Defending against Adversarial Machine Learning Attacks using Hierarchical Learning: A case study on Network Traffic Attack Classification

Andrew McCarthy[a], Essam Ghadafi[a], Panagiotis Andriotis[a], Phil Legg[a]

[a]*Computer Science Research Centre, University of the West of England, Bristol, UK*

Abstract

Machine learning is key for automated detection of malicious network activity to ensure that computer networks and organizations are protected against cyber security attacks. Recently, there has been growing interest in the domain of adversarial machine learning, which explores how a machine learning model can be compromised by an adversary, resulting in misclassified output. Whilst to date, most focus has been given to visual domains, the challenge is present in all applications of machine learning where a malicious attacker would want to cause unintended functionality, including cyber security and network traffic analysis. We first present a study on conducting adversarial attacks against a well-trained network traffic classification model. We show how well-crafted adversarial examples can be constructed so that known attack types are misclassified by the model as benign activity. To combat this, we present a novel defensive strategy based on hierarchical learning to help reduce the attack surface that an adversarial example can exploit within the constraints of the parameter space of the intended attack. Our results show that our defensive learning model can withstand crafted adversarial attacks and can achieve classification accuracy in line with our original model when not under attack.

Keywords: adversarial learning, hierarchical classification, network traffic analysis, functionality preservation, machine learning, model robustness

1. Introduction

Cyber security and the protection of associated computer and network systems is fundamental for most organizations. The recent Cyber Security

Breaches survey 2022 conducted by the UK Government found that 39% businesses had experienced a cyber attack in the last 12 months, with the average cost of a cyber attack currently estimated as £2.2 million [1]. The sheer scale and magnitude of modern cyber attacks requires automated response and intervention. Machine learning (ML) is becoming widely used for the detection and classification of malicious network activity to aid the response to cyber attacks, where a mathematical model is learned to relate input feature observations to a set of possible output classes. For the classification of network traffic attacks, input features may be derived from the observed network communications and packet header information, which may be indicative of either benign traffic, or a malicious attack such as a Denial of Service, a Remote Access Trojan, a BotNet, or other network-based attack.

Whilst machine learning can help manage this wealth of information, it is not without limitation. Recent years have seen a growing interest in the domain of adversarial machine learning [2] that seeks to identify well-crafted examples that knowingly force misclassification by the model. This has been particularly effective in the computer vision domain since the manipulation of few input features (i.e., image pixels) may inadvertently adjust the performance of the model without being noticeable to the human observer, due to small perturbations of pixel intensity values. The challenge in adversarial learning is to determine which features are most susceptible such that a minimal change can result in misclassification by the model, whilst the overall input to the model appears unchanged or unaltered to the human observer. Drawing a parallel to the challenge of network traffic classification, a malicious attack should exhibit the same characteristics such that the activity is still deemed as malicious, whilst identifying the minimal amount of perturbation in the derived features such that the model believes the observation to be benign, hence resulting in misclassification. We refer to this characteristic as *functionality preservation*.

In this paper, we first examine the impact of adversarial attack generation against a well-trained network traffic classification model, and show the performance degradation. To combat this, **we present a novel defensive strategy based on hierarchical learning to help reduce the attack surface that an adversarial example can exploit** within the constraints of the parameter space of the intended attack. Our results show that our defensive learning model can withstand crafted adversarial attacks and can achieve classification accuracy in line with our original model when not under attack.

The primary contributions of our work are:

- We provide the **first comprehensive study of applying functionality-preserving adversarial learning attacks** against a multi-class network traffic classification model, and demonstrate successful attack misclassification within a constrained attack parameter space. 90.25% of the attacks were able to evade detection of a well-trained classifier, while also constraining the parameter space to preserve functionality.
- We propose a **novel defensive strategy based on hierarchical learning** to reduce the attack surface that an adversarial example can exploit within the constraints of the parameter space of the intended attack, achieving classification accuracy in line with our original model when not under attack.

The remainder of the paper is structured as follows: Section 2 presents the related works on adversarial machine learning exploring both the creation of attacks, and the defense against attacks; Section 3 presents our study on the creation of adversarial attacks against a well-trained network traffic classification model; Section 4 presents our novel defensive strategy based on hierarchical learning; Section 5 provides discussion of the research on model robustness, attack transferability, and limitations of hierarchical classification; and Section 6 concludes the work and considers future directions.

