs.
- sent between that packet's source and destination TCP/UDP socket.

In order to extract and analyze features for every packet efficiently, Kitsune maintains some internal state for each of the flows it sees. Upon each packet's arrival, it updates states related to that flow and calculates those 115 features based on the new state. Then, Kitsune feeds those features to the anomaly detector component. For anomaly detection, they use an ensemble of autoencoders (called KitNET), which attempts to reconstruct the extracted features. Finally, based on the reconstruction error, Kitsune decides whether a packet is malicious or not.

Mirsky et al. proposed the use of an ensemble of autoencoders in order to detect anomalies in an online manner. Doing so, made Kitsune efficient and runnable on lightweight hardware. However, an ensemble of autoencoders is not the only anomaly detection mechanism that can be used in this system. For example, as they did in their evaluation, the autoencoders can be replaced by a Gaussian Mixture Model (GMM), a heavy-weight, offline algorithm. As a result, we evaluate the performance of both Kitsune with autoencoders and Kitsune with GMM.

2.2.2 DAGMM

Zong et al. in [110] presented a new method for anomaly detection called Deep Autoencoding Gaussian Mixture Model (DAGMM). The authors implemented DAGMM to detect anomalies and illustrated its high effectiveness in detecting network attacks on the KDDCUP99 dataset [56], which contains features extracted from an entire network flow. Therefore, we evaluate the DAGMM method in the context of a flow-based NIDS.

DAGMM is a neural network consisting of two major components: a compression network and an estimation network. The compression network is a deep autoencoder and provides a low-dimensional representation of the input data points to the estimation network. The low-dimensional representation consists of reduced space and reconstruction error features. Based on this feed, the estimation network predicts their likelihood/energy in the framework of the Gaussian Mixture Model (GMM). In other words, given a sample x, the compression network computes its low-dimensional representation z as follows:

$$z_c = enc(x;\theta_e) \text{ and } x' = dec(z_c;\theta_d)$$

$$z_r = f(x, x')$$

$$z = [z_c, z_r]$$

where $enc(.)$ is the encoder part of the deep autoencoder and θ_e is its parameters. $dec(.)$ is the decoder part of the deep autoencoder and θ_e is its parameters. z_c is the reduced low-dimensional representation learned by the deep autoencoder and x' is the reconstructed version of x. z_r is the features derived from reconstruction error and $f(.)$ is the function for calculating these features. For example, $f(x, x')$ can be $[\frac{||x-x'||_2}{||x||_2}, \frac{x.x'}{||x||_2||x'||_2}]$ which is the relative Euclidean distance and cosine similarity, respectively. We used these two features in each experiment we did with DAGMM. Finally, z is what the compression network feeds to the estimation network.

Given z and K as the number of mixture components, the estimation network, which is another multi-layer neural network, predicts the mixture membership for each sample as follows:

$$p = MLN(z, \theta_m) \text{ and } \hat{\gamma} = softmax(p)$$

where $\hat{\gamma}$ is a K-dimensional vector for the mixture-component membership prediction, and p is the output of the estimation network parameterized by θ_m. Finally, they calculate the energy of each input in the context of GMM by using $\hat{\gamma}$. This energy shows the probability that a given sample is anomalous.

By building the model this way, they argued that they can train it in an end-to-end fashion which performs better than training autoencoder and GMM separately and they showed that DAGMM outperforms other state-of-the-art anomaly detection methods.

2.2.3 BiGAN

Zenati et al. in [108] described a method to detect anomalies by using a special kind of Generative Adversarial Networks (GAN) known as BiGAN [23]. Intuitively, the generator in a GAN is well-trained to fit the distribution of normal samples. As a result, it should be able to reconstruct samples x from the normal sample distribution with less reconstruction error compared to samples that are not from said distribution. However, in order to calculate this reconstruction error, we need to know the latent representations z corresponding to the inputs x. That is to say, if we know the latent representations corresponding to the input samples, we can feed those latent representations to the generator of the GAN and compare the reconstructed values with real input samples. Based on the reconstruction error, we then can decide whether that input is normal or an anomaly. The problem with a regular GAN is that finding those latent representations corresponding to the input samples is not an efficient process. Thus, in this paper, Zenati et al. suggest training a BiGAN for this purpose. To train a BiGAN, they simultaneously trained an encoder E that maps input samples x to a latent representation z, along with a generator G and discriminator D during training in a way to make $E = G^{-1}$. In this context the discriminator D receives $(x, E(x))$ and $(G(z), z)$ as its input during the training phase. Thus $E(x)$ results in the latent representation corresponding to the input sample x and $G(E(x))$ results in the reconstruction of the input sample. Finally, they defined the score function $A(x)$ as follows:

$$A(x) = \alpha L_G(x) + (1 - \alpha) L_D(x)$$

in which α is a hyperparameter. $L_G(x) = ||x - G(E(x))||_1$ measures the reconstruction error, and $L_D(x) = \sigma(D(x, E(x), 1)$ captures the discriminator's confidence that a sample is derived from the real data distribution and will be higher if the input is anomalous. σ is also the cross-entropy loss. Like DAGMM, the KDDCUP99 10 percent dataset was used to evaluate BiGAN's ability to detect network anomalies. As a result, we also evaluate BiGAN in the context of a flow-based NIDS.

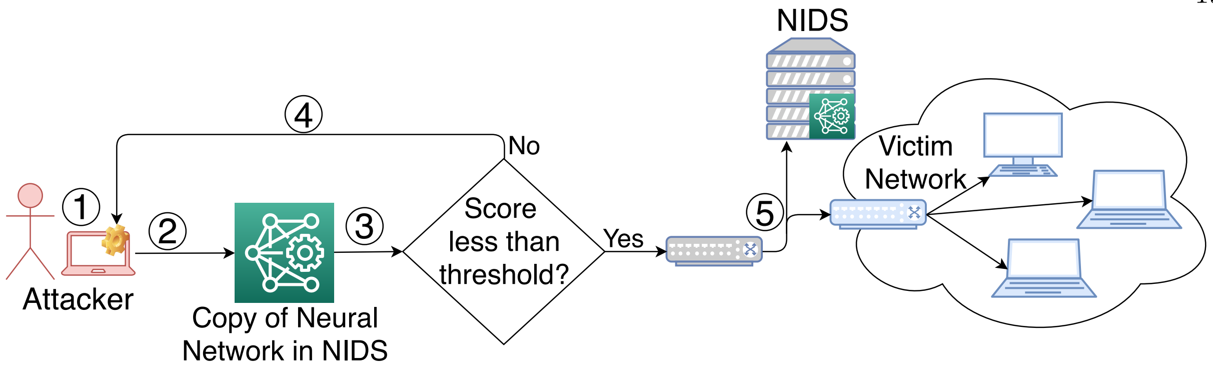

Figure 2.1: System overview and threat model considered when evaluating and designing anomaly-based intrusion detection systems. ①: The attacker sits outside the victim network and generates adversarial examples. ②: Adversarial examples are sent to the local copy of the NIDS for evaluation. ③: A classification score is produced by the NIDS based on the input. If the output score is greater than the threshold, the attacker applies some modifications, ④, to improve the adversarial example. This loop back process is carried out a maximum of `N` times. If the score in ③ is less than the threshold, the packet is mirrored to the NIDS and sent to the victim network ⑤.

2.3 NIDS in an Adversarial Setting

As we mentioned in the previous chapter, deep learning models are vulnerable to the adversarial example attack but existing anomaly-based NIDS that leverage such models have not been evaluated in an adversarial setting. Moreover, it is essential to know to what extent these NIDS are vulnerable to the adversarial examples as being vulnerable to such an attack can face the network systems with significant threats. Therefore, in this section, we discuss how we can evaluate the anomaly-based NIDS in an adversarial setting.

2.3.1 Threat Model

Before outlining our approach, we define the threat model we consider in evaluating anomaly-based NIDS. Figure 2.1 provides an overview of our threat model and system overview. In order to have a complete evaluation, we consider a white-box setting. That is to say, we consider that the attacker has a copy of the NIDS deployed on the victim network and knows all of its parameters. The NIDS deployed on the victim network receives a copy of all the packets that travel through the network entrances (⑤). We also consider that attacker's resources are limited to what they already used to create the original attack. In other words, to generate the adversarial version of a network attack, we assume the attacker does not want to use more machines or more IP addresses. The attacker also is considered to be outside of the victim's network (①).

2.3.2 Challenges in Crafting Adversarial Examples for NIDS

Crafting adversarial examples against NIDS that are trained on network traffic introduces its own complications and constraints. Thus, the crafting procedure needs to be tailored for NIDS. Here, we mention some of the differences between images and network traffic that prevent an adversary from fooling the NIDS with the same procedure used against image classifiers. First of all, pixels in an image can be modified freely. This is not the case for a sequence of network packets. For example, if features that are fed into an NIDS are packet headers, changing some of the headers could cause the communication between the attacker and the victim to breakdown. To this extent, during the crafting procedure, attackers should ensure that the communication channel does not timeout or breakdown. Second, pixels in an image can be modified independently of each other. This is not true for typical features fed into an NIDS. In many cases, these features are dependent on each other, and there is no guarantee that a valid network flow exists that matches the features generated by the crafting procedure. For example, a flow's average interarrival time between packets is directly tied to the flow's duration and number of packets through the following relationship: $Flow\_IAT_{avg} = \frac{Duration_{Flow}}{Pkt\_count_{Flow} - 1}$

As a result, we cannot arbitrarily change these features independently. We must ensure that the inherent properties of flows are not violated. In addition, all adversarial image pixels can be modified to fool an image classifier, but this is not the case for NIDS. Many of the features that are fed into them are extracted from the packets generated by the victim. These are packets that the attacker doesn't have control over. The differences between adversarial image generation and adversarial network traffic generation along with the security concerns that fooling an NIDS raise, demonstrate the need to explore how an NIDS can be evaluated in an adversarial setting.

2.3.3 Legitimate Packet Transformations

If we are able to manipulate the malicious packets of an attack to have specific features that mimic benign traffic, we will be able to bypass NIDS. We declare an attack a success if the manipulated attack packets meet the following three requirements.

(1) The packets must carry out their original malicious intent effectively (e.g. a port scan, after transformation, should scan the victim's ports).

(2) Packet transformations must not break the underlying protocols the attack relies on (e.g., a TCP-based attack cannot violate TCP).

(3) The attack must not be flagged as an intrusion by the anomaly-based NIDS. We will evaluate this requirement for existing systems in Section 2.5.

From these requirements and from studying the features used in existing anomaly-based NIDS, we identify three general packet manipulation techniques that can be used for crafting adversarial versions of network attacks.

The manipulations are as follows:

- **Split**: The attacker can increase the number of packets sent by splitting the original payload of each packet across multiple packets. For TCP, as long as sequence numbers, acknowledgment numbers, and IP IDs are updated properly, the attack remains effective as no information is lost and the packets are reassembled at the victim host.
- **Delay**: The attacker may adjust the time between outgoing packets by either increasing or decreasing the time elapsed between subsequent packets. Since the packets themselves are not modified, the attack will not only maintain its effectiveness (so long as there is not a connection timeout), but it will also adhere to the underlying network protocols.
- **Inject**: The attacker also has the ability to construct fake packets with arbitrary lengths, transmission times, and flag combinations. They can send the decoy packets among the real attack packets as long as they can ensure that these fake packets are ignored by the victim but processed by the NIDS. By doing so, an NIDS takes into account packets that

both reach and don't reach the victim into its decision on whether or not the current flow is malicious. The attacker can rely on the fundamentals of TCP, UDP, and IP protocols to guarantee these decoy packets are processed by the NIDS but not by the victim host. For example, the attacker can inject a TCP packet with a sequence number smaller than the ACK number acknowledged by the victim. Furthermore, by setting the TTL field of the IP header such that the TTL is greater than zero when processed by the NIDS but decrements to zero prior to reaching the victim, the attacker ensures the packet is dropped after reaching the NIDS but before the victim.

Therefore, in order to fool an NIDS which is trained on network traffic packets, the adversary should modify the malicious traffic with a set of legitimate transformations as described above. In the next section, we describe how we can use these transformations to attack several NIDS.

2.4 Crafting Adversarial Examples

In this section, we first explain how to tailor the legitimate transformations introduced in the previous section towards packet-based NIDS, and then move on to the flow-based NIDS.

2.4.1 Adversarial Examples for Packet-based NIDS

Algorithm 3 shows how we tailored legitimate transformations, introduced in the previous section, towards Kitsune. In a nutshell, Kitsune keeps some internal states for each flow and each packet moves through the network, updates the corresponding state. Then it calculates a score based on features extracted from the internal state to decide whether the current packet is from malicious traffic or not. In order to fool Kitsune, Each malicious packet that is sent from the attacker is fed through the local neural network copy and the output score is registered. If the packet's score is close to the threshold found during training time, we see if waiting a few moments can help reduce its score. More specifically, we implement the TryDelay procedure, which performs a binary search in the range between 0 and 15 seconds to see if adding a delay can bring the current packet score to less than $0.9 \times \mathit{threshold}$. In the case that the score is greater than the threshold, we also try splitting the packet.

The TrySplit procedure tries to convert a large packet into multiple smaller packets such that the score of all of them becomes smaller than the threshold. Since we don't know what the right cut-offs are to split the original packet, we search for the correct cut-off by trying different values. More specifically, we split the payload of packet with L bytes into two packets with r and $L - r$ bytes of payload, where r is chosen randomly. Since this cut-off might not be the right one, we need to back up the local NIDS state related to the current flow and restore it in case the split failed. When this happens, we try a different r. We need to checkpoint the NIDS's state to ensure that the state of local copy remains the same as the remote NIDS. If the first portion (r bytes) of payload could fool the NIDS, we would do the same thing for the second part (the remaining $L - r$ bytes) recursively until the whole packet's payload would be sent and none of them would be detected. Finally, if delaying or splitting the original packet could help fool the local copy, the attacker will make the appropriate change(s) and send the packet(s) to the victim. Otherwise, the original packet would be sent.

If the malicious packet is sent from the victim and its score is larger than the threshold, the only thing the attacker can do is to change the state of the NIDS such that the victim's packet does not pass the threshold. In this case, we see if injecting a fake packet from the attacker before the victim's packet can fool the NIDS for both packets such that the score of both of them becomes less than the threshold. More specifically, in the TryInject procedure, we send a packet from the attacker with different payload sizes. If that packet's score is less than the threshold, we send the victim's packet after that. If the score of both packets is less than the threshold, we inject that packet; otherwise, we restore the state of the local NIDS to the state before sending the fake packet. We repeat this for another fake packet with a different length. Also, since the TryInject procedure is a slow process, we run it occasionally. We keep track of the times that TryInject succeeds and fails for each attack. Then, for each new packet from the victim, we run the TryInject procedure with the probability of $\delta = \frac{\#successes}{(\#successes + \#failures)}$. After each success, we reset δ to one. In practice, this means that, given a network attack, if TryInject does not work for a while, we run it less frequently.

If suddenly it succeeds for a packet, we again try it on consecutive victim's packets more frequently.

Algorithm 1 Crating adversarial examples for Kitsune

```
procedure CraftAdvEx($x$)                                  ▷ $x$ is a malicious packet
    if $x$ is sent from the attacker then
        if $score_x > 0.9 \times threshold$ then
            TryDelay($x$)
            if $score_x > threshold$ then
                TrySplit($x$)
            end if
            Send the split packets with appropriate delay if successful.
        end if
    else                                                   ▷ $x$ is sent from victim
        TryInject($x$)
        Send the fake packet before victim's packet if successful.
    end if
end procedure
```

2.4.2 Adversarial Examples for Flow-based NIDS

In order to evaluate flow-based NIDS in an adversarial setting, we group the features fed into them into four different groups. As we mentioned earlier, manipulating the features fed to an NIDS in an adversarial manner is different from changing the pixels of an image. Here we consider two of the main differences. One difference is that some of the flow-based features can't be changed because the attacker doesn't have control over them since they are extracted from the victim's traffic. Also, some features depend on other features. For example, the mean of packet payloads in the forward direction can be calculated based on two other features, the total length of payloads in the forward direction and the total number of forward packets. There is another type of feature in which their value depends on the actual packets of the flow and cannot be calculated by the value of other features (e.g., std of packet payloads in the forward direction). As a result, we group flow features into the following four groups.

(1) Features that should not be changed because they are extracted from backward flowing packets (victim packets).

(2) Features that can be changed independently of each other by using legitimate transformations.

These include total forward packets, the total number of push flags in the forward direction, maximum packet interarrival time (IAT) in the forward direction, etc.

(3) Features whose values depend on the second group and can be calculated directly by a set of them.

(4) Features that cannot be directly recalculated based on independent features and a sequence of packets affect their values.

We tailored our adversarial crafting algorithm based on these four groups. We defined three masks that are the subset of each other. Each mask blocks a specific number of features from being updated by back propagating gradients through the models. The first mask only allows the procedure to modify the independent features (e.g., the second group). The second mask adds some of the 4th group features, and finally, the third mask adds all of the fourth group features to the set of modifiable features. In the crafting procedure, we first check whether we can fool the NIDS using the first mask. In the case of failure, we use the second and third masks. More specifically, the loss function we defined to minimize during the crafting procedure is as follows:

$$AdvLoss = F(x + \delta \odot mask_i)$$

where F is the model and $F(.)$ is the score predicted by the model. $\odot$ is the element-wise multiplication operator and δ is the perturbation that we want to find to add to the original features to fool the NIDS. By generating the adversarial features this way, we can be sure that applying legitimate transformations to the malicious flows will result in each feature from the first three groups matching the adversarial feature found.

However, the fourth group of features would have different values, and that can cause the overall flow to be detected by the NIDS. Therefore, in order to increase the chance of fooling the NIDS, in the crafting procedure, we do not stop the algorithm immediately after the score of a given sample drops below the threshold. To have a confidence interval, we continue to modify features in order to decrease the score further below the threshold. We considered this interval in order to compensate for the effect of different values between the fourth group of features and increase the chance of fooling the NIDS with the real sequence of packets.

Algorithm 2 demonstrates how we tailored the crafting procedure for flow-based NIDS. In this algorithm $threshold'$ is a smaller value than the real threshold of the NIDS to provide the confidence interval we discussed. Note that we start with a small learning rate to keep the modifications small and increase the learning rate exponentially in case of failure. The adversarial features we find with this algorithm against a given NIDS show the lower bound of the NIDS robustness. This is because, for some of the adversarial examples, there might not be a real sequence of packets that have those features.

Algorithm 2 Crating adversarial examples for Flow-based NIDS

```
procedure CraftAdvEx($x$)                                ▷ $x$ is a malicious flow
    for each mask $\in mask_1, mask_2, mask_3$ do
        for each lr $\in 0.001, 0.01, 0.1, 1.0$ do
            for each i $\in [0, totalIter]$ do
                take one step of GD with learning rate=lr
                $x' \leftarrow x + \delta$
                Recalculate group 3 features
                if $score_{x'} < threshold'$ then
                    return $x'$
                end if
            end for
        end for
    end for
end procedure
```

2.5 Evaluation

In this section, we evaluate the performance of the aforementioned NIDS in both a normal setting and an adversarial setting with the traffic manipulations described in Section 2.3. We first discuss the dataset used, then discuss the metrics used for our evaluation and finally, empirically demonstrate to what degree Algorithms 3 and 2 are effective in fooling different NIDS.

2.5.1 Dataset

To evaluate network intrusion detection systems, we used a highly cited dataset containing network traces of twelve network attacks from the Canadian Institute of Cybersecurity (CIC) [1] [88]. Sharafaldin et al. in [88] compared eleven available datasets based on eleven criteria and concluded that all of them have some shortages such as lack of traffic diversity and volumes, a limited number of attacks, etc. Therefore they built a new dataset that satisfies all of the eleven criteria.

The attacks are FTP-Patator, SSH-Patator, Dos slowloris, DoS slowhttptest, DoS Hulk, DoS GoldenEye, Heartbleed, Web attacks, Infiltration, Botnet, PortScan and DDoS. These attacks were carried out over a 5-day work week in a controlled environment. Each attack was implemented using popular network tools or was written in Python by the authors. We exclude web attacks from this dataset because in our evaluation, we only extract features from packet headers and in order to detect web attacks, packet payloads should also be inspected. In the remainder of this section, we briefly describe each attack and then we will provide some statistics about the dataset.

2.5.1.1 Network Attacks Description

FTP/SSH Patator: FTP and SSH-based, brute force, password guessing attacks were launched using the multithreaded Python tool, Patator [2].

GoldenEye: GoldenEye is an HTTP denial of service tool used to flood a victim's server with HTTP requests [86].

Hulk: Similar to GoldenEye, Hulk is an application-level denial of service tool. However, Hulk sends unique HTTP requests and obfuscates traffic by dynamically changing the user-agent and referrer fields in the HTTP header [91].

Slowhttptest/Slowloris: Low-bandwidth, application-layer denial of service attacks, Slowhttptest and Slowloris, aim to exhaust a server's connection pool by making many HTTP connections to the victim's server. The connections are kept open by the attacker via a slow transfer rate and incomplete requests, thereby draining the server's connection pool [19, 90].

[1] The dataset can be downloaded at https://www.unb.ca/cic/datasets/ids-2017.html

Heartbleed: Heartbleed is an attack over TLS that exploits a buffer overflow vulnerability in the OpenSSL library. Attackers send malformed heartbeat messages to the server and the server leaks private information back to the attacker [93].

Infiltration: An infiltration attack is typically carried out by exploiting vulnerable software. Here, a payload is downloaded to launch a backdoor, and then an NMAP port scan is executed [79].

Botnet: A set of internet-connected devices all controlled by a central leader to carry out various tasks (e.g., log a user's keystrokes). In this case, a remote shell is launched on the victim machines, and file uploads and downloads are executed [4].

DDoS: A distributed denial of service attack typically occurs when many devices, often orchestrated into a botnet structure, send many requests to the victim server in hopes to flood the bandwidth and/or resources of the server [3].

Port Scan: Port scans are used to gather information about a victim's host to uncover exploitable vulnerabilities [1].

2.5.1.2 Dataset Details

The CICIDS2017 [88] dataset contains flows extracted from packets files using the CICFlowMeter Tool [24]. The tool also extracts 80 behavioral flow features for each flow. The full list of features can be seen in Table 2.1. Each flow and its corresponding flow features were labeled as either benign or with the specific attack name, but the individual packets were not labeled. Thus, in order to evaluate the packet-based NIDS, we labeled packets as malicious or benign based on the information Sharafaldin et al. provided for this dataset. From the PCAP files provided within the dataset, we excluded IPv6 packets and labeled the other packets in the following way: for each attack, we labeled all of the packets sent or received between the attacker IP(s) and the victim IP(s) as malicious for the duration of that attack. All other packets were labeled as benign.

Features	**Fwd**	**Bwd**	**Flow**
Total Duration	✗	✗	✓
Total Packets	✓	✓	✗
Total Length of Packets	✓	✓	✗
Pkt Len Min/Max/Mean/Stddev	✓	✓	✓
IAT Min/Max/Mean/Sttdev	✓	✓	✓
Bytes/s	✗	✗	✓
Pkts/s	✓	✓	✓
PSH/URG Flags	✓	✓	✓
FIN/SYN/RST/ACK/CWE/ECE Flags	✗	✗	✓
Total Length of Headers	✓	✓	✗
Down/Up Ratio	N/A	N/A	✓
Avg Bytes/Bulk	✓	✓	✗
Avg Packets/Bulk	✓	✓	✗
Avg Bulk Rate	✓	✓	✗
Initial Window Bytes	✓	✓	N/A
Packets w/ payload >= 1	✓	✗	✗
Min. Packet Header Size	✗	✓	✗
Active Time Min/Max/Mean/Stddev	✗	✗	✓
Idle Time Min/Max/Mean/Stddev	✗	✗	✓

Table 2.1: Features extracted from flows for classifying network traffic with flow-based NIDS. ✓ and ✗ indicate whether or not the feature was calculated for packets moving in the labeled direction. "Flow" indicates features calculated taking into account packets flowing in both directions. Features were extracted using the CICFlowMeter Tool [24].

Table 2.2 also summarizes the contents of the dataset.

Set	Type	# of P	% of P	# of F	% of F
Train	Benign	11,680,917	100	529,481	100
Test	Benign	39,946,287	89.67	1,741,803	75.78
	FTP-Patator	110,736	0.25	7,935	0.35
	SSH-Patator	138,621	0.31	5,897	0.26
	DoS slowloris	47,586	0.11	5,796	0.25
	DoS slowhttptest	39,257	0.09	5,499	0.24
	DoS Hulk	2,245,526	5.04	230,124	10.01
	DoS GoldenEye	106,177	0.24	10,293	0.45
	Heartbleed	49,296	0.11	11	0.00
	Web Atks	39,823	0.09	2,179	0.10
	Infiltration	209,920	0.47	36	0.00
	Botnet	9,871	0.02	1,956	0.09
	PortScan	324,062	0.73	158,839	6.91
	DDoS	1,280,602	2.87	128,025	5.57
	All Attacks	4,601,477	10.33	556,628	24.22
	All	44,547,764	100.00	2,298,431	100.00

Table 2.2: The statistics of the dataset used for our evaluation. Column headers containing "P" contain packet information, while column headers containing "F" show flow information.

The whole dataset contains more than 56 million packets. We trained the packet and flow-based NIDS on the Monday traffic, which contains over 11.6 million benign packets (529,481 flows). The NIDS was then tested on the network traffic generated from Tuesday to Friday, which contains both benign and network attack traffic. This test set contains 12 different network attacks, which make up 10.33% of the overall packets and 24.22% of the overall flows.

2.5.2 Evaluation Metrics

True Positive Rate (TPR): TPR shows the ratio of malicious traffic that is detected as malicious to all of the malicious traffic when the model's threshold is fixed to a specific number.
False Positive Rate (FPR): FPR shows the ratio of benign traffic that is considered malicious to all of the benign traffic when the model's threshold is fixed to a specific number.

2.5.3 Performance in an Adversarial Setting

In order to see how each of the aforementioned NIDS detect adversarially modified network attacks, we chose their individual thresholds in a way to keep their FPR at 0.1 since those NIDS can detect most of the network attacks at this rate in a normal setting. In order to evaluate Kitsune, we used GMM as its detector because it could detect malicious traffic better than using the suggested ensemble of autoencoders. To fool this NIDS, we modified the malicious packets from the CICIDS2017 dataset with Algorithm 3. We fed all the packets into the NIDS, as in the normal setting, but due to the computational complexity of crafting adversarial examples, we only ran it on the first 25,000 packets of an attack.

In order to evaluate DAGMM, we used the code implemented at [69]. The model architecture and hyperparameters we used for this NIDS can be found in Table 2.3. To prevent the singularity problem during training, a small value (10^{-4}) was also added to the diagonal of GMM covariance matrix. For training BiGAN we used the code provided by Zenati et al. [108]. The model architecture and hyperparameters we used for this NIDS can be found in Table 2.4. To evaluate the flow-based NIDS in an adversarial setting, we used Algorithm 2 to find the adversarial features for malicious flows. Due to the computational complexity of this procedure, we only did it for the first 5000 flows of each attack in the cases where the attack contained more than 5000 flows.

Compression Network	FC(100)-FC(50)-FC(20)-FC(4)-FC(20)-FC(50)-FC(100)
Activation	Tanh
Estimation Network	FC(10)-Drop(0.5)-FC(4)
Activation	Tanh
Learning Rate	0.0001
Batch Size	1024
λ_1	0.1
λ_2	0.0001
Epoch	200

Table 2.3: The model architecture and hyper parameters used for training DAGMM on the flow-level features.

Operation	Units	Non Linearity	Dropout
E(x)			
FC	64	Leaky ReLU	0.0
FC	32	Linear	0.0
G(z)			
FC	64	ReLU	0.0
FC	128	ReLU	0.0
FC	77	Linear	0.0
D(x)			
FC	64	Leaky ReLU	0.2
D(z)			
FC	128	Leaky ReLU	0.2
Concatenate D(x) and D(z)			
D(x,z)			
FC	128	Leaky ReLU	0.2
FC	1	Linear	0.0
Optimizer	$\text{Adam}(\alpha = 10^{-5}, \beta_1 = 0.5)$		
Batch size	50		
Latent dimension	32		
Epochs	35		
Leaky ReLU slope	0.1		
w	0.1		

Table 2.4: The model architecture and hyperparameters used for training BiGAN on the flow-level features.

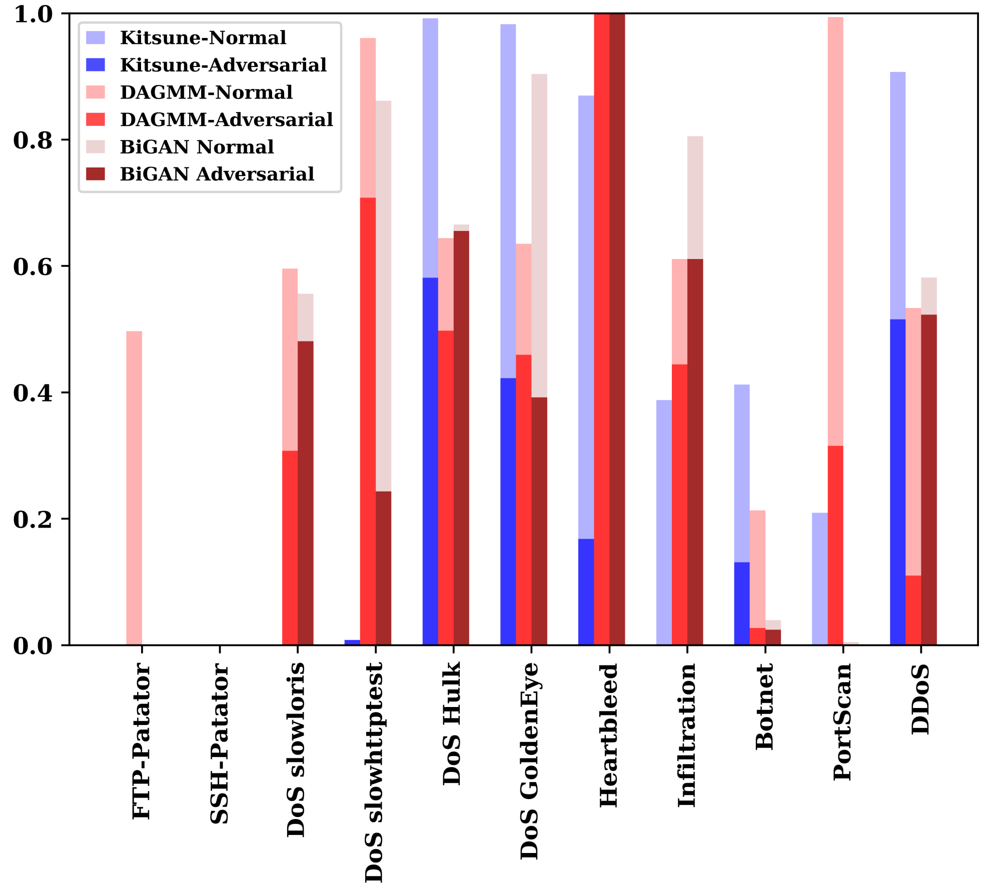

Figure 2.2: The TPR of different NIDS for each attack when FPR is 0.1 when sending normal traffic and its adversarial version.

The results of this evaluation are shown in Figure 2.2. For each NIDS considered, we show both the TPR under normal conditions as well as under adversarial conditions. As shown in this figure, for a packet-based NIDS, the detection rate drops by up to 70% (for Heartblead) in an adversarial setting and for flow-based NIDS, the detection rate drops by up to 68% (for PortScan). In fact, each NIDS performance decreases dramatically in most cases, indicating that these NIDS are not robust in the face of adversarial examples. More specifically, for Kitsune, the average TPR in an adversarial setting across all attacks drops to 16.6% from 43.6% in a normal setting; for DAGMM, it drops to 35.2% from 60.8%, and for BiGAN-based, it drops to 35.7% from 49.3%.

2.6 Discussion

As we saw, all of the existing NIDS are vulnerable to the adversarial example attack to some degree. One thing that is common among all of them is that they behave in a deterministic way. That is to say, the inputs that the attacker generates with the help of their local copy will fool the victim's NIDS if they fool the local copy as for the same input these NIDS output the same score. We know that the attacker can't query the victim's NIDS directly; otherwise, they would be detected. Therefore, one way to make NIDS more robust against adversarial examples is to make them behave stochastically. Then for the same input, the adversary's local copy and actual NIDS would output different scores. This adds extra hardship for making adversarial examples. Because in this case, fooling the local copy doesn't guarantee to fool the actual NIDS. In the next chapter, we show how making an NIDS less deterministic improves the NIDS's robustness against the adversarial examples.

Also, as we mentioned earlier, for the evaluation of flow-based NIDS, we found the lower bound of each NIDS's robustness. In order to find their exact robustness, we need a tool to modify packets in a flow to match the extracted features with those we found. We leave the production of this tool for future work.

Also, in previous work [33], it is discussed how to add a network forwarding element known as a traffic normalizer on the joint path of the NIDS and the victims to make sure that the NIDS processes the same packets that the victim does. Thereby limiting an attacker's ability outside the victim's network to inject fake packets into the stream by using bad sequence numbers, specific TTL values, etc. However, normalizers tend to introduce complications for regular traffic. A normalizer can raise the TTL value of packets it forwards to a pre-defined value to ensure the packet is not only processed by the NIDS but also by the destination. This can cause some of the packets to loop forever and consume a network's bandwidth if the normalizer is deployed on a loop. Also, normalizers can prevent some of the tools used for debugging in the network from functioning properly, such as traceroute. Furthermore, normalizers process each packet in-depth; therefore, adding latency into the communication channels. Extra latency can prove to be detrimental to real-time applications.

Due to the issues these normalizers introduce, they are not widely used, so we did not consider them in our evaluations. However, even by considering them, they will not affect the algorithms we introduced to modify traffic. The only difference is that only the delay and split transformations can be used to fool the NIDS in this situation.

2.7 Conclusion

In this chapter, we showed how to evaluate anomaly-based NIDS in an adversarial setting. We identified the legitimate transformations which an adversary can make to malicious traffic to fool the NIDS and not break the underlying network protocols. Finally, in our Empirical study, we showed our approach's effectiveness by tailoring the legitimate transformations towards both packet-based and flow-based NIDS. We found out that by using the transformations, we introduced in this chapter, the detection rate of an NIDS trained on the packet-level features can be dropped by up to 70% and the detection rate of an NIDS trained on the flow-level features can be dropped by up to 68%.

Chapter 3

Enhancing Robustness Against Adversarial Examples in Network Intrusion Detection Systems

The increase of cyber attacks in both the numbers and varieties in recent years demands to build a more sophisticated network intrusion detection system (NIDS). Because of the inability to detect zero-day attacks, signature-based NIDS, which were traditionally used for detecting malicious traffic, are beginning to get replaced by anomaly-based NIDS built on neural networks. However, as we showed, such NIDS have their own drawback, namely being vulnerable to the adversarial example attack. Moreover, they were mostly evaluated on the old datasets that don't represent the variety of attacks network systems might face these days. In this chapter, we present Reconstruction from Partial Observation (RePO) as a new mechanism to build an NIDS with the help of denoising autoencoders capable of detecting different types of network attacks in a low false alert setting with an enhanced robustness against adversarial example attack. Our evaluation conducted on a dataset with various network attacks shows denoising autoencoders can improve detection of malicious traffic by up to 29% in a normal setting and by up to 45% in an adversarial setting compared to other recently proposed anomaly detectors.

3.1 Introduction

The continuous growth of network attacks in number, scale and complexity [92] has caused a wide range of impacts to many individuals and exploited businesses. Network intrusion detection systems (NIDS) are one part in the line of defense against network attacks. Because of the technologies such as P4-based network telemetry, network function virtualization (NFV), cloud-native security services (such as what Zscaler provides) and network-wide view that the control plane in an SDN provides, these NIDS are getting more and more utilized to detect different types of threats [6, 55]. However, signature-based NIDS, which were traditionally being used to detect malicious traffic, can't cope with today's variety of network attacks as they are incapable of detecting zero-day attacks, which are significantly increased in recent years [43].

In response, there has been a great deal of research and even commercial offerings which leverage machine learning (with deep neural networks) to augment the detection capabilities [7, 17, 62, 65, 104, 107, 108, 110]. These anomaly-based NIDS have been introduced due to their ability to detect zero-day attacks (for which there is no pre-existing signature) by looking for deviations from typical, benign network traffic. To do so, these NIDS are trained only on benign traffic. Then, during inference time, the NIDS measures how similar the new traffic is to the traffic seen during training time. Each packet or flow seen by the NIDS is given a similarity score and compared to a pre-defined threshold. If the packet or flow score exceeds the threshold, then the traffic is considered malicious. This threshold should be set in a way to make sure that the NIDS doesn't generate too many false alerts on benign traffic.

Unfortunately, as we have shown in Chapter 2 the detecting capability of ML-based NIDS, can be significantly reduced by the help of the adversarial example (evasion) attack. This attack lets the attacker carefully and in many cases, slightly manipulate malicious traffic to fool and bypass the NIDS while carrying out the original malicious intent without breaking the underlying network protocols. The deterministic behavior of the previously proposed anomaly-based NIDS makes it easy to craft adversarial examples against them. In addition, minimizing the reconstruction error of the benign traffic in the training phase used by some of the NIDS [65] based on the full observation of the inputs can lead to an over-generalization problem. It means that the model learns to reconstruct the malicious traffic which was not trained on, as good as benign traffic, making it hard for the model to distinguish between them, leading to a low detection rate. Therefore, a new method for detecting malicious traffic is required to be more robust against adversarial example attack. Such a method should be able to detect a wide range of threats while generating a low number of false alerts. Because a high false alert rate significantly increases the required effort of security experts to manually sort through all alerts and differentiate real attacks from falsely identified attacks.

In this chapter, we present Reconstruction from Partial Observation (RePO) as a new method to build a more accurate NIDS in an unsupervised manner which is also more robust in the presence of adversarial examples by utilizing denoising autoencoders [101] and combining the inputs with multiple random masks before feeding them into the model. As we show, leveraging multiple random masks makes it harder to craft adversarial examples against the NIDS by making the model non-deterministic. Furthermore, it prevents the over-generalization problem we mentioned resulting in better distinguishment of malicious inputs from the benign traffic leading to a higher detection rate of the network attacks.