2. Related Work

Our related works section addresses relevant topics of adversarial machine learning, functionality preservation with feature perturbations, localized classification, and machine learning-based intrusion detection. We also draw upon our previous works on surveying functionality-preserving attacks [3], and feature vulnerability and robustness in network traffic analysis [4].

2.1. Adversarial Machine Learning

Unfortunately, machine learning systems are susceptible to carefully crafted attacks that aim to yield an arbitrary, or specific, misclassification. Szegedy et al. [5] first found that imperceptible perturbations of input values can result in significant differences in the output of a ML classifier. Adversarial machine learning has been a research topic for over a decade and is now an accepted topic with open problems.

Papernot et al. [6] note the security and privacy of ML is an active but nascent area of research. In this early work, they systematize their findings on security and privacy in machine learning. Indicating that a science for understanding the vulnerabilities of ML and countermeasures is slowly emerging. They utilize the classical confidentiality, integrity and availability (CIA) model to analyze: training in adversarial settings; inferring adversarial settings; robust, fair, accountable, and private ML models. Their analysis points toward two related notions of sensitivity. The sensitivity of learning models to their training data is essential for privacy-preserving ML, and similarly, the sensitivity to inference data is essential for secure ML. The crux of both notions of sensitivity is the generalization error (i.e. the gap between performance on training and test data). They focus on attacks and defenses for machine learning systems aiming to further understanding of the sensitivity of modern ML algorithms to input data and foster a science of privacy and security in machine learning.

A primary focus of this work is how adversarial examples might translate to the cyber security domain. A further complication in this domain is that of functionality preservation. In cyber security domains, it is critical that an intrusion detection classifier correctly identifies malicious traffic while minimizing false positive and false negative results since identifying truly malicious activity in a profusion of false positives is problematic. ML performance can be evaluated by accuracy, precision, recall, and other metrics such as F1-Score.

Zhang and Li [7] discuss opportunities and challenges arising from adversarial examples. They survey state-of-the-art adversarial example generation methods and defenses before raising future research opportunities and challenges. They note three challenges for adversarial example construction:

1. The difficulty of building a generalizable method.

2. The difficulty in controlling the size of perturbation (too small will not result in adversarial examples, and too large can easily be perceived).

3. Difficulty in maintaining adversarial stability in real-world applications (some adversarial examples do not hold for transformations such as blurring).

They identify black-box attacks as a challenge for defenses. Because black-box attacks require zero-knowledge of the model architecture and therefore might not be easily resisted by modifying the model architecture or parameters. Second, defenses are often specific to a single attack method and are often less suitable as a general defense. They subsequently identify an opportunity for highly transferable adversarial examples (high confidence).

2.2. Threat Model

A Realistic threat model can help harden intrusion detection systems by prioritising the smaller subset of attacks that are realistic[8]. We present the following threat model considering the strengths and weaknesses of an adversary's possible strategies.

Adversarial attacks can be designated as either poisoning attacks or inference-time attacks. Poisoning attacks effect the training phase and aim to influence classification by augmenting the training dataset with new samples or modifying existing samples. Inference attacks aim to influence classification by leveraging the sensitivity of the model to its training data. Typical strategies include gradient descent [9] and Generative Adversarial Networks (GANS) [10]. Such attacks are commonly known as evasion attacks. The goal of an attacker may be to misclassify malicious samples as benign; however, it is plausible that an attacker could gain advantage by causing misclassification between malicious classes [4].

There are three main types of adversarial attack: white-box, black-box, and gray-box. White-box attacks assume complete knowledge of the target model. Black-box attacks assume zero-knowledge of the system; although an adversary might optionally acquire knowledge through exploiting 'oracle' attack strategies using multiple queries incrementally modifying a malicious sample until it is misclassified. An alternative strategy requires the adversary to create a surrogate model. The goal here is to employ the *transferability* property of adversarial examples. Adversarial examples generated on the surrogate may *transfer* and subsequently successfully fool the target model. An attacker with 'oracle' access might be able to reverse-engineer the target model to generate a surrogate model; however, Apruzzse[8] rightly indicates that using a NIDS as an oracle is complex and demanding, feasible only in limited scenarios. Adversaries choosing this route face two obstacles: Querying the target model while stealthily avoiding detection forcing a low-and-slow approach; Acquiring feedback is difficult in IDS because the output of the model may not be directly observable to the adversary.