In summary, we make the following contributions:

- We leverage denoising autoencoders to build an NIDS that can detect malicious traffic better than previously proposed anomaly-based NIDS in a low false alert setting, which is also more robust in an adversarial setting.
- We show how a packet-based NIDS can be built with denoising autoencoders on top of raw values directly extracted from packet headers without any manual feature engineering. We also show how a flow-based NIDS can be built with them on top of manual features, calculated based on a whole flow.
- We evaluate our NIDS on a new network traffic dataset, which contains a wide range of attacks to show its effectiveness in detecting different types of attacks in both normal and adversarial settings.

3.2 Problems with an Existing Work

To build a better NIDS in an unsupervised manner, we first explore what is wrong with one of the existing work, namely Kitsune [65] in more details. Figure 3.1 illustrates a typical anomaly-based NIDS. These NIDS have two major components: a feature extractor and an anomaly detector. In a

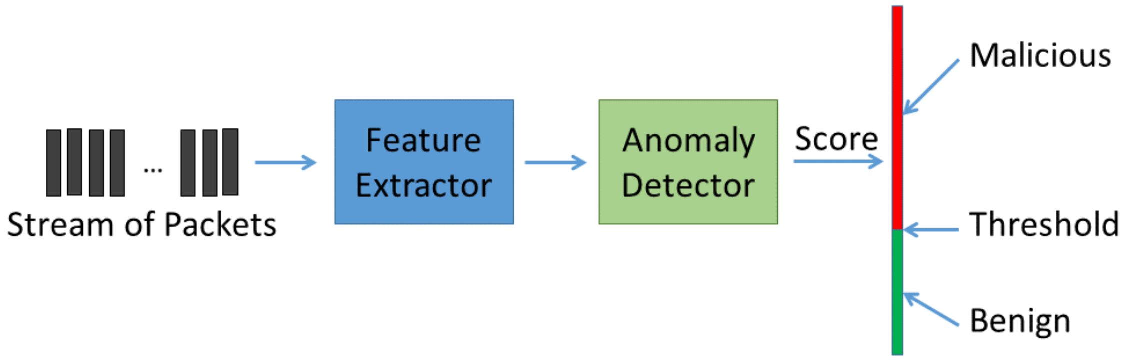

Figure 3.1: Illustration of the structure of a typical anomaly-based NIDS.

nutshell, the feature extractor receives a stream of packets and extract features from them to feed them to the anomaly detector. Then, the anomaly detector outputs a score for each input it receives that gets compared against a threshold. If the score is less than the threshold, the corresponding input will be predicted as benign otherwise, it'll be predicted as malicious. In the remaining of this section we want to see what are the underlying problems of the feature extractor of Kitsune as well as its anomaly detector.

Regarding the feature extractor component, we realized that the features extracted by Kitsune are manually designed to detect specific types of network attacks. More specifically, they are designed in a way to detect streams in which lots of packets are sent in a short amount of time. But, the problem is that hand-engineered features designed to detect specific types of attacks contradict our initial goal to detect zero-day attacks.

Regarding the anomaly-detector, we realized that the reconstruction error of the inputs, which is calculated with the help of autoencoders based on the full observation of the inputs, leads to an over-generalization problem. Ideally, we would like to get very small scores when we feed normal inputs to the model and very large scores when we feed anomalies. However, this is not the case when we use an autoencoder as the anomaly detector and we train it with the mean square error to force it to reconstruct the same input it sees. The problem is that reconstruction of the input based on full observation is too easy when the model has enough capacity. Because it just approximates the identity function. Therefore, the model learns to reconstruct anomalies as good as normal points.

That is to say, when we feed a normal point to the model, it reconstructs it pretty well. Therefore the score we calculate, which is the reconstruction error, would be pretty small. But the same thing happens for the anomaly points, leading to small reconstruction error for them, too.

In addition, in an adversarial setting, when the attacker modifies the network attack adversarially, there is no mechanism in Kitsune to remove the adversarial perturbations. Therefore, the modifications that the attacker designs will always be fully effective. Moreover, since Kitsune is a deterministic NIDS, when the attacker fools their local copy of NIDS they can be sure that the same input will fool the victim's NIDS because the local copy and the victim's NIDS behave the same.

Based on our understanding of the problems of the existing work, we design our NIDS. In the next sections, we first discuss how a better anomaly detector can be built. Then we show how we can design a more general packet-based feature extractor and combine it with our anomaly detector to build a better packet-based NIDS. Finally, we show that our anomaly detector can also be combined with a flow-based feature extractor to build a better flow-based NIDS.

3.3 Anomaly Detection with Denoising Autoencoders

In this section, we introduce our method for building an NIDS that can detect a wide range of network attacks while maintaining a low false alert level. We first show how a denoising autoencoder can be used as an anomaly detector. Then, we show how we can make its predictions more accurate and also more robust against adversarial examples. Finally, we show how it can be used as the core of an NIDS to build both packet-based and flow-based NIDS.

3.3.1 Reconstruction from Partial Observation

In order to solve the issue of over-generalization we mentioned in Section 3.2, we use denoising autoencoders to force the model to solve a more challenging problem. We train the model in a way to reconstruct a given input based on observing some parts of it. This way, the model has to not only reconstruct the visible parts of the input but also to generate the hidden parts of it. As a result, as illustrated in Figure 3.2, the reconstruction errors of the malicious inputs become larger than the

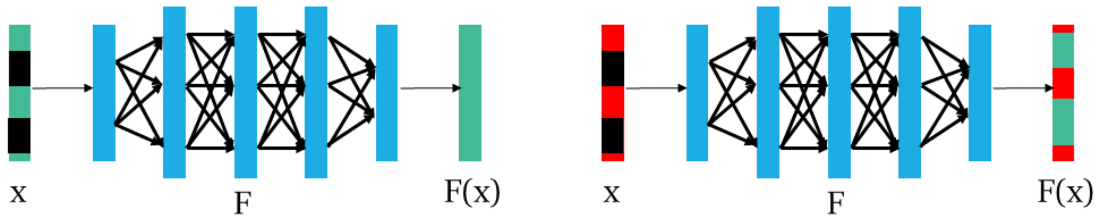

Figure 3.2: Illustration of the RePO technique. On the left, a benign input is fed to the model while some parts of it are hidden. Since the model has been trained on similar inputs, it can reconstruct the hidden parts. On the right, a partially visible malicious input is fed to the model. Since the malicious inputs were not in the train set, the model can't reconstruct the hidden parts well enough and fill in the hidden parts with some values similar to the benign inputs, leading to high reconstruction error.

benign inputs as the model can't reconstruct the hidden parts of the malicious inputs well enough, which leads to better distinguishment between malicious and benign traffic and a higher detection rate. Also, note that as malicious inputs get further away from the model's decision boundary (i.e. the threshold), it becomes harder to craft adversarial examples for them. Therefore, when we train the model in a way to make the gap between the malicious and benign scores larger, it also becomes more robust against the adversarial example attack.

We refer to this approach as Reconstruction from Partial Observation (RePO). More specifically, given a model F, we use the following loss function in our training phase:

$$Loss_{RePO} = \frac{1}{N} \sum_{i=1}^{N} ||F(x_i \odot r_i) - x_i||_2^2$$

where N is the number of inputs in the training set. x_i is the i-th input sample in the training set and r_i is a tensor with the same size as x_i. The elements in r_i are randomly 1 and 0 and the average percentage of 0s is δ, which is a hyperparameter that should be chosen with regard to the dataset the model is trained on (we set δ to 0.75 in all of our evaluations). In other words, we want to minimize the mean square of reconstruction errors. During inference time, we again mask some parts of the input randomly and then feed the result into the model. Finally, like any other anomaly detector we discussed so far, we need a score function. We define the score function as follows:

$$score(x) = \frac{1}{M} \sum_{j=1}^{M} |F(x \odot r)^j - x^j|^2$$

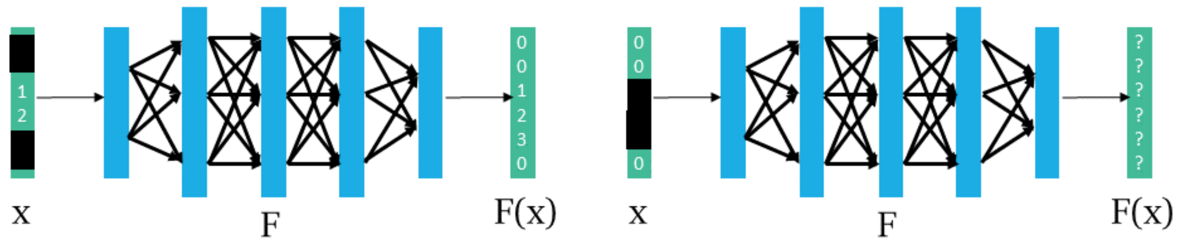

Figure 3.3: Illustration of the effect of "bad" masks. The model can reconstruct the input from what it observes on the left side. However, on the right side, all the informative parts of the input are blocked by random masks; therefore, the model can't reconstruct it from what it observes, which leads to a high reconstruction error.

where M is the number of features in x. x^j and $F(.)^j$ are the j-th features in x and its reconstructed version, respectively. Therefore, during inference time, if $score(x)$ is greater than a pre-defined $threshold$, x is considered an anomaly.

3.3.2 RePO+

In our approach, since r is a random matrix, by having different masks, the model outputs different scores. In such a setting, even if an input sample is normal, the random mask might block the most important parts of the input. Therefore, the model may not be able to reconstruct the input from what it observes. In this case, the reconstruction error would also be high. For example, assume that the normal samples have a pattern like $0, ..., 0, 1, 2, 3, 0, ..., 0$. Figure 3.3 illustrates how a random mask might cause a high reconstruction error even when the input is a normal sample. In this figure, on the left, the model can observe 2 and 1. Also, it knows that whenever there were 2 and 1 in the training data, they were followed by number 3 and other elements were zero. So based on what it observes, it can successfully reconstruct the input. However, on the right side, the random masks block all informative parts of the input. So what the model sees is a vector of zeros. Therefore, in this case, there is no way to reconstruct the input from what it observes, which leads to a high reconstruction error.

In order to solve this issue, as illustrated in Figure 3.4 during inference time, we replicate each

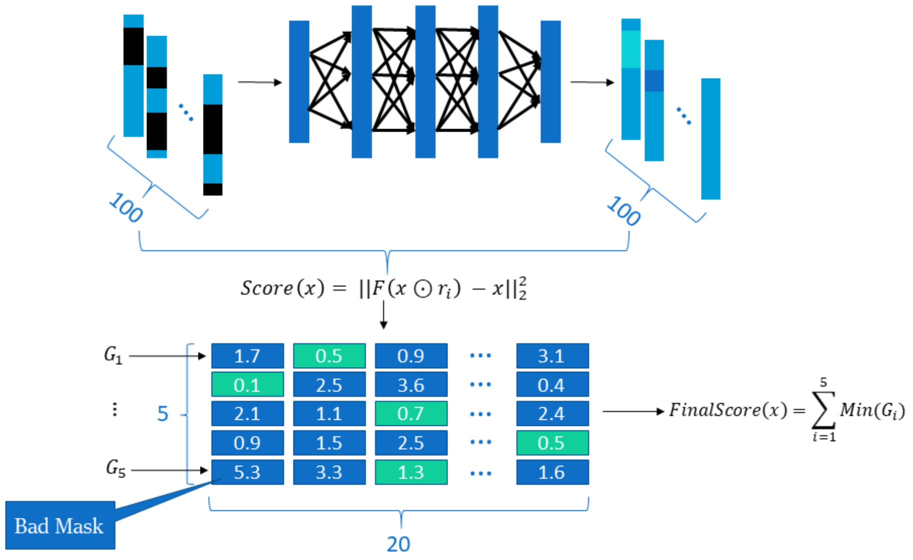

Figure 3.4: Illustration of the RePO+ technique.

sample 100 times and feed them in parallel to the model such that each of them is masked with a different mask r_i (we justify the choice of 100 masks in Section 3.5). We then calculate the score of all of them and group them equally into five groups, and from each group, we keep the minimum score. Finally, we calculate a new score by adding these five minimum scores together and we use this new score for deciding whether or not an input is anomalous. When we use 100 different masks, it becomes more likely to have "better" masks. By choosing the ones with smaller reconstruction errors, we essentially ignore the cases that have high error because of a "bad" mask (i.e., a mask that blocks the essential features of a sample needed to classify it correctly). Note that, here, we don't train an ensemble of 100 different models that can take a very long time to be trained. We only train one single model that receives multiple parallel copies of each input masked with different masks during inference time. We call this approach RePO+, and we empirically found that it has a higher detection rate.

Also, note that crafting adversarial examples against RePO+ becomes harder as for a given input, the adversary should plan to modify a larger set of features to fool the model compared to when there is only one mask. This is because if only a few features get changed, they can get masked by one of the masks with a high chance, which makes those changes ineffective or less effective. Nevertheless, such changes might not be feasible. Because for example, the adversary might not be able to find a set of transformations on the network traffic to generate those perturbed feature values. Moreover, since the adversary can't directly query the NIDS deployed on the victim's network (otherwise would be detected) and has to craft the adversarial example by the help of their local copy of the NIDS, Even if the adversary can fool the local copy when the sample goes through the original NIDS because of a different random mask, it would generate a different score which might be higher than the threshold and therefore will be detected. This stochastic nature of RePO+ also makes it more robust against adversarial examples. In Section 3.4.2, we empirically show the extent to which the model is more robust due to this property.

3.3.3 Building an NIDS with RePO

Here, we discuss how to build a packet-based NIDS using the RePO technique. To build a packet-based NIDS with RePO, instead of building hand-engineered features, we build our feature vectors directly based on raw values extracted from packet headers. More specifically, we first group the received packets by their sender and receiver IPs (i.e., the packets from A to B would be in the same group as packets from B to A). Then for each packet, we build a feature vector with the following features:

- Inter-arrival time: The time between this packet and the previous packet in the group.
- Features extracted from Ethernet header: the length of the frame.
- Features extracted from IP header: IP header length, IP length, IP flags (df, mf, rb), TTL.
- Features extracted from TCP header: source port, destination port, sequence number, acknowledgment number, TCP flags (res, ack, cwr, ecn, fin, ns, push, reset, syn, urg), TCP window size, urgent pointer. These features will be zero if the current packet is not a TCP packet.
- Features extracted from UDP header: UDP length, source port, destination port. These features will be zero if the current packet is not a UDP packet.

- Features extracted from ICMP packet: ICMP type. This feature will be zero if the packet is not an ICMP packet.
- Direction: This feature is a binary feature that shows whether the packet is from the IP, which started the communication or from the other end.

In total, the feature vector corresponding to each packet contains 29 features. In order to make a decision for a given packet, in addition to the features of that packet, we also send the features extracted from the previous 19 packets from that group to our model. Therefore the decision is made based on observing 20 consecutive packets, and we feed 580 features to the model in each case. If there are not enough packets before a given packet, we pad it with feature vectors, which all of their elements are zero. The model architecture we used to train RePO as a packet-based NIDS is a light-weight neural-network with only one hidden layer with 2048 neurons. During the training phase, we first normalized each feature separately by a min-max scaling approach. We set the batch size to be 512 and we trained the model for 30000 different batches, which were selected randomly with learning rate 0.001, 0.0001 and 0.00001 each for 10000 iterations.

Note, it is not required to utilize RePO only in the packet-based context. RePO can also be used to build a flow-based NIDS when combining it with a feature extractor that extracts features at the flow level. The model architecture we used to detect anomalies at the flow level is a fully-connected network with six hidden layers, each with size 256 and ReLU non-linearity, in addition to 2 dropout layers after the third and fifth hidden layers of the network. During the training phase, we first normalized each feature by a min-max scaling approach. We used a batch size of 256 and trained the model for five epochs with a learning rate of 0.001.

3.4 Evaluation

To evaluate our method, we used CICIDS-2017 [88], which is the same dataset that we described in the previous chapter. On this dataset, we evaluate how RePO performs in detecting network attacks in both normal and adversarial settings by comparing it against the same NIDS we evaluated in the previous chapter. Finally, we evaluate the system performance of our NIDS to measure the training time and its throughput in run-time.

3.4.1 Detection Performance of RePO in a Normal Setting

3.4.1.1 Packet-based NIDS:

In the packet-based context, we compare our method with Kitsune. As Mirsky et al. [65] showed in their paper, the ensemble of autoencoders in Kitsune can also be replaced with GMM. Here, in addition to comparing with Kitsune while it uses an ensemble of autoencoders (Kitsune-AE), we compare our approach with it when it uses GMM (Kitsune-GMM), as well. Figure 3.5 demonstrates how our NIDS performs in detecting different attacks compared to these baselines when the threshold of each NIDS was set in a way to make FPR be 0.01. With RePO+, we can detect 8 attacks with a TPR more than FPR whereas, Kitsune-AE and Kitsune-GMM could only detect 1 and 2 attacks, respectively. The average detection rate of our NIDS using RePO and RePO+ across all attack categories is 27.65% and 34.77%, while for Kitsune-AE and Kitsune-GMM, it is 5.71% and 0.44%, respectively. Thus, when using RePO+, our detection in the low false alert setting is almost 6 times better than Kitsune-AE (29% improvement) and 79 times better than Kitsune-GMM.

3.4.1.2 Flow-based NIDS:

In the flow-based context, we compared RePO with the DAGMM and BiGAN-based anomaly detectors. Figure 3.6 demonstrates how our NIDS performs in detecting different attacks compared to these baselines when FPR is low (0.01). With RePO+, we can detect 8 attacks with a TPR more than FPR, whereas, BiGAN and DAGMM could detect 7 and 5 attacks, respectively. The average detection rate of our NIDS using RePO and RePO+ across all attack categories is 21.61% and 25.49%, while for BiGAN and DAGMM, it is 15.05% and 4.91%, respectively. Thus, when using RePO+, our detection in the low false alert setting is 1.69 times higher than BiGAN and 5.2 times better than DAGMM.

Finally, note that there is a difference between the detection rates of packet-based NIDS and flow-based NIDS for some of the attacks such as FTP-Patator, PortScan, SlowHttpTest and

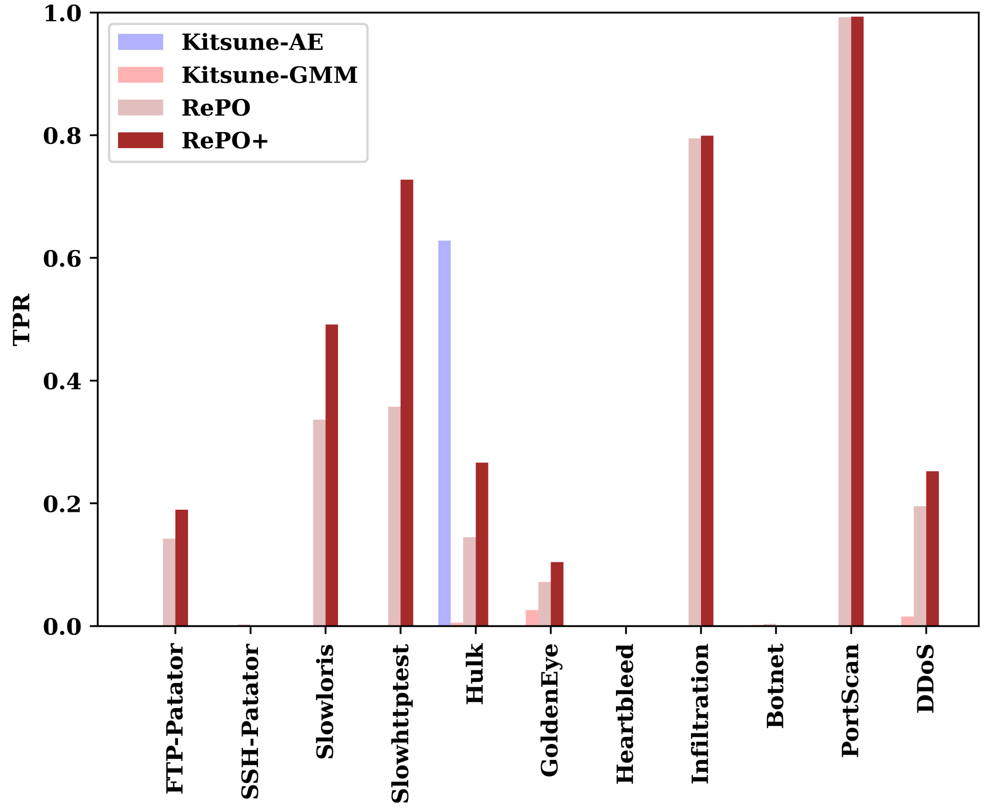

Figure 3.5: The TPR of packet-based NIDS for each attack when FPR is 0.01.

Heartbleed. This is because the features extracted for packet-based NIDS are different from the features extracted for flow-based NIDS. For packet-based NIDS, we only have individual values from packet headers, while features for flow-based NIDS are aggregated over the whole flow and calculated differently. In addition, feature vectors we created in the packet-based scenario are extracted from packets grouped by source IP and destination IP, whereas for the flow-based scenario, a flow is defined based on the 5 tuples (source IP, destination IP, source port, destination port and protocol). Furthermore, some of the attacks like Botnet are hardly detected by all of the NIDS, including ourselves, in this low false alert setting. This is also because of the way we build our feature vectors, as in both flow-based and packet-based cases, the feature vectors are created based on the packets sent between a single source IP and a single destination IP. However, Botnet could be detected better by looking into the traffic coming from multiple source IPs. A better feature extractor component

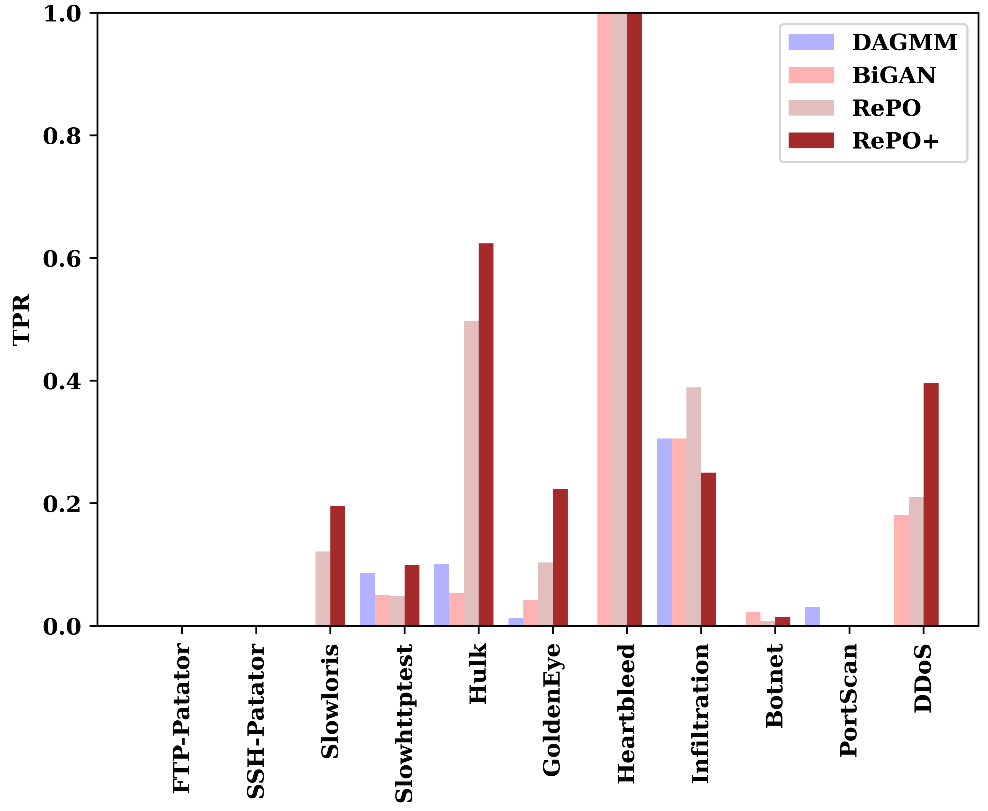

Figure 3.6: The TPR of flow-based NIDS for each attack when FPR is 0.01.

can help to detect such attacks better. We leave the designing of such a component for future work.

3.4.2 Detection Performance of RePO in an Adversarial Setting

In order to evaluate our approach in an adversarial setting, we use the crafting procedures introduced in the previous chapter. Also, we follow the experiment setting introduced in that chapter and set the threshold of our NIDS in a way to keep FPR at 0.1. This is because our baselines didn't perform well at the low FPR (0.01) even in a normal setting and there wasn't too much malicious traffic that is detected by those NIDS at the low FPR to craft adversarial examples for them. Therefore, for evaluation in an adversarial setting, we used a higher FPR (0.1) in which the robustness of different NIDS can be compared better against each other.

3.4.2.1 Packet-based NIDS

In order to evaluate the packet-based RePO+ in an adversarial setting, we tailored the crafting procedure introduced for Kitsune in Section 2.4.1 as follows: for each packet, if the packet is malicious and its score is more than the threshold, we first check whether the packet is sent from the attacker or the victim. If it was sent from the attacker, we first try to see if changing the delay between this packet and the previous packet or splitting this packet into multiple packets can fool the NIDS. If so, we make the appropriate changes and send the modified packet(s). If not, we check to see if injecting a fake TCP packet will fool the NIDS for both the fake packet and the current packet. If so, we send both of the packets; otherwise, we will only send the current packet and proceed to the next packet. If the current packet is sent from the victim, then we only check to see if injecting a packet before that works. In the case of injection, we let our algorithm bound the IAT between the current packet and the previous packet between 0 and 15 seconds (same as what is considered for Kitsune). Other features can be changed between their minimum values and their maximum values. For example, in the fake packet, any of the TCP flags or IP flags can be turned on. The source port and the destination port can also be any valid value. Note that fake packets are designed in a way such that they are not processed by the victim's machine but only by the NIDS. Also, because the crafting procedure is computationally expensive, we only applied it on the first 25000 packets of each attack.

Figure 3.7 demonstrates how well RePO+ performs in an adversarial setting compared to Kitsune-GMM. we only compare against Kitsune-GMM because GMM, as Kitsune's detector, could detect malicious traffic better than using an ensemble of autoencoders at this FPR. As can be seen, in the adversarial setting and even when accepting a higher FPR, Kitsune-GMM can only detect 5 different attacks at a rate higher than the FPR; whereas, our NIDS can detect 10 out of 11 attacks in the same setting. Also, in an adversarial setting, the average detection rate of Kitsune-GMM is 16.62%, while ours is 62.07% (3.73x better). On average, Kitsune's performance dropped 26.74%, while our performance only dropped 2.36%, an improvement over Kitsune of 11.33x.

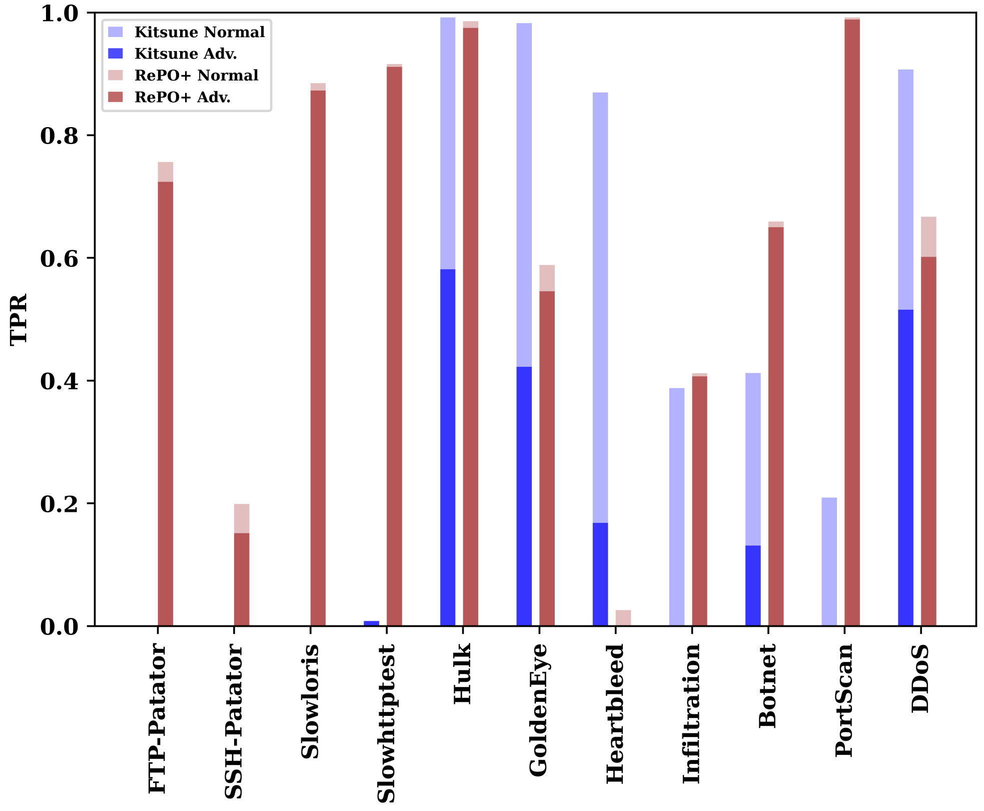

Figure 3.7: The TPR of RePO+ and Kitsune-GMM for each attack when FPR is 0.1 when sending normal traffic and the adversarial version of it.

3.4.2.2 Flow-based NIDS

In order to evaluate the flow-based RePO+ in an adversarial setting, we used the same procedure introduced in Section 2.4.2 for flow-based NIDS. Figure 3.8 shows how well RePO+ can detect different attacks in an adversarial setting compared to the other flow-based NIDS. The other NIDS that we mentioned had all deterministic behavior during inference time. For this reason, when the adversary crafts an adversarial version of a given flow for their own local copy, the exact same features can fool the NIDS deployed on the victim's network. However, as we mentioned earlier, RePO+ predictions are not deterministic. That is to say, the RePO+ output for the same set of features might be different between the adversary's local copy and the actual NIDS deployed on the victim's site. This gives the model more robustness and the results for this case are marked with "RePO+ Adv. Remote" on the figure. As can be seen, in an adversarial setting, and when we accept

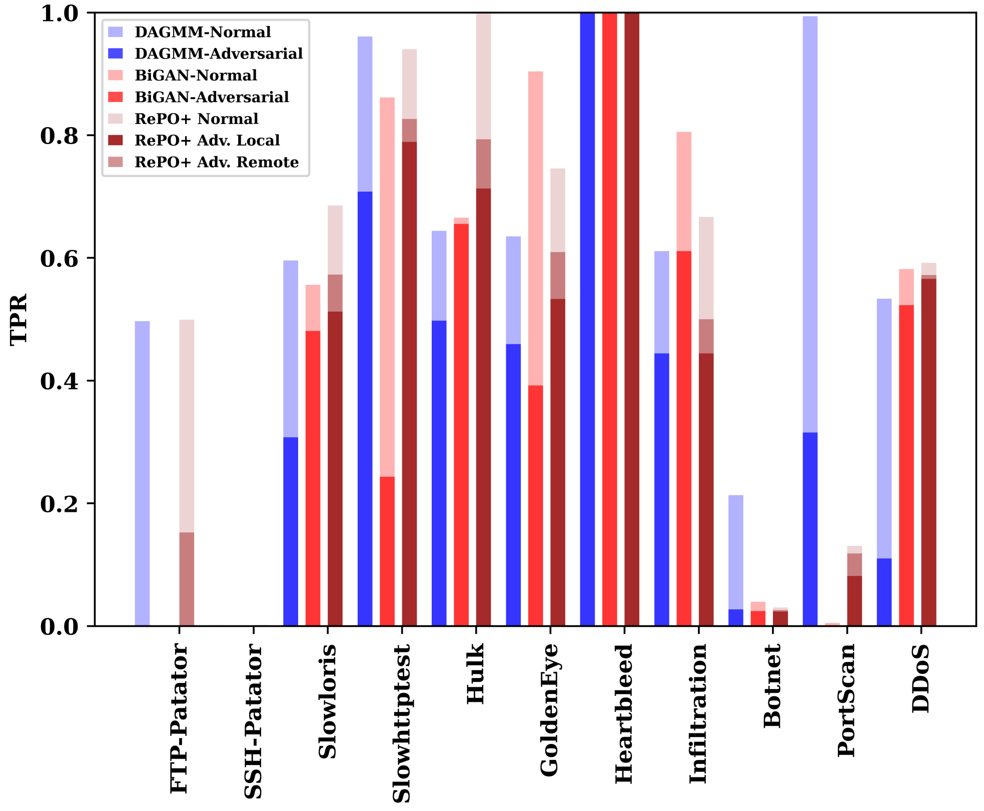

Figure 3.8: The TPR of RePO+, DAGMM and BiGAN for each attack when FPR is 0.1 and when sending normal traffic and the adversarial version of it.

a higher rate of false alerts, BiGAN and DAGMM can detect 7 and 8 different attacks at a rate higher than FPR. In comparison, our NIDS can detect 9 out of 11 attacks in the same setting. Also, in an adversarial setting, the average detection rate of BiGAN and DAGMM is 35.74% and 35.19%, respectively, while ours is 47.02% (1.3x better).

3.4.3 System Performance of RePO

Our NIDS is also time-efficient. In our experiments, by using an Nvidia Titan V GPU, it only took 25 seconds to train RePO+ as a flow-based NIDS. Moreover, During inference time, RePO+ could process 17,550 flows per second. That is to say, the whole test set, including 2.3 million flows collected over 4 days, could be processed in only 131 seconds. Also, as a packet-based NIDS, we could train RePO+ in 7.5 minutes and it could process 3,285 packets per second during test time.

This means that the whole test set containing more than 44 million packets could be processed in 226 minutes. Note that this processing rate is for when we classify every single packet, which is not necessary but in section 3.4.1.1 we did it this way to have a fair comparison against Kitsune, which outputs a score for every single packet. That is to say in our evaluation we considered a window with size 20, grouped 20 consecutive packets, fed them into the model and moved the window one step forward. We can simply move this window 20 steps forward to reduce the redundancy while still considering all of the packets. In this case, RePO+ can process 65,700 packets per second, making it capable of processing the whole test set in only 12 minutes. It's also worth saying that in this setting, we don't sacrifice the detection rate at all. The average detection rate when FPR is 0.01 is still 34.83% which is almost the same as what reported in 3.4.1.1. In addition, since RePO+ can become fully parallelized, we can use more GPUs to process a higher number of flows/packets per second if needed. In this case, the number of flows/packets processed per second will be multiplied by the number of GPUs used.

3.5 Discussion

As we mentioned earlier, in RePO+ we replicate each input 100 times and mask each one with a different random mask. The number of random masks in our method has an impact on two different things: The processing time during inference and the robustness against adversarial examples. If we decrease the number of random masks to half, then during inference, we can increase the number of packets we process in each second by a factor of 2, but the model becomes more vulnerable to adversarial examples. Figure 3.9 shows how decreasing the number of random masks impacts the detection rate of the model in an adversarial setting. In this figure, for four different attacks, we reported TPR of RePO+ in an adversarial setting when each input was masked with 10, 25, 50, and 100 different random masks. As can be seen, in all of the cases when we decrease the number of random masks, the detection rate also decreases.

Also, As discussed before, in order to evaluate our method in an adversarial setting we modified the malicious traffic with the adversarial example attack we introduced in Chapter 2. We

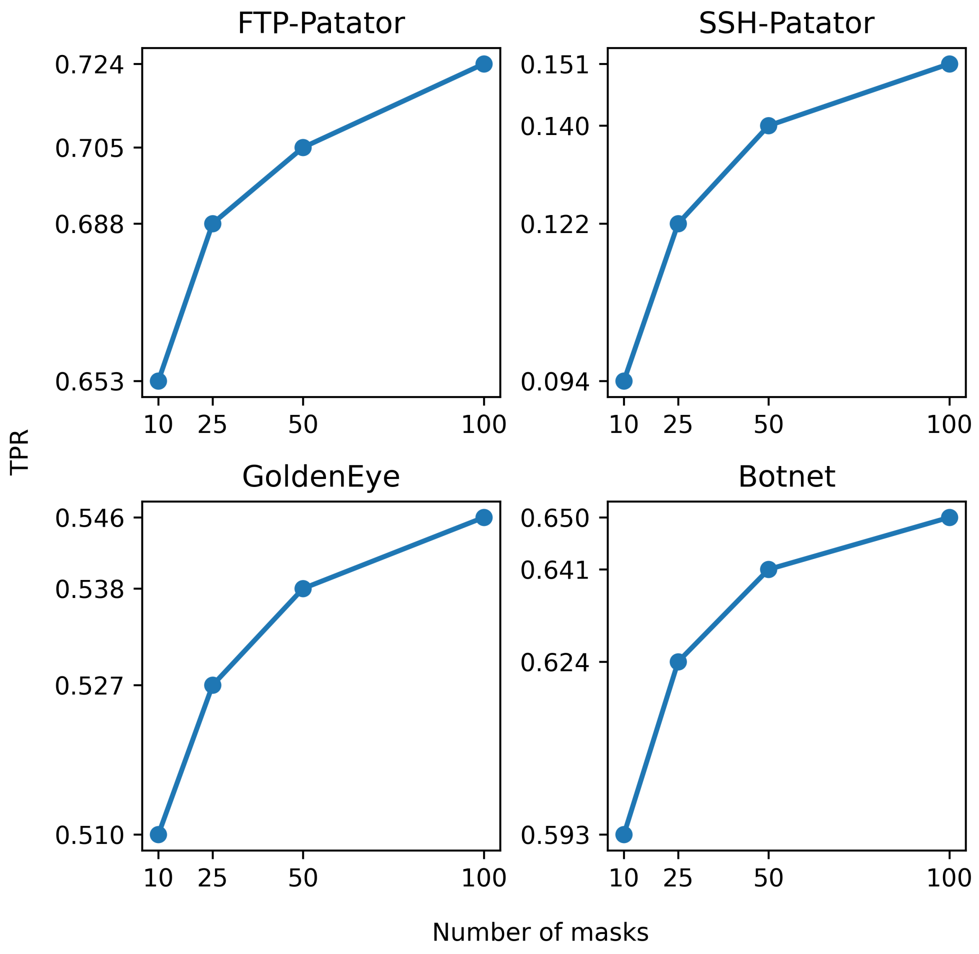

Figure 3.9: The effect of number of random masks used in RePO+ technique on the detection rate in an adversarial setting.

intentionally considered a white-box setting and a very strong attacker to have more confidence in the level of robustness RePO and RePO+ provide against adversarial examples. To the best of our knowledge currently, there exists no stronger adversarial example attack against NIDS in the literature. However, it should also be noticed that our evaluation doesn't guarantee to make a model robust against every possible adversarial example attack that might be designed in the future.