Fortunately, 'oracle' access is unnecessary for the creation on a surrogate model. Papernot et al.[11] states that adversarial examples affecting one model often affect another model, even where the models have different architectures or were trained on different training sets. For transferable adversarial examples it is sufficient that both models were trained to perform the *same task*. An adversary may train their own surrogate model, craft adversarial examples against the surrogate, and transfer them to the target with *very little* knowledge of the target model. Recently, Yang et al. [12] examined the adversarial transferability of surrogate models. Specifically they claim the adversarial transferability of a surrogate model can be improved when *any model* for the *same task* is used to extract and leverage soft (probabilistic) labels to train a surrogate model *without* querying the target model.

Gray-box attacks assume partial knowledge of the target model. For example, an adversary might have some knowledge of the features used by the model. Indeed this is likely. By necessity all ML IDS analyze either raw network traffic or derived metadata [8]. ML IDS systems are likely to be trained on network flows[13]. A skillful adversary with sufficient domain knowledge could estimate which features are likely to be used and how they could impact performance.

2.3. Functionality Preserving Adversarial Examples

Apruzzese et al. [8] examine adversarial examples to consider realistic attacks, highlighting that most literature considers adversaries with complete knowledge about the classifier who are free to interact with the target systems. They emphasize that few works consider 'realizable' perturbations that take account of domain and/or real-world constraints. Commonly black-box attacks assume adversaries are able to repeatedly query an 'oracle' model. The attacker may iteratively and incrementally perturb samples. At each iteration, a sample is presented, and the output is examined, pursuant to determining model decision boundaries and ultimately achieving misclassification. Such 'oracle' attacks are challenging. Each query increases the risk that the strategy will be detected and the attack foiled. Moreover, direct output from an intrusion detection system (IDS) is not easy to achieve. Apruzzese et al. [8] consider situations where such attacks gain feedback while remaining undetected. However, they acknowledge that such attacks require persistence, skill, and resources. Transferability attacks avoid these problems because the oracle access to the target model is unnecessary as the target is not queried. An adversary can build a surrogate model for the same task on which to generate adversarial examples, subsequently transferring them to the target model.

Sheatsley et al.[14] advise that most adversarial examples in cyberdetection domains violate one or more domain constraints. Moreover, they find that crafting adversarial examples in constrained domains requires a different process to unconstrained domains. They argue that constrained domains are inherently more robust against adversarial examples. Further, they posit that the exploitable threat surface of models in constrained domains is likely narrower than previously understood; however, we stress that it is important not to take succour from this statement. The attack surface may be narrower; however, carefully crafted attacks can nevertheless successfully exploit it.

Apruzzse allege that a misconception exists in the literature in that much research pursues 'minimal' perturbations. They claim that in reality adversaries are not bound by this constraint[15]. We acknowledge that adversaries will use *any* suitable method to fool the classifier, regardless the size of perturbation; However, we do not consider this final and decisive. Adversaries exist in a arms-race environment. Defenses are continually improving, and adversaries must adapt their strategies. The stark reality is that large perturbations are more easily detected by statistical measures[16]. Smaller perturbations are less easily detected and therefore confer advantages to adversaries who wish to hold persistence and remain undetected for a period of time. Furthermore, large perturbations do not necessarily confer an additional advantage over smaller perturbations. Domain constraints might be broken by large and small perturbations alike. We predict that the future trend of adversarial examples in NIDS will be to constrain the adversarial example in scope, size, or both. Specifically, we predict perturbations to small combinations of increasingly fewer features to a lesser degree. ART[17] allows feature masks to exclude certain (constrained) features from perturbation. In this work we carefully limit the scope of our perturbations to one feature (gamma=0.05, and the size of perturbation to 0.02 (theta=0.02)

2.4. Intrusion Detection

Zhang et al [13] note common classification methods for internet traffic are based on statistical properties captured as netflows. This method addresses problems of dynamic port numbers and protects user privacy. Systems can be deployed to search for patterns in the netflows. Most such systems employ machine learning to perform automated classification of traffic types, detecting and/or dropping malicious traffic.