3.6 Conclusion

In this chapter, we demonstrated how denoising autoencoders can be utilized to build a more accurate NIDS than the current state of the art methods, which is also more robust against adversarial example attacks. As we showed in our experiments, on average, with our approach, we can improve the detection of different attacks by 29% in the packet-based context and by 10% in the flow-based context in a normal setting. Furthermore, in an adversarial setting, we can improve the detection rate by 45% in the packet-based context and by 12% in the flow-based context.

Chapter 4

Building a better NIDS with Data Labeling

While training deep neural networks in a supervised manner holds promise, labeling a huge dataset consisting of billions of packets to train a model on is not practical. In addition, supervised training on a subset of network traffic that only contains a few network attacks provides limited benefit to detect unseen attacks that might happen in the future.

Domain adaptation techniques help transfer the knowledge gained from a labeled source domain to an unlabeled target domain. Such techniques have the potential to make a model trained on a dataset containing a few network attacks to detect new types of anomalies that might happen in the future. During the past few years, different domain adaptation techniques have been published. One common flaw of these approaches is that while they might work well on images, their performance drops when applied to other input types, such as network traffic or text. In this chapter, we identify the problems of existing work and then we introduce Proportional Progressive Pseudo Labeling (PPPL) an effective technique to build a more general domain adaptation technique that can be applied on several different input types. At the beginning of the training phase, PPPL progressively reduces target domain classification error, by training the model directly with pseudo-labeled target domain samples, while excluding samples with more likely wrong pseudo-labels from the training set and also postponing training on such samples. Our evaluation conducted on the CICIDS-2017 dataset shows that PPPL can significantly outperforms other baselines on an anomaly detection task with up to 58% improvement based on average F1 score.

4.1 Introduction

In the previous section, we discussed how a better NIDS can be built without labeling any data. However, deep learning models have shown their full potential in other tasks such as image classification, sentiment analysis, etc., when they were trained on labeled datasets [18, 22, 38, 87, 94]. Nevertheless, training NIDS in a supervised manner is not a straightforward task.

The problem is that labeling a huge dataset consisting of billions of packets is expensive, time-consuming and needs a human in the loop. Ideally, we would like to label a small portion of network traffic that includes some network attacks and be able to detect most types of attacks, including unseen attacks in the future, without a need to label them directly. However, when we train a model this way, while such a model works well in detecting network attacks included in the train set, it doesn't work well to detect unseen anomalies that might happen in the future. This is because the characteristics of unseen attacks are different from those the model is trained on. In addition the input data distribution can constantly get changed and we might observe a domain gap between current data distribution and those we trained the model on. In such cases that we observe a domain shift in input data, the model can't do a good job in detecting different types of anomalies. [78, 106].

Unsupervised domain adaptation methods are used to address such problems where there is a domain shift between data distributions of a labeled source domain and an unlabeled target domain. Recently, many different domain adaptation methods have been proposed [26, 28, 39, 42, 50, 57, 59, 61, 75, 85, 103].

Despite different techniques used in recent approaches, one common flaw among them is that they don't generalize well across different types of inputs. Some of the methods such as [26, 39] are intrinsically designed for images, as they do image-to-image translation at pixel level or need specific data augmentation that should be applied to them at the pixel level. Therefore there is no straightforward way to apply these techniques to other input types such as text, network traffic or time-series. For the other approaches, they either leverage adversarial loss [27, 28, 59] or other techniques such as clustering [42]. One common problem is that there is no guarantee preventing the wrong alignment of the target samples. In other words, target domain representations from one class can get aligned with another class during the domain adaptation, leading to lower performance of the model. In addition, the complexity and the large number of hyper-parameters that some of these methods have weighs on this problem as there is no straightforward way to find the hyper-parameters that minimize the target error due to the lack of a labeled validation set for the target domain. Therefore more complexity leads to less generalization.

In this chapter, we introduce Proportional Progressive Pseudo Labeling (PPPL), a more general domain adaptation technique that works across different input types. PPPL assigns pseudo-labels to the target samples and trains the model directly with them. Key to PPPL is that it tries to minimize the number of target samples that will align with a wrong class by excluding uncertain samples from the training set at the beginning of the training procedure and progressively bring them back into the training loop with a weight proportional to their certainty. Further, we assume that we can guess the proportions of target samples that belong to each class. Note that while we don't know the individual labels in the target domain, the class proportions can be guessed in many cases. We consider three different scenarios with regard to this condition: First, we assume that target domain class proportions can be guessed accurately. Then, we evaluate our method while considering some different levels of error in the class proportions. Finally, we will relax this condition and demonstrate how we can modify our technique to work in a situation where there is no information about the target domain class proportions. While other domain adaptation methods don't consider this condition, we show that enforcing class proportions during the training further decreases incorrect alignment of target samples that results in further improvement of overall system performance.

Our experiments on the CIC-IDS2017 [88] dataset demonstrate that our approach is superior to other baselines on the anomaly detection task in the network traffic. Furthermore, experiments on four other datasets (Office-31 [82] and Office-Home [99], CIFAR-STL [20, 46], Multi-Domain Sentiment [14]) including tasks such as object recognition and text sentiment analysis show that our approach generalizes better than them across different input types. Our evaluation shows that while PPPL is capable of improving the accuracy of image classifiers on visual domain adaptation tasks as good as state-of-the-art methods, it significantly outperforms them on other tasks with up to 58% improvement for anomaly detection in network traffic based on the F1 score. Finally, an ablation study at the end of this chapter demonstrates the necessity of each component in our method.

4.2 Related Work

Recent methods that are proposed for domain adaptation leverage a wide range of techniques to make a classifier get better results on the target domain. Some of them, such as [26,39] are explicitly designed for images. Hoffman et al. [39] introduced CyCADA, an image-to-image translation method in which they combined generative adversarial networks with cycle-consistency constraints at the pixel level and semantic-consistency constraints at the feature level to reduce the domain shift between the source and target domains. French et al. [26] built their domain adaptation method on top of the mean teacher [96] idea and called it self-ensembling (SE). Basically, in addition to training a model with source samples and cross-entropy loss, two different stochastic transformed versions of each target sample are fed to two different models called student and teacher and the squared difference of their outputs is minimized. As French et al. described in their paper, for some datasets, the transformations they used for their stochastic data augmentation are highly tuned towards that dataset and therefore, can't be easily applied to other input types. This group of works is intrinsically designed for images and can't be applied to other input types in a straightforward manner.

The second group of methods are those that don't require being applied on a specific input type as their focus is to mitigate the gap between the source and the target domain in the feature space at some intermediate layer of a deep network. The adversarial domain adaptation is the basic idea behind a large portion of recent approaches [27,28,59], in which the classifier is trained jointly with a domain discriminator like a GAN [29]. The discriminator is trained to distinguish between source and target samples based on their representation captured from an intermediate layer of a deep network, while the classifier itself is trained in a way to fool the discriminator that results in generating domain invariant features. Ganin et al. [28] proposed Domain Adversarial Neural Network (DANN) in which they augment a classifier with a domain discriminator and train them in an adversarial fashion by back-propagating the reverse gradients of the domain classifier to learn domain invariant representations. Long et al. [59] designed Conditional Adversarial Domain Adaptation (CDAN) in which they condition the domain discriminator on the cross-covariance of domain-specific feature representations and classifier predictions, as well as on the uncertainty of the classifier to prioritize the discriminator on the easy to transfer samples. While we also model target domain uncertainty into our PPPL method, our approach differs from approaches like CDAN as they model the uncertainties into the domain discriminator. Doing so makes those approaches deal with the complications of training GANs, whereas in our approach there is no discriminator and we model uncertainty directly into the classification loss. This results in less complexity and greater generalization.

Beyond adversarial domain adaptation methods, there are also other approaches like Contrastive Adaptation Network (CAN). Kang et al. [42] mitigates the gap between the source and target domains at some feature space through an alternating optimization method in which they initially cluster target samples into multiple different groups with some complicated clustering method and then they assign some pseudo labels to them. They then train the model by minimizing intra-class discrepancy and maximizing inter-class discrepancy. Similar to CAN, we also assign some pseudo labels to the target samples, but unlike CAN, we don't use any clustering method. We instead assign the pseudo labels directly based on the model predictions. This, again, leads to less complexity and greater generalization. As we will show in the evaluation section, while these domain adaptation methods perform well in the image classification task, they don't provide the same level of benefit in other tasks such as sentiment analysis based on a text input or anomaly detection in network traffic.

Semi-supervised learning (SSL) methods [31, 51, 53, 83, 96, 100] are also used to train a model with a combination of labeled and unlabeled samples and have the potential to being used for detecting unseen anomalies in the network traffic. Many recently proposed SSL methods add a loss term, calculated based on the unlabeled data to make the model generalize better to unseen data. As discussed by Berthelot et al. [11], these SSL methods can be categorized into three different classes. Some of these techniques [31, 53] aim to minimize the entropy of the model's outputs on unlabeled data to make the model output more confident predictions on the unlabeled data. The second class of SSL methods [51, 83, 96] regularize the model to have consistent outputs when provided by different perturbed versions of the same unlabeled input. Finally, the last class of SSL methods [100] aim to make the model generalize better by leveraging some generic regularization techniques to prevent the model to overfit the training data. MixMatch unifies all of these techniques to benefit from all of them. In MixMatch, first, the model will be provided with K random perturbed versions of each unlabeled sample, and a unique probability vector will be "guessed" and assigned to all of those K inputs to achieve consistency regularization. Then, this probability vector will be sharpened by adjusting the "temperature" of this categorical distribution to minimize its entropy. Finally, to achieve generic regularization, for each batch of the labeled and unlabeled data points a new batch will be created in which each sample is built from a linear combination of two random samples from the original batch. In the end, the model will be trained with the labeled samples by using cross-entropy loss and with the unlabeled one by minimizing the L2 norm of models outputs and the "guessed" labels. While we also assign some pseudo-labels to the unlabeled samples during our training procedure, but we don't mix up the samples, and also in contrast to MixMatch we model the certainty of our guessed labels into the training procedure and postpone the training on the less certain samples. Furthermore, unlike MixMatch we directly reduce the domain gap between labeled and unlabeled samples in the logits space and have fewer hyperparameters. As we will show all of these differences make PPPL generalize better than MixMatch and get better results on the anomaly detection task in the network traffic.

4.3 Design Insights

Our goal is to train a deep network with the help of a labeled source domain in order to maximize the correct predictions on an unlabeled target domain. Formally, given a set of labeled samples known as source domain $S = \{(x_1^s, y_1^s), (x_2^s, y_2^s), ..., (x_{N_s}^s, y_{N_s}^s)\}$ such that $y_i \in Y = \{0, 1, ..., M-1\}$ and another set of unlabeled samples known as target domain $T = \{x_1^t, x_2^t, ..., x_{N_t}^t\}$ which come from two different data distributions our goal is to train a model $F : x \mapsto y$ to predict $\hat{y}_i^t \in Y$ to minimize the prediction error on the target domain. That is to say we want to minimize $\Sigma_{i=1}^{N_t} \mathbb{1}(\hat{y}_i^t \neq y_i^t)$ where y_i^t are the ground truth labels of the target domain samples. By doing so, we essentially can label a portion of network traffic containing a few network attacks and be able to detect new types of anomalies in the rest of the unlabeled traffic with the help of our approach. To do so, in the rest of this section, we first provide some insights that helped us to design our approach. Then, in the following section, we explain how we incorporate these insights to design PPPL.

4.3.1 Insight 1 - when training with pseudo-labels mean square error is a better choice

As we mentioned earlier, we want to assign pseudo labels directly based on model predictions to the target domain samples and train the model with them. For training, there are two choices here: We can feed the outputs of the final layer of the model to a Softmax function and train the model with the cross-entropy (CE) loss. We can also directly minimize the distance of the final layer outputs and the one-hot encoding of the labels with mean square error loss (MSE). We argue that for such a setting, MSE is a better choice. First, consider the Softmax function which is defined as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\Sigma_{j=0}^{M-1} e^{z_j}} \text{ for } i = 0, 1, ..., M-1$$

where M is the number of classes. Note that because of the nature of this function there is no one-to-one mapping between the probabilities (outputs of the Softmax layer) and the logits (inputs of the Softmax layer). That is to say, many different points in the logit space can be mapped to a single vector of probabilities. For example, when there is only 2 classes, both of the points $[1, 100]$ and $[200, 299]$ will be mapped to the $[\frac{1}{1+e^{99}}, \frac{e^{99}}{1+e^{99}}]$. This means that potentially, the points that belong to the same class can form multiple different clusters in the logit space. More specifically, the target domain samples and the source domain samples that share the same label or pseudo label can fall into different clusters. On the other hand, when we train the model with MSE, the points that have real or pseudo label C_i will fall into one single cluster very close to the point $[0, ..., 0, 1, 0, ...0]$ where the i-th index is 1 after we train the model on them. This characteristic is more desirable as it mitigates the domain gap between the source and the target samples at logit space and forces the model to learn features in the earlier layers of the network that leads to indistinguishable representations at the logit space between the source and the target domain samples. From another point of view, if we would have a domain discriminator to distinguish between the source and the target samples logits it could be completely fooled. Therefore by using MSE loss in this setting we get the same advantages of adversarial domain adaptation techniques without being worried about the complications of training GANs.

4.3.2 Insight 2 - training only on correct samples gradually reduces the target error

Consider a model pre-trained on the source domain. We argue that we can make the model predict more target domain samples correctly if we further train this model on the target domain samples that are already classified correctly. Also, we argue that if we do this technique for more iterations, we can gradually reduce the target error further.

To understand this better, consider a model pre-trained on the source domain. Also, as illustrated in Figure 4.1 consider some samples (e.g., A, B and C) from the target domain that belongs to the same class (e.g., class 1) and fall into close proximity of each other at some middle representation of the network. Assume that we assign pseudo-labels to these samples directly based on model predictions. Suppose that some of these pseudo-labels are correct and some are wrong (e.g. $\hat{y}_A = 1, \hat{y}_B = 2, \hat{y}_C = 2$). If we exclude wrong samples (B, C) and train the model only on correct samples (A), then the model learns that the points (B) near these points (A) are also more likely to be from the same class (class 1) and potentially some of them will get a correct pseudo-label in the next iteration. This effect gets propagated to the points near B (C) in the next iteration. Therefore, excluding wrong samples and training only on correct samples gradually reduces the target error.

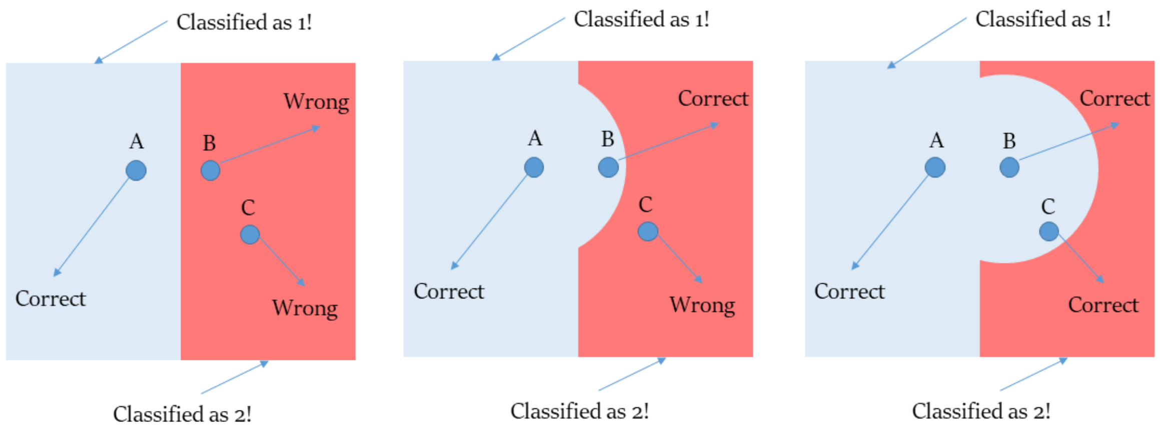

Figure 4.1: The illustration of training a model on the correct target domain samples. In the left figure, there are three target domain samples and only point A is predicted correctly. When we train the model on point A and label it as 1, the model learns that close proximity of A should also be classified as class 1 as illustrated in the middle figure and therefore, point B also will be classified correctly. Then this effect gets propagated to the points near B and therefore point C will be classified correctly in the next iteration as illustrated in the right figure.

To further show this, in Figure 4.2 we demonstrate the results of such training on four different domain adaptation tasks, namely Amazon → DSLR and DSLR → Amazon from the Office-31 dataset, Art → Clipart from the OfficeHome dataset, and Electronics → Books from the Multi-Domain Sentiment (MDS) dataset. As can be seen in all cases, the model progressively learns to predict target domain samples more accurately. These results are better than the results of any other domain adaptation approach by a large margin and it also generalizes well across different input types. In other words, we can significantly reduce the target error only by assigning pseudo labels to the target samples directly based on the model predictions and training the classifier with them. We just somehow need to determine in which cases the model is wrong to exclude them from the training procedure.

4.3.3 Insight 3 - an uncertainty metric can guide which predictions are wrong

Unfortunately, there is no straightforward way to know which of the model predictions are correct and which ones are wrong on the target domain as we don't know the target domain's labels.

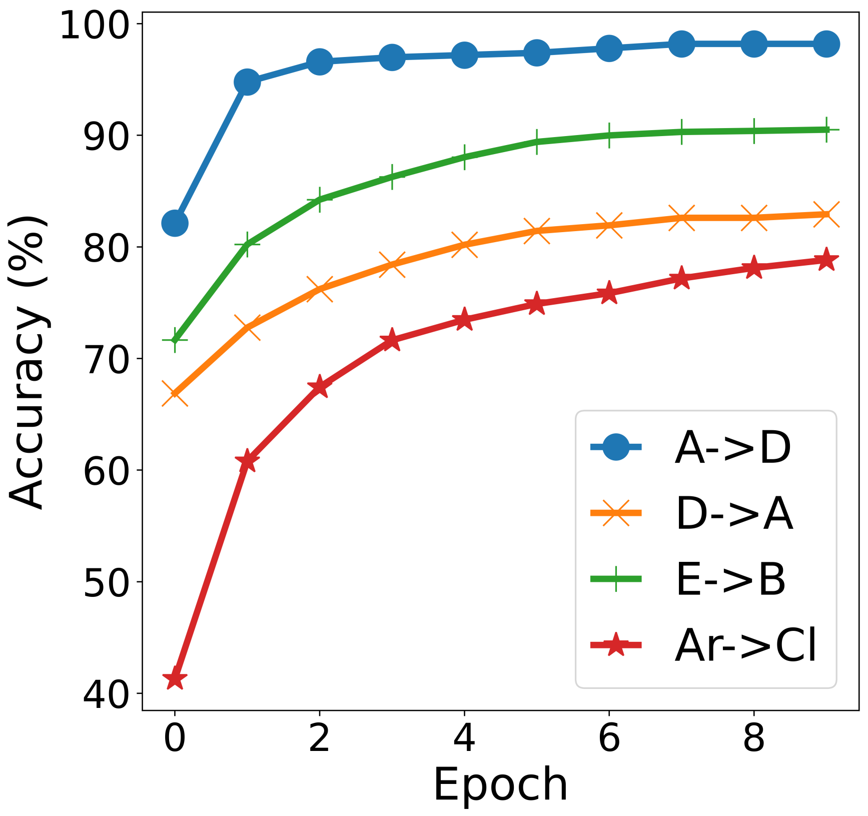

Figure 4.2: Model accuracy on the target domain when trained only on the correct predictions for four different domain adaptation tasks.

But, among all samples that are predicted as the same class, there is a relation between the model's certainty and the chance of wrong predictions. We capture the model's certainty with the difference between the two largest scores that the model outputs for each sample and call it the certainty score. This difference becomes smaller as in some intermediate representation of the inputs the points get further away from the same-labeled source points, falling into sub-spaces that are not well explored by the model or when they fall in close proximity to other points with different labels meaning getting closer to the decision boundaries. Thus, in such cases, it becomes more likely to get predicted wrongly.

This characteristic can also be seen in Figure 4.3. In general when certainty score decreases among samples that are given the same pseudo-label, a larger portion of predictions becomes wrong. For this figure, we first trained the model on the source domain for each of the aforementioned tasks

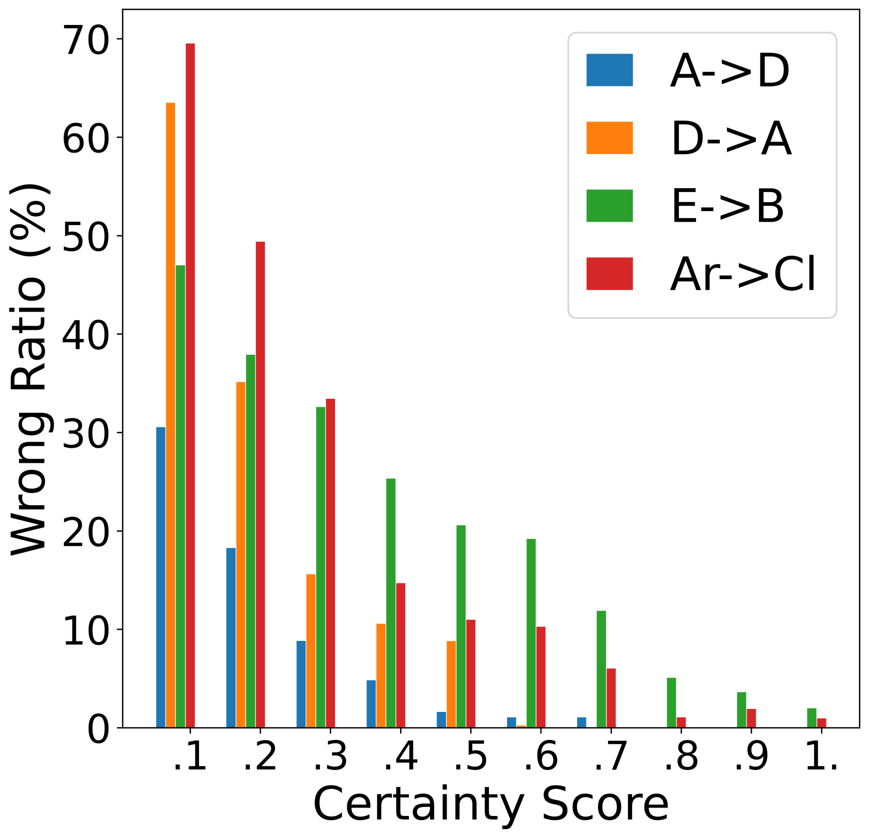

Figure 4.3: The ratio of wrong predictions at different levels of model certainty.

with MSE loss. Then for each class C the ratio of wrong predictions to all of the target samples that are predicted as C and their certainty scores fall into the interval $[\frac{i-1}{10}, \frac{i}{10}]$ is calculated. For each task, the i-th bar demonstrates the average of such ratio across all the classes.

4.3.4 Insight 4 - the timing of inclusion of a wrong-prediction matters

While we can predict better, we cannot know for sure which predictions are correct, and therefore it might be inevitable we assign some wrong pseudo-labels to some of the target samples and train the model on them. But one thing that is important is the time when we train the model on the target samples with the wrong pseudo-labels. We argue that the early inclusion of such samples into the training procedure deteriorates the model's performance on the target domain more than later inclusion. This is because of the same phenomenon that we discussed in insight 2: When

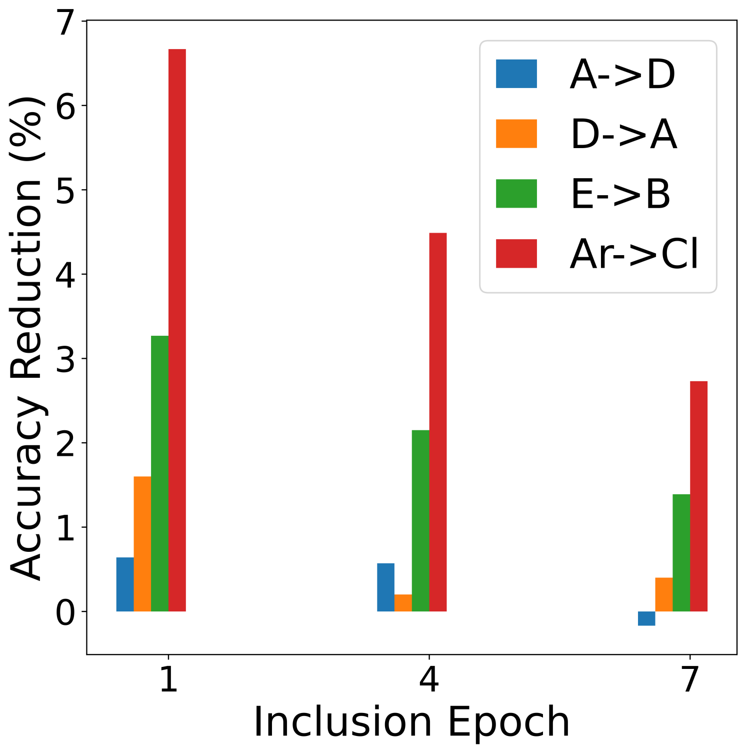

Figure 4.4: The negative impact of early training on wrong pseudo-labels.

we train a model on a target sample with a wrong pseudo-label, it would be more likely for the model to assign that wrong label to the points that are in close proximity of that wrong sample. Then, this wrong label propagates to the neighborhood of these newly affected samples in the next iteration. Therefore, the earlier we train the model on a wrong sample, the further its impact will propagate, the more model's accuracy deteriorates on the target domain.

This problem can also be seen in Figure 4.4. For this figure, for each of the tasks mentioned in the second insight, we trained the model the same way we discussed but also we included some samples with wrong pseudo-labels into the training procedure at different epochs (epochs 1,4,7 and 10). The size of the wrong pseudo-labeled samples in each task is equal to 10% of the target domain sample size. For all of the cases, we trained the model for 10 epochs. In this figure, we illustrate the change in the model accuracy when the wrong pseudo-labeled samples were included in the training loop at epoch 1, 4, and 7 in comparison with when they were included at epoch 10. Therefore a larger bar shows a greater decrease in the model's accuracy. As can be seen, the earlier the model got trained on the wrong samples the more its accuracy is decreased. For example, for the Art $\rightarrow$ Clipart task if we include the wrong pseudo-labeled samples at the first epoch of training the final accuracy will be almost 7% lower than when we postpone their inclusion to epoch 10.

4.4 Proportional Progressive Pseudo Labeling (PPPL)

Based on the insights we discussed we designed our approach. In a nutshell, based on the second insight we know that if we use a model pre-trained on the source domain and assign pseudo-labels to the target samples with it and exclude the wrong pseudo-labels the model progressively gets better. Also, based on the first insight we know that for such a setting using MSE loss is better than CE loss. Unfortunately, since we don't know target labels we can't find out exactly for which cases the pseudo-labels are wrong but based on the third insight we know that the ratio of wrong predictions increases as the model certainty decreases. In addition, based on the fourth insight we know that it is better to postpone training the model on such samples.

Algorithm 3 Proportional Progressive Pseudo-Labeling

```
1:  procedure PPPL($F, X_s, Y_s, X_t, CP_t$)
2:      for $i \leftarrow 1$ to 45 do
3:          $N \leftarrow 10 + 2 \times i$
4:          $S_t \leftarrow F(X_t)$
5:          $PL_t \leftarrow argmax(S_t)$
6:          $CS_t \leftarrow CalcCertaintyScore(S_t)$
7:          $W_t \leftarrow CalculateWeight(CS_t, PL_t, N)$
8:          $X_t', Y_t', W_t' \leftarrow Exclude(X_t, PL_t, W_t, CP_t)$
9:          $X_s', Y_s', W_s \leftarrow Select(X_s, Y_s)$
10:         $Train(F, X_s', Y_s', W_s, X_t', Y_t', W_t')$
11:     end for
12: end procedure
```

Algorithm 3 describes our approach in more detail. The inputs are F which is the model pretrained on the source domain, X_s which is the set of all source domain samples, Y_s which is the set of all the source domain labels, X_t which is the set of all target domain samples and CP_t which is the set of target class proportions that are guessed or known from other sources. We first train the model (F) on the source domain samples (X_s, Y_s) with the MSE loss function (based on insight

1). Then in each iteration of the algorithm, we first get the score which is a vector with size M (M is the number of available classes) for each of the target samples (line 4) and assign a pseudo-label to that sample based on its largest score. (line 5). Then for all of the target samples, we calculate the "certainty score" (line 6) and then assign a weight value to each of the target samples based on its certainty score (line 7). This weight will be used later during training to control the impact of each sample on the model parameters.

The function $CalculateWeight(CS_t, PL_t, N)$, first groups all of the samples that are assigned the same label. Then, for each group, it assigns a weight between $[0.2 - 1.0]$ to $N\%$ of the samples and 0 to the rest of the samples of that group. Within each group, the weights are monotonically assigned based on the certainty scores such that a sample with a larger certainty score will be assigned a larger weight. More specifically, if the number of samples that fall into top $N\%$ for a given group is L_c then the weights for those samples are calculated as follows: $w_j = \frac{1}{t_j}$ where $t_j = 1 + \frac{4}{L_c} \times j$ in which $j \in [0, L_c)$ and w_j is assigned to the j-th sample with the largest certainty score.

For better illustration, in Figure 4.5 we show the weights that will be assigned to the target samples with the same pseudo-label at epoch 1, 20 and 45 of our method. We assumed that the size of this group would remain in 1000 during the whole training. As can be seen, for the first epoch only 10% of the samples will be assigned with a weight of more than 0, which essentially means that the other 90% samples that have a lower certainty score will be excluded from training in the first epoch. At epoch 20, 50% of the samples will be included in the training and at the final epoch, all the samples will be included. Using this weighting strategy decreases the chance of training on wrong samples on the early epochs by excluding the less certain samples (designed based on insights 3 and 4). Also for the samples that will be included at each epoch, we assign a smaller weight to the less certain samples to decrease the impact of those that are assigned wrong pseudo-labels and potentially are among included samples on the model parameters (designed based on insight 3).

There is also another way we try to exclude wrong samples which is based on guessed class proportions (line 8). In the *Exclude* function, for each label, we first calculate the ratio of samples

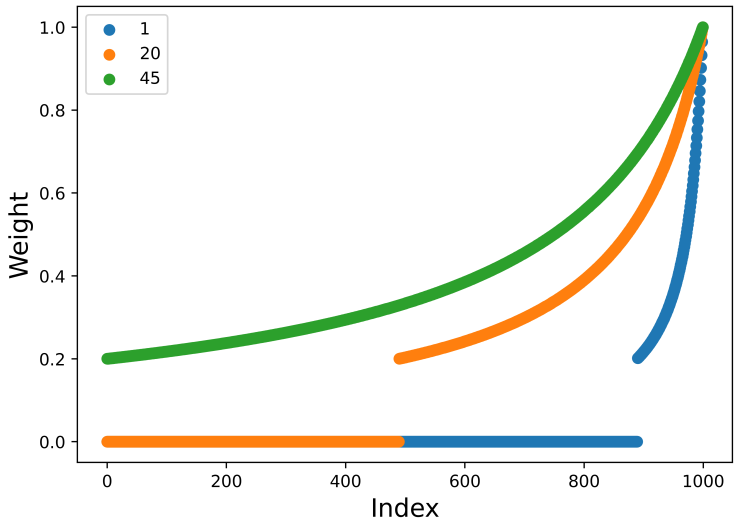

Figure 4.5: The illustration of weights assigned to the target samples with the same pseudo-label.

with that pseudo-label to the whole target domain sample size. Then we exclude some samples for classes that their ratio is higher than its corresponding guessed ratio because it means that the model predicted samples with that label more than what we expect. Therefore, we expect that some of these predictions to be wrong. So, from each class, we keep excluding the most uncertain predictions until the ratio of remaining samples becomes equal to our expected ratio (designed based on insights 2 and 3).

In addition to the target samples, we also select some samples randomly from the source domain at each epoch to train the model on them. We do this because in the initial phases of our algorithm we only include a small portion of the target samples and we don't want to make the model over-fit on them. We also assign a weight equal to 1 to these source samples as all of their labels are correct. Finally with the combination of pseudo-labeled target samples and these source samples we train the model with MSE loss (designed based on insight 1). More specifically, the loss function we use for training is as follows:

$$Loss_{PPPL} = \frac{1}{N}\sum_{i=1}^{N} w_i||F(x_i) - y_i||_2^2$$

in which y_i is the one-hot encoding of (pseudo)label assigned to sample x_i and N is the total number of included samples at the current epoch.

4.5 Evaluation

In this section, we first evaluate how well PPPL can detect unseen anomalies in network traffic. We first demonstrate how well PPPL can perform when the target domain class proportions can be guessed accurately and then we show how this assumption can be relaxed. In addition, to show how well PPPL can generalize across different input types we report its performance on four other datasets.

4.5.1 Anomaly Detection Performance of PPPL in the Network Traffic

4.5.1.1 Dataset

To evaluate how well PPPL can detect unseen anomalies in network traffic, we used CICIDS-2017 [88], which is the same dataset we used for the evaluation of packet-based NIDS in the previous chapters. We built three different subsets from this dataset such that each one consists of different types of network attacks as follows:

- **Domain A:** FTP-Patator, SSH-Patator.
- **Domain B:** DoS Slowloris, DoS Slowhttptest, DoS Hulk, DoS GoldenEye, Heartbleed.
- **Domain C:** Web attacks, Infiltration.

We consider the traffic collected in each of these datasets as one domain and define 6 domain adaptation tasks between each pair of them as follows: A→B, A→C, B→A, B→C, C→A and C→B. In order to classify the packets in this dataset, we first preprocessed them with the same method described for building a packet-based NIDS in Chapter 3. More specifically, we first grouped packets based on their source and destination IPs. Then from each packet, we extracted some features from its Ethernet header, IP header, TCP header and UDP header. We also extracted inter-arrival time (the time between the current packet and the previous one in the same group) and direction (either the packet is from the sender or receiver) of each packet. In total, we extracted 29 features for each packet. Finally, within each group, we concatenated the feature vectors of every 20 consecutive packets together to feed them to the classifier. Given this preprocessing method, the A, B and C domains contain 573,544, 685,241 and 462,031 samples, respectively. Also, all of these domains are imbalanced. That is to say, a large portion of the packets in each of them are benign and only a small portion is malicious. More specifically, the percentage of malicious traffic in the A, B, and C domains are 2.2%, 18.2% and 2.7%, respectively. Therefore, for evaluation we report the F1 score which equals to $2.\frac{precision.recall}{precision+recall}$.

4.5.1.2 Baselines

We compare PPPL with three different baselines: CAN [42], CDAN [59] and MixMatch [11]. CAN and CDAN are among the best domain adaptation techniques to the best of our knowledge and MixMax is one of the best semi-supervised learning methods which has been recently published. More specifically, CAN is state-of-the-art for the Office-31 [82] dataset. Kang et al. in their evaluation showed that CAN outperforms many other baselines such as Domain Adversarial Neural Network (DANN) [27, 28], Joint Adaptation Network (JAN) [61], Multi-adversarial Domain Adaptation (MADA) [75], Deep Adaptation Network (DAN) [58] and Self Ensembling (SE) [26]. Also, CDAN is among the best adversarial domain adaptation techniques that we are aware of. Long et al. in their evaluation showed that CDAN outperforms many other methods such as DAN [58], Residual Transfer Networks (RTN) [60], DANN [27,28], Adversarial Discriminative Domain Adaptation (ADDA) [98], JAN [61] and CyCADA [39]. Finally, Berthelot et al. [11] showed that MixMatch can outperform several other semi-supervised learning methods such as Π-Model [52, 84], Mean Teacher [96], Virtual Adversarial Training [66], Pseudo-Label [54] and MixUp [109] on multiple datasets. Therefore by comparing our approach with CAN, CDAN and MixMatch and outperforming them, the superiority of PPPL over many other baselines can be inferred.

4.5.1.3 Implementation Details

To evaluate PPPL on the CICIDS-2017 dataset we implemented our method in the TensorFlow framework and run our experiments on a system with 32 GB of RAM, a 3.6Ghz Core-i7 CPU and an Nvidia TITAN V GPU. For each of the six domain adaptation tasks we defined, we first trained a two-layer fully-connected network ($2048 \rightarrow 2$) with ReLU non-linearity after the first layer on the source domain. During this phase, we trained the model for six epochs with a batch size of 256 using Adam adam optimizer and adjusted learning rate by $\eta_p = \eta_0(1 + \alpha)^{-\beta}$ where $\eta_0 = 0.0001, \alpha = 0.001, \beta = 0.75$ and $p = i/5$ where i is the number of iterations passed from the beginning of the training. Since this dataset is imbalanced, in each epoch, we first downsampled the benign inputs to become the same size as malicious inputs and then trained the model on them. For the domain adaptation phase, we trained the model using Alg. 3. During this phase, we also used the Adam optimizer with a learning rate of 0.0001 and a batch size of 256 and also, similar to the initial phase, at the beginning of each epoch of our algorithm we balanced our dataset. In addition, during the first 30 iterations of our algorithm, instead of excluding samples of the class which is predicted more than its expected class proportion at line 8 of Alg. 3 since there are only two classes, we change their pseudo-labels to the opposite class and keep them into the training set. For the rest of the iterations, we train the model with the exclusion of samples which are predicted more than their expected class proportions.

To evaluate other baselines we used the codes published by the authors of those methods and trained the same model on each of the domain adaptation tasks we defined. We evaluated these methods on our network traffic dataset with several different hyperparameters and reported the best result they could achieve.

4.5.1.4 Scenario 1: Perfect knowledge about target domain class proportions

To see how well PPPL can detect unseen anomalies in the network traffic we first consider a scenario in which the target domain class proportions can be accurately guessed. Table 4.1 shows how PPPL performs in this situation in comparison to other baselines. As can be seen, our baselines perform very poorly on this dataset. In contrast, PPPL significantly improves the detection rate of unseen anomalies compared to only training on the source domain in all of the domain adaptation tasks and also it is 58.4% better than the best baseline we compared with based on the average F1 score.

Table 4.1: Results of all methods on the CICIDS-2017 dataset. The numbers reported in this table are F1 scores.