Wu et al. [18] consider several types of deep learning systems for network attack detection, including supervised and unsupervised models to compare the efficiency and effectiveness of different attack detection methods using two intrusion detection datasets: "KDD Cup 99" dataset and an improved version known as NSL-KDD [19] [20]. These datasets are commonly used; however, they do not fairly represent modern network traffic analysis problems due to concept-drift. Networks have increasing numbers of connected devices, increasing communications per second, and new applications using the network. Moreover, the use of computer networks and the Internet has changed substantially in twenty years. The continued introduction of IPv6, Network address Translation, Wi-Fi, mobile 5G networks, and cloud providers has changed network infrastructure [21]. Furthermore, the internet is increasingly used for financial services. Akamai [22] report financial services see millions or tens of millions of attacks each day. These attacks were less common twenty years ago. Furthermore, social media constitutes much of today's internet traffic and most social media platforms were founded after the KDD Cup 99 and NSL-KDD datasets were introduced. For example, Facebook, YouTube, and Twitter were founded in 2004, 2005, and 2006 respectively but are now in the top five most visited sites [23].

Kok et al. [24] warn the dangerous trend of using outdated datasets could result in no or insufficient progress on IDS. This would ultimately lead to an untenable situation, with obsolete intrusion detection systems (IDS) while intrusion attacks continuously evolve along with user behavior and the introduction of new technologies.

Martins et al. [25] note that IDS are typically signature-based and that machine learning approaches are being widely employed for intrusion detection. They describe common white-box methods to generate adversarial examples including: Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS), Fast Gradient Sign Method (FGSM), Jacobian-based Saliency Map Attack (JSMA), Deepfool, and Carlini & Wagner attacks (C&W). They also consider black-box methods using Generative Adversarial Networks (GANS). Traditional GANS sometimes suffer problems of mode collapse. Wasserstein Generative Adversarial Networks (WGANS) solve some of these problems. They introduce the Zeroth-order optimization attack (ZOO) as a black-box method. ZOO *estimates* the gradient and optimizes an attack by iteratively adding perturbations to features. They note that most attacks have been initially tested in the image domain, but can be applied to other types of data, which poses a security threat. Furthermore, they consider there is a trade-off when choosing an adversarial attack. For example, JSMA is more computationally intensive than FGSM, but modifies fewer features. They consider JSMA to be the most realistic attack because it perturbs fewer features However, variations of FGSM and other methods can be configured to modify certain features[26], making other less computationally intensive attacks potentially realizable.

2.5. Model Training For Robust Models

The trustworthiness and quality of a model is impacted by the distribution, quality, quantity, and complexity of dataset training samples [27]. Biased models are more susceptible to adversarial examples. Therefore models should be trained on unbiased training data; although Johnson et al. consider the *absolute* number of training samples may be more important than the ratio of class imbalance [28]. Cybersecurity datasets are often prone to bias, partly because of limited samples of some malicious traffic (E.g. *zero-day* attacks) and large amounts of benign traffic. Sheatsley et al. [29] state biased distributions enable successful adversarial examples with very few feature modifications. Common data-level techniques tackle biased datasets by resampling: oversampling, undersampling, and hybrid sampling by combining modest oversampling of minority classes and modest undersampling of majority classes, aiming to give better model performance than applying either technique alone. Algorithm-level techniques tackling dataset bias commonly employ cost-sensitive learning where a class penalty or weight is considered or decision thresholds are shifted to reduce bias [28].

2.6. Robustness

Robustness can be defined as the performance of well-trained models facing adversarial examples [30]. Essentially, robustness considers how sensitive a model's output is to a change in the input. The robustness of a model is related to the generalization-error of the model. A recognized trade-off exists between accuracy and robustness in machine learning. That is, highly accurate models are less robust to adversarial examples. Machine learning models in adversarial domains must be both highly accurate and robust. Therefore, improving the robustness of machine learning models enables safer deployment of ML systems across a wider range of domains, including IDS. To critically evaluate and make fair comparisons of the robustness of a model under attack, robustness metrics are necessary. Common machine learning metrics can be used to provide consistency, such as Precision, Recall, and F1-Score.

2.7. Common Defences

Some research considers how model weights can be used to disrupt generation of adversarial examples using white-box methods [31] [32]. However, these defenses can be bypassed by using black-box methods. Much research considers the detection of adversarial examples by considering