Method	A → B	A → C	B → A	B → C	C → A	C → B	Avg
Only-Src	0.055	0.215	0.028	0.087	0.010	0.016	0.068
CDAN	0.007	0.004	0.006	0.013	0.000	0.025	0.009
CAN	0.662	0.169	0.123	0.333	0.000	0.005	0.215
MixMatch	0.000	0.000	0.000	**0.708**	0.000	0.650	0.226
PPPL	**0.967**	**0.634**	**0.821**	0.661	**0.818**	**0.956**	**0.810**

4.5.1.5 Scenario 2: Inaccurate knowledge about target domain class proportions

Since it might not always be feasible to accurately guess the ratio of malicious traffic, in the second scenario, we evaluate our approach in a situation where some error exists in the guessed target domain class proportions. More specifically, for each of the domain adaptation tasks in this dataset, we modified the anomalous class proportion such that $|\hat{CP}_a - CP_a| = E \times CP_a$. where $E \in \{0.1, 0.2, ..., 0.7\}$, CP_a is the real ratio of the malicious traffic in the target domain and $\hat{CP}_a$ is the guessed one.

Figure 4.6 demonstrates how our method performs in such a scenario. In this figure, the left-most bar shows the average F1 score across all the six domain adaptation tasks when there is no error in the guessed class proportions and the next bars show the same metric while we considered different levels of error. The right-most bar also shows the detection rate in the target domain when we only train the model on the source domain. Note that even when there is 70% error in the guessed malicious traffic ratio the detection rate only drops by 3% compared to the case when the target domain class proportions are guessed accurately and it is still 71% better compared to the case where we only trained the model on the source domain samples. Therefore in a scenario where the target domain class proportions can be guessed but with some error, our method still performs significantly better than other baselines in detecting anomalies in network traffic.

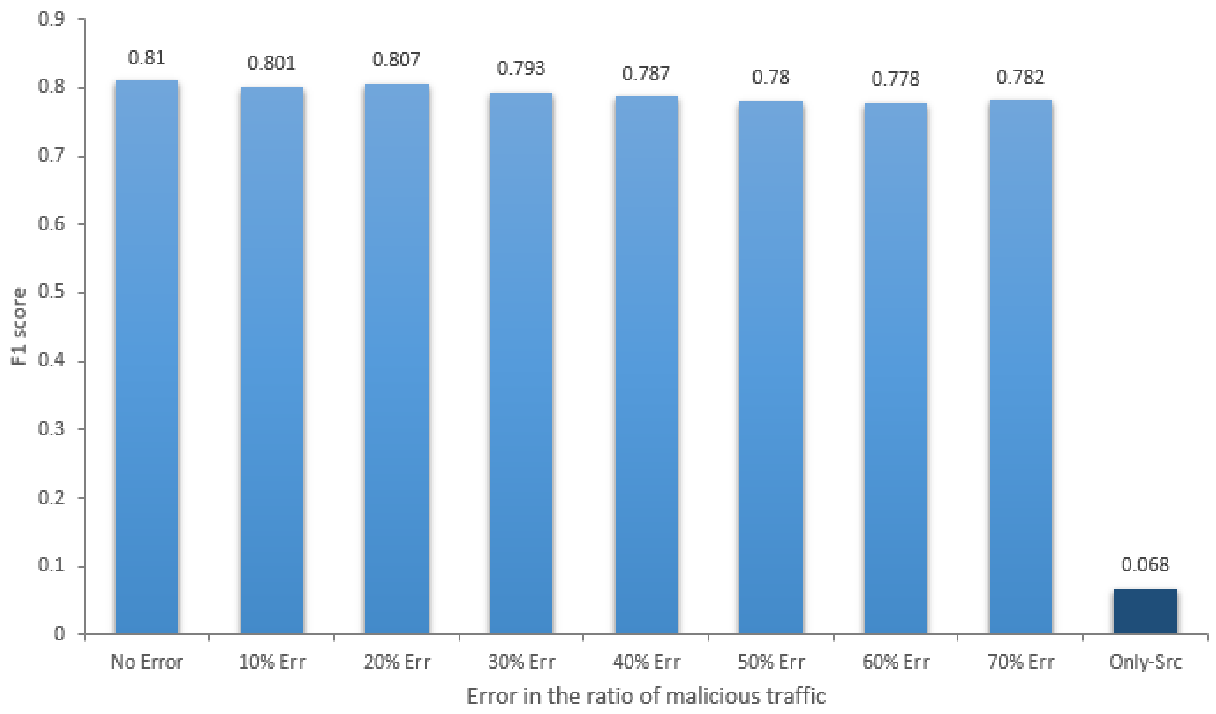

Figure 4.6: Average F1 score across all the six domain adaptation tasks defined on the CICIDS-2017 dataset when the guessed ratio of malicious traffic used in PPPL algorithm has some error.

4.5.1.6 Scenario 3: No prior knowledge about target domain class proportions

In this scenario, we want to further relax the condition related to the target domain class proportions and see how well PPPL can detect anomalies when there is no prior knowledge about the target domain class proportions. In this situation, instead of fixed class proportions, we run our algorithm by enforcing adaptive class proportions as follows:

$$CP_t = (1 - w).CP_{t-1} + w.\hat{CP}_t$$

$$CP_0 = CP_{src}$$

where CP_{src} is the source domain class proportions, $\hat{CP}$ is the class proportions calculated based on the pseudo-labels assigned to the target domain samples and w is a hyperparameter in the range of $[0, 1]$. In other words, during the first iteration of our algorithm, we enforce source domain class proportions. Note that the source domain class proportions can always be calculated as we know the source domain labels. Then during the next iterations, we update these values using an exponential moving average of the *predicted* class proportions of the target domain (which are calculated based on their assigned pseudo-labels).

In this formula, w controls how strongly we want to enforce source domain class proportions: When w is 0, we enforce the source domain class proportions during all the iterations of our algorithm which will be beneficial for the cases where the source domain class proportions are very similar to the real target domain class proportions. On the other hand, as w grows we make our estimation to be more similar to the latest predicted target domain class proportions which might be more beneficial for the cases where source domain and target domain class proportions are different. Therefore, different values of w lead to models which have different detection rate. Since we don't know which w works the best, in order to detect unseen anomalies with PPPL in this scenario we train an ensemble of models, each using a different w and average their predictions to calculate our final score as follows:

$$FinalScore = \frac{1}{K}\sum_{i=1}^{K} F_i(x)$$

where K is the total number of models in our ensemble, each of F_i is a model trained with different value of w and $F_i(x)$ is the output of a model which is a $2-$dimensional vector. In practice, for each of our domain adaptation tasks, we trained two models with $w = 0.1$ and $w = 0.7$. Figure 4.7 shows how PPPL in this scenario performs compared to the case when we have accurate knowledge about target domain class proportions. In this figure, each bar is the average F1 score across all the six domain adaptation tasks in the CICIDS-2017 dataset. Note that while the detection rate drops compared to the case where we know the target domain class proportions accurately, the average F1 score of our method in this scenario is 0.698 which is still significantly better than the other baselines. Our approach still detects unseen anomalies 63% better than just training on the source domain and 47.2% better than MixMatch as the best baseline on this dataset.

The average detection rate could be further increased if we could find a way to select the model which detects anomalies in the target domain better in the ensemble of our models. Note

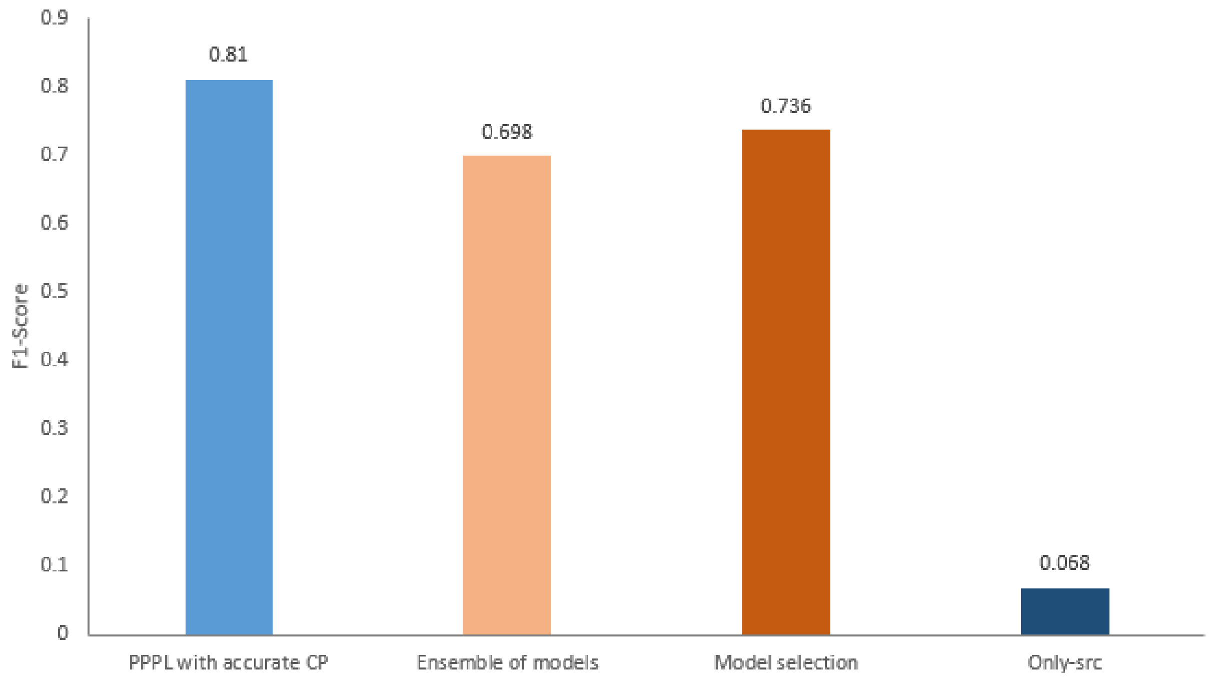

Figure 4.7: Average F1 score across all the six domain adaptation tasks defined on the CICIDS-2017 dataset when there exists no prior knowledge about target domain class proportions.

that since we don't know the labels of the target domain samples there is no straightforward way to select the best model. However, we can approximate the detection rate in the target domain with the help of source domain samples. Among the models that we train, we expect that the one that detects different perturbed versions of anomalies in the source domain with a higher rate, to generalize better and therefore have a higher chance of detecting anomalies in the target domain. In practice, we defined a new metric to measure the generalization capacity of each of the models in our ensemble as follows:

$$GeneralizationScore = \sum_{i=1}^{20} F1(X_{src} + 0.01 \times i \times R, Y_{src})$$

$$R \sim Uniform(0,1), X \in [0,1]$$

where R is a random matrix with the same size as X_{src} which is generated from a Uniform distribution and $F1(.)$ is the F1-score calculated for a given model when provided by a perturbed version of the source domain samples. In other words, the generalization score of a model is the summation of the

F1-scores calculated on the source domain when perturbed with different levels of noise and a higher value shows better generalization capacity. Therefore, in our second approach instead of getting the average of models' outputs in our ensemble, we select the one with a higher generalization error and only use that model to detect unseen anomalies in the target domain. Following this approach, as shown in Figure 4.7, we could further increase the detection rate to 0.736 based on average F1-score which is 66.8% better than just training on the source domain and 51% better than the best baseline.

4.5.2 Generalization of PPPL on other input types

As we mentioned earlier, PPPL has been designed to generalize better than other domain adaptation methods across different input types. Therefore to evaluate how well it generalizes we report its performance on four other input types on tasks such as image recognition and sentiment analysis of text inputs.

4.5.2.1 Datasets

Office-31: This dataset is widely used for the evaluation of visual domain adaptation methods and contains 4,652 images in 3 different domains known as Amazon (A), DSLR (D) and Webcam (W). The images in the Amazon domain are collected from amazon.com and have white backgrounds. The images from the DSLR domain and Webcam domains are captured from the objects in an office with DSLR and webcam cameras respectively. The images are from objects available in ordinary offices (e.g. phone, calculator, monitor, keyboard, ruler, ...) and they are categorized into 31 different classes. The Amazon domain contains 2,817 images, The DSLR domain contains 498 images and the Webcam domain contains 795 images. For this dataset, there are 6 domain adaptation tasks: A$\rightarrow$D, A$\rightarrow$W, D$\rightarrow$A, D$\rightarrow$W, W$\rightarrow$A and W$\rightarrow$D. For each task, we used all the images and their labels from the source domain and all the images without their labels for the target domain. Then for each task, the classification accuracy on the target domain samples is reported.

Office-Home: This dataset is also used for the evaluation of visual domain adaptation methods but it is harder than Office-31. This dataset contains 15,588 images in 65 different categories from objects available in ordinary homes and offices. For this dataset, there are four available domains known as Art (Ar), Clipart (Cl), Product (Pr) and Real World (Rw). The Art domain contains 2,427 images, the Clipart domain contains 4,365, the Product domain contains 4,439 and the Real World domain contains 4,357 images. For this dataset, there are 12 domain adaptation tasks (e.g. Ar$\rightarrow$Cl, Pr$\rightarrow$Ar, ...). For each task, we used all the images and their labels from the source domain and all the images without their labels from the target domain during training. Then for evaluation for each task, the classification accuracy on all of the target domain samples is reported.

CIFAR-STL: This dataset is created from images of 9 overlapping classes of CIFAR-10 and STL training sets. There are 45,000 images in the CIFAR domain (5,000 images in each class) and 4,500 images in the STL domain (500 images in each class). each image size in the CIFAR domain is 32x32. We also resized STL images to 32x32 before training for all methods. There are two domain adaptation tasks for this dataset (CIFAR $\rightarrow$ STL and STL $\rightarrow$ CIFAR). For each task, we used all the images and their labels from the source domain and all the images without their labels from the target domain during training. Then for evaluation for each task, the classification accuracy on all of the target domain samples is reported.

Multi-Domain Sentiment (MDS): This dataset is widely used to evaluate domain adaptation methods built for sentiment analysis based on text inputs. It contains 27,677 product reviews from amazon.com about four product domains: books (B), dvds (D), electronics (E) and kitchen appliances (K). The goal is to classify the reviews into positive and negative classes. For each domain, 2,000 reviews are named labeled and around 4,000 (4,465 for books, 3,586 for dvds, 5,681 for electronics and 5,945 for kitchen appliances) are named unlabeled. For this dataset, there exist 12 different tasks between each pair of domains. For each task, during training of each method, we used 2,000 labeled reviews from the source domain and both of the labeled and unlabeled reviews, from the target domain (without their labels). Then for evaluation, the classification accuracy is reported on the unlabeled reviews of the target domain. Also for evaluation of all the methods, Bag of Words (BoW) prepossessing was used. Then for each task, we selected 30,000 features that were the most

Table 4.2: Model architecture used for training on CIFAR-STL dataset.

Layer Type	Parameters
Convolution + BatchNorm + ReLU	#filters=128, kernel_size=3, stride=1, pad=1
Convolution + BatchNorm + ReLU	#filters=128, kernel_size=3, stride=1, pad=1
Convolution + BatchNorm + ReLU	#filters=128, kernel_size=3, stride=1, pad=1
Max Pooling	pool_size=2, stride=2
Dropout	dropout_ratio=0.5
Convolution + BatchNorm + ReLU	#filters=256, kernel_size=3, stride=1, pad=1
Convolution + BatchNorm + ReLU	#filters=256, kernel_size=3, stride=1, pad=1
Convolution + BatchNorm + ReLU	#filters=256, kernel_size=3, stride=1, pad=1
Max Pooling	pool_size=2, stride=2
Dropout	dropout_ratio=0.5
Convolution + BatchNorm + ReLU	#filters=512, kernel_size=3, stride=1, pad=0
Convolution + BatchNorm + ReLU	#filters=256, kernel_size=1, stride=1
Convolution + BatchNorm + ReLU	#filters=128, kernel_size=1, stride=1
Average Pooling	pool_size=6, stride=1
Fully Connected	#units=9

frequent among the reviews of that task and discarded other ones and fed them to our model.

4.5.2.2 Implementation Details

For the Office-31 and Office-Home datasets, we used ResNet-50 [37] trained on ImageNet as the feature extractor. We removed the final layer of the ResNet-50 model and added 4 new layers at the end: 3 new layers, each with size 4096 and ReLU non-linearity followed by another layer with the same size as the number of classes ($4096 \rightarrow 4096 \rightarrow 4096 \rightarrow \#classes$). We first fine-tuned the model on the source domain. We used Adam adam optimizer and adjusted learning rate by $\eta_p = \eta_0(1+\alpha)^{-\beta}$ where $\eta_0 = 0.001, \alpha = 0.001, \beta = 0.75$ and $p = i/5$ where i is the number of iterations passed from the beginning of the training. During this phase we only trained the new layers that we added to the ResNet-50 as follows: we fed all the source images which were randomly cropped to the ResNet model and extracted features from the last layer of it (the one that is connected to the first new layer that we added) for three times. Then after each time, we trained the new layers with these extracted features for 2,500 iterations with a mini-batch of 36 images. We also followed the implementation of CAN and used domain-specific batch-normalization layers

Table 4.3: Accuracy (%) for all the six tasks of Office-31 based on ResNet-50.

Method	A $\rightarrow$ D	A $\rightarrow$ W	D $\rightarrow$ A	D $\rightarrow$ W	W $\rightarrow$ A	W $\rightarrow$ D	Avg
Only-Src	82.1	79.4	66.9	98.1	66.3	99.8	82.1
CDAN	92.9	94.1	71.0	98.6	69.3	100.0	87.7
CAN	95.0	94.5	**78.0**	99.1	77.0	99.8	90.6
PPPL	**95.0**	**96.1**	77.8	**99.2**	**77.3**	**100.0**	**90.9**

Table 4.4: Accuracy (%) for all the 12 tasks of Office-Home based on ResNet-50.

Method	Ar$\rightarrow$ Cl	Ar$\rightarrow$ Pr	Ar$\rightarrow$ Rw	Cl$\rightarrow$ Ar	Cl$\rightarrow$ Pr	Cl$\rightarrow$ Rw	Pr$\rightarrow$ Ar	Pr$\rightarrow$ Cl	Pr$\rightarrow$ Rw	Rw$\rightarrow$ Ar	Rw$\rightarrow$ Cl	Rw$\rightarrow$ Pr	Avg
Only-Src	41.3	67.8	75.0	56.9	65.2	67.2	53.7	39.0	74.5	66.2	43.2	78.1	60.7
CDAN	50.7	70.6	76.0	57.6	70.0	70.0	57.4	50.9	77.3	70.9	56.7	81.6	65.8
CAN	60.0	79.0	81.3	68.2	78.9	78.3	67.7	57.1	83.1	74.3	**62.9**	84.9	73.0
PPPL	**62.9**	**80.2**	**82.2**	**70.0**	**80.8**	**80.7**	**69.5**	**58.4**	**83.6**	**77.2**	62.0	**85.8**	**74.4**

for these two datasets. Then during domain adaptation, we trained the whole model with Adam optimizer and the same learning rate adjustment function but we set $\eta_0 = 0.0001 \times 0.25$ and we updated it after each epoch. We also used a batch size of 64 where half of it was from source domain images and the other half was from target domain images. For these two datasets, we used the code from CAN for loading datasets and data preprocessing.

For the CIFAR-STL dataset, we first trained the model architecture described in Table 4.2 on the source domain from scratch. Then during domain adaptation, we trained the model with Adam optimizer with a learning rate of 0.0001 and batch size of 128.

For the MDS dataset, we trained a fully-connected model with 4 layers: 3 layers each with size 2048 and ReLU non-linearity followed by another layer with size 2 ($2048 \rightarrow 2048 \rightarrow 2048 \rightarrow 2$). First, for each task, we trained this model from scratch on the source domain for 1,000 iterations. Then we did domain adaptation. We used the same hyper-parameters as the ones we used for Office-31 and Office-Home datasets with the same initial values during both phases.

Table 4.5: Results of all methods on the CIFAR-STL dataset.

Method	CIFAR $\rightarrow$ STL	STL $\rightarrow$ CIFAR	Avg
Only-Src	76.0	61.3	68.6
CDAN	77.6	63.3	70.5
CAN	76.6	55.5	66.0
PPPL	**79.6**	**69.7**	**74.6**

4.5.2.3 Results

PPPL can work as good as other baselines on the Office-31 and Office home datasets where our baselines demonstrate their best performance. The results on the Office-31 dataset are reported in Table 4.3. On this dataset, PPPL works as good as CAN and even slightly better. The results on the Office-Home dataset are reported in Table 4.4. PPPL outperforms other baselines on this dataset. Our results are 1.4% better than CAN and 8.6% better than CDAN.

In addition, PPPL outperforms other baselines in other datasets. Table 4.5 shows the results of all methods on the CIFAR-STL dataset. Note that on average our approach notably improves model performance on this dataset compared to training only on the source domain samples (6% improvement). Whereas, on average CAN performs even worse than just training on the source domain and CDAN performs just slightly better than training a model using only source domain samples on the CIFAR-STL dataset. The same problem can be observed for the MDS dataset. The results for this dataset are reported in Table 4.6. Note that CDAN performs worse than training only on the source domain samples and CAN just slightly works better than it. While PPPL outperforms it by 6.3%. Therefore these results confirm that our approach generalizes better across different input types.

4.5.2.4 Ablation Study

Finally, to show the necessity of each component of our method, we compare PPPL with the following alternative training strategies:

- **A1:** Instead of the MSE loss function, we train the model with cross-entropy loss.

Table 4.6: Accuracy (%) for all the 12 tasks of the MDS dataset.

Method	B→ D	B→ E	B→ K	D→ B	D→ E	D→ K	E→ B	E→ D	E→ K	K→ B	K→ D	K→ E	Avg
Only-Src	81.0	73.0	75.6	78.0	74.2	78.5	71.4	72.3	85.7	73.2	74.0	86.3	76.9
CDAN	**83.3**	80.5	82.4	79.0	68.3	82.1	61.8	65.9	86.0	65.5	65.4	84.3	75.4
CAN	79.3	77.8	80.5	75.6	76.2	81.4	73.9	74.4	85.3	73.9	70.9	83.1	77.7
PPPL	83.2	**84.2**	**86.0**	**81.9**	**84.4**	**87.1**	**75.5**	**80.3**	**89.9**	**77.1**	**79.6**	**89.2**	**83.2**

Table 4.7: Alternative training strategies.

Task	A1	A2	A3	A4	PPPL
A $\rightarrow$ D	86.6	89.4	91.4	89.2	95.0
B $\rightarrow$ E	81.7	74.3	76.3	79.3	84.2

- **A2:** We include all the target samples into the training loop from the first epoch. That is to say we modify line 7 of Algorithm 3 to become $W_t \leftarrow CalculateWeight(CS_t, PL_t, 100)$.
- **A3:** Instead of class-aware weight assignment in $CalculateWeight$ function, we assign weights to target samples without grouping them based on their assigned pseudo-label and group them only based on their certainty scores.
- **A4:** We don't adjust the training set based on target class proportions. In other words, we remove line 8 from Algorithm 3.

Table 4.7 shows the results of training models with these alternative training strategies where in each of them one of the components of PPPL is eliminated in comparison to PPPL. For doing this ablation study, we trained the models on two different domain adaptation tasks namely A $\rightarrow$ D from the Office-31 dataset and B $\rightarrow$ E from the MDS dataset. Note that on average, all of the alternative training strategies perform worse than PPPL. Therefore, all the components of our method are necessary to achieve the best results.

4.6 Conclusion

In this chapter, we proposed Proportional Progressive Pseudo Labeling (PPPL), a simple and novel method for domain adaptation that generalizes better than other methods across different input types and we showed how a more accurate NIDS can be built with it while labeling a limited number of input samples. PPPL aims to progressively reduce the error on the target domain by assigning pseudo-labels to the target domain samples and training the model with them while excluding samples with more likely wrong pseudo-labels from the training set and also postponing training on such samples. Experiments on multiple different tasks confirm the superiority of our approach compared to other strong baselines with up to 58.4% improvement in detection rate based on average F1-score for detecting unseen anomalies in network traffic.

Chapter 5

Future Work and Conclusion

5.1 Future Work

In this section, we discuss the directions we would like to extend our work in the future: Making NIDS robust against the poisoning attack and reducing the number of required labeled data points by leveraging active learning.

5.1.1 Making NIDS robust against poisoning attack

In this dissertation, we discussed how to enhance the robustness of ML-based NIDS against the adversarial example attack. The adversarial example attack is carried out during the inference time when the training phase has been finished. However, this is not the only attack that threatens the ML models. There is also another type of attack known as poisoning attack which happens during the training phase [13]. During a poisoning attack, the attacker injects some poisoning samples into the training data with the goal to make the trained model predict some other inputs during the inference phase as the attacker's desired class. It has been shown that such poisoning attacks can be applied to mislead worm signature generators [72, 76], subvert spam filters [70] and corrupt many other ML-models used in other tasks [13, 71, 81, 102]. Since it might be necessary to retrain NIDS trained on network traffic after a while, therefore such models have the potential to be vulnerable to the poisoning attack. As a future direction, we want to explore how a poisoning attack can be successfully carried out against NIDS and study the extent to which NIDS are vulnerable to it. Furthermore, we want to see how we can utilize some of the defenses proposed against this attack [41] and also design our own defense to build an NIDS which is also more robust against poisoning attack.

5.1.2 Reducing the need for labeled data points by leveraging active learning

In this dissertation, we showed how an ML-based NIDS can be trained with the help of our domain adaptation method, and by labeling some known attacks to better detect unseen anomalies. In the future, we want to explore how we can further decrease the number of required labeled data points to detect unseen anomalies by leveraging active learning. Active learning methods aim to reduce the number of points that need to be labeled by bringing a human in the loop. Instead of labeling a huge dataset beforehand and training a model on it, an active learning algorithm initially needs a small labeled dataset and then iteratively queries a human to label some of the data points during the training phase. In the past few years, many different active learning algorithms have been proposed [8, 44, 45, 64, 77, 80, 105] which have shown the number of required labeled samples can be significantly reduced while achieving comparable results with traditional supervised learning approaches. Most of these methods differ from each other in the way they select the samples that they should query for. active learning has also been used for labeling network traffic datasets and training models to detect network attacks in situations where no domain gap exists [10, 21, 48, 97]. We would like to leverage active learning in our problem where there exists a domain gap between network attacks. More specifically, in the future, we want to first train a model in an unsupervised manner on the benign network traffic and then use an active learning method to make that model detect known attacks in the source domain and then make that model detect unseen anomalies in the target domain by utilizing our domain adaptation method. We hope this approach significantly reduces the number of labeled data points needed in the source domain while achieving the same detection rate in the target domain as the case where we label all of the source domain samples.

5.2 Conclusion

In this dissertation, we showed how a better NIDS can be built by utilizing neural networks while addressing the challenges that arise from training these models on network traffic. First of all, we discussed how the lack of evaluation of NIDS in an adversarial setting can put network systems in danger and demonstrated how malicious traffic can be modified adversarially without breaking the underlying network protocols and by preserving the original malicious intent. We showed that recently proposed NIDS are vulnerable to the adversarial example attack and our approach can reduce their detection rate for different network attacks by up to 70%. Second, we presented Reconstruction from Partial Observation (RePO) as a new method to make NIDS more robust against adversarial examples. We showed that our method can improve the average detection of different network attacks by up to 45% in an adversarial setting.

Finally, to fully utilize neural networks, better detect new types of anomalies in the network traffic and in order to get adapted to the new changes that might happen in the input data distribution after a while, we proposed to train a model while labeling a portion of network traffic and leveraging a domain adaptation technique. We showed that existing domain adaptation techniques don't work well for detecting anomalies in network traffic As a result, we designed our own domain adaptation method called Proportional Progressive Pseudo Labeling (PPPL). We demonstrated that PPPL generalizes better than other methods across different input types and also can significantly improve the detection rate of unseen anomalies in network traffic compared to the state-of-the-art methods with up to 58% improvement based on average F1-score.

Bibliography

[1] Nmap. `https://nmap.org/`, 2009.

[2] Patator. `https://github.com/lanjelot/patator`, 2011.

[3] Low orbit ion canon. `https://github.com/NewEraCracker/LOIC`, 2014.

[4] Botnet ares. `https://github.com/sweetsoftware/Ares`, 2015.

[5] Accenture. Ninth annual cost of cybercrime study. `https://www.accenture.com/_acnmedia/PDF-99/Accenture-Cost-Cyber-Crime-Infographic.pdf`, 2019.

[6] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov. Security in software defined networks: A survey. IEEE Communications Surveys Tutorials, 17(4):2317–2346, 2015.

[7] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi. Deep learning approach combining sparse autoencoder with svm for network intrusion detection. IEEE Access, 6:52843–52856, 2018.

[8] Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In International Conference on Learning Representations, 2020.

[9] Shumeet Baluja and Ian Fischer. Learning to attack: Adversarial transformation networks. In Proceedings of AAAI-2018, 2018.

[10] Anaël Beaugnon, Pierre Chifflier, and Francis Bach. Ilab: An interactive labelling strategy for intrusion detection. In Marc Dacier, Michael Bailey, Michalis Polychronakis, and Manos Antonakakis, editors, Research in Attacks, Intrusions, and Defenses, pages 120–140, Cham, 2017. Springer International Publishing.

[11] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

[12] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In ECML/PKDD, 2013.

[13] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In Proceedings of the 29th International Coference on International Conference on Machine Learning, ICML'12, pages 1467–1474, USA, 2012. Omnipress.

[14] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 440–447, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[15] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In 2017 IEEE Symposium on Security and Privacy, pages 39–57, 2017.

[16] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In Deep Learning and Security Workshop, 2018.

[17] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. CoRR, abs/1901.03407, 2019.

[18] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), pages 2722–2730. IEEE, 2015.

[19] Cloudflare. Slowloris. `https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/`.

[20] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single layer networks in unsupervised feature learning. In AISTATS, 2011.

[21] Rup Kumar Deka, Dhruba Kumar Bhattacharyya, and Jugal Kumar Kalita. Active learning to detect ddos attack using ranked features. Computer Communications, 145:203–222, 2019.

[22] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14, page I–647–I–655. JMLR.org, 2014.

[23] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In International Conference on Learning Representations, 2017.

[24] Gerard Draper-Gil., Arash Habibi Lashkari., Mohammad Saiful Islam Mamun., and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP,, pages 407–414. INSTICC, SciTePress, 2016.

[25] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In Computer Vision and Pattern Recognition. IEEE, 2018.

[26] Geoff French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. In International Conference on Learning Representations, 2018.

[27] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, page 1180–1189. JMLR.org, 2015.

[28] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. J. Mach. Learn. Res., 17(1):2096–2030, January 2016.

[29] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[30] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations, 2015.

[31] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In L. Saul, Y. Weiss, and L. Bottou, editors, Advances in Neural Information Processing Systems, volume 17. MIT Press, 2005.

[32] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, Computer Security – ESORICS 2017, pages 62–79, Cham, 2017. Springer International Publishing.

[33] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In USENIX Security Symposium, 2001.

[34] Mohammad J. Hashemi, Greg Cusack, and Eric Keller. Towards evaluation of nidss in adversarial setting. In Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks, Big-DAMA '19, page 14–21, 2019.

[35] Mohammad J. Hashemi and Eric Keller. Enhancing robustness against adversarial examples in network intrusion detection systems. In 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pages 37–43, 2020.

[36] Mohammad J. Hashemi and Eric Keller. General domain adaptation through proportional progressive pseudo labeling. In 2020 IEEE International Conference on Big Data (Big Data), pages 155–162, 2020.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. The IEEE International Conference on Computer Vision (ICCV), pages 1026–1034, 2015.

[39] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. CyCADA: Cycle-consistent adversarial domain adaptation. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 1989–1998, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[40] ICS-CERT. Cyber-attack against Ukrainian critical infrastructure. `www.ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01`, 2016.

[41] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA, pages 19–35, 2018.

[42] Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4893–4902, 2019.

[43] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder and Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity, 2, 2019.

[44] Kwanyoung Kim, Dongwon Park, Kwang In Kim, and Se Young Chun. Task-aware variational adversarial active learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 8166–8175, June 2021.

[45] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

[46] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[47] Ram Shankar Siva Kumar, Andrew Wicker, and Matt Swann. Practical machine learning for cloud intrusion detection: Challenges and the way forward. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec '17, pages 81–90, New York, NY, USA, 2017. ACM.

[48] V Valli Kumari and P Ravi Kiran Varma. A semi-supervised intrusion detection system using active learning svm and fuzzy c-means clustering. In 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), pages 481–485, 2017.

[49] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In International Conference on Learning Representations, 2017.

[50] Vinod Kumar Kurmi, Shanu Kumar, and Vinay P. Namboodiri. Attending to discriminative certainty for domain adaptation. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[51] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017.

[52] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017.

[53] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. ICML 2013 Workshop : Challenges in Representation Learning (WREPL), 07 2013.

[54] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. ICML 2013 Workshop : Challenges in Representation Learning (WREPL), 07 2013.

[55] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran. Athena: A framework for scalable anomaly detection in software-defined networks. In 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 249–260, 2017.

[56] M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.

[57] Hong Liu, Mingsheng Long, Jianmin Wang, and Michael Jordan. Transferable adversarial training: A general approach to adapting deep classifiers. In Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 4013–4022, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[58] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, page 97–105. JMLR.org, 2015.

[59] Mingsheng Long, ZHANGJIE CAO, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In Advances in Neural Information Processing Systems 31, pages 1640–1650. Curran Associates, Inc., 2018.

[60] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. Unsupervised domain adaptation with residual transfer networks. NIPS'16, page 136–144, Red Hook, NY, USA, 2016. Curran Associates Inc.

[61] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. Deep transfer learning with joint adaptation networks. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, page 2208–2217. JMLR.org, 2017.

[62] R. K. Malaiya, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim. An empirical evaluation of deep learning for network anomaly detection. In 2018 International Conference on Computing, Networking and Communications (ICNC), pages 893–898, March 2018.

[63] Markets and Markets. Intrusion detection and prevention systems market. `https://www.marketsandmarkets.com/Market-Reports/intrusion-detection-prevention-system-market-199381457.html`, 2020.

[64] Christoph Mayer and Radu Timofte. Adversarial sampling for active learning. In 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 3060–3068, 2020.

[65] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In Network and Distributed System Security Symposium 2018 (NDSS'18), 2018.

[66] Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(8):1979–1993, 2019.

[67] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2574–2582, 2016.

[68] Steve Morgan. Global cybercrime damages predicted to reach $6 trillion annually by 2021. `https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021`, 2020.

[69] Toshihiro Nakae. `https://github.com/tnakae/DAGMM`, 2018.

[70] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, LEET'08, USA, 2008. USENIX Association.

[71] Andrew Newell, Rahul Potharaju, Luojie Xiang, and Cristina Nita-Rotaru. On the practicality of integrity attacks on document-level sentiment analysis. In Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop, AISec '14, page 83–93, New York, NY, USA, 2014. Association for Computing Machinery.

[72] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In Diego Zamboni and Christopher Kruegel, editors, Recent Advances in Intrusion Detection, pages 81–105, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[73] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17, pages 506–519, New York, NY, USA, 2017. ACM.

[74] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In Security and Privacy (EuroS&P), 2016 IEEE European Symposium on, pages 372–387. IEEE, 2016.

[75] Zhongyi Pei, Zhangjie Cao, Mingsheng Long, and Jianmin Wang. Multi-adversarial domain adaptation. In AAAI Conference on Artificial Intelligence, 2018.

[76] R. Perdisci, D. Dagon, Wenke Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In 2006 IEEE Symposium on Security and Privacy (S P'06), pages 15 pp.–31, 2006.

[77] Robert Pinsler, Jonathan Gordon, Eric Nalisnick, and José Miguel Hernández-Lobato. Bayesian batch active learning as sparse subset approximation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

[78] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. Dataset Shift in Machine Learning. The MIT Press, 2009.

[79] Rapid7. Metasploit. `https://www.metasploit.com/`, 2011.

[80] Pengzhen Ren, Y. Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and X. Wang. A survey of deep active learning. ArXiv, abs/2009.00236, 2020.

[81] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Antidote: Understanding and defending against poisoning of anomaly detectors. In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, IMC '09, page 1–14, New York, NY, USA, 2009. Association for Computing Machinery.

[82] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, Computer Vision – ECCV 2010, pages 213–226, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[83] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, page 1171–1179, Red Hook, NY, USA, 2016. Curran Associates Inc.

[84] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. NIPS'16, page 1171–1179, Red Hook, NY, USA, 2016. Curran Associates Inc.

[85] Swami Sankaranarayanan, Yogesh Balaji, Carlos D. Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[86] Jan Seidl. Goldeneye. `https://github.com/jseidl/GoldenEye`.

[87] Aliaksei Severyn and Alessandro Moschitti. Twitter sentiment analysis with deep convolutional neural networks. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15, page 959–962, New York, NY, USA, 2015. Association for Computing Machinery.

[88] Iman Sharafaldin, Arash Habibi Lashkari, , and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In 4th International Conference on Information Systems Security and Privacy (ICISSP), 2018.

[89] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pages 1528–1540, New York, NY, USA, 2016. ACM.

[90] Sergey Shekyan. Slowhttptest. `https://github.com/shekyan/slowhttptest`.

[91] Barry Shteiman. Hulk. `https://www.kitploit.com/2014/04/hulk-web-server-dos-tool.html`.

[92] Symantec. Internet security threat report. https://www.symantec.com/security-center/threat-report, 2019.

[93] Synopsis. Heatbleed. http://heartbleed.com/.

[94] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17, page 4278–4284. AAAI Press, 2017.

[95] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[96] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, page 1195–1204, Red Hook, NY, USA, 2017. Curran Associates Inc.

[97] Jorge L. Guerra Torres, Carlos A. Catania, and Eduardo Veas. Active learning approach to label network traffic datasets. Journal of Information Security and Applications, 49:102388, 2019.

[98] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[99] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[100] Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. Interpolation consistency training for semi-supervised learning. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pages 3635–3641. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[101] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th International Conference on Machine Learning, ICML '08, page 1096–1103, 2008.

[102] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In Francis Bach and David Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pages 1689–1698, Lille, France, 07–09 Jul 2015. PMLR.

[103] Xiang Xu, Xiong Zhou, Ragav Venkatesan, Gurumurthy Swaminathan, and Orchid Majumder. d-sne: Domain adaptation using stochastic neighborhood embedding. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[104] C. Yin, Y. Zhu, S. Liu, J. Fei, and H. Zhang. An enhancing framework for botnet detection using generative adversarial networks. In 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD), pages 228–234, May 2018.

[105] Donggeun Yoo and In So Kweon. Learning loss for active learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[106] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[107] Yang Yu, Jun Long, and Zhiping Cai. Network intrusion detection through stacking dilated convolutional autoencoders. Security and Communication Networks, 2017, 2017.

[108] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. CoRR, abs/1802.06222, 2018.

[109] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In International Conference on Learning Representations, 2018.

[110] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In International Conference on Learning Representations, 2018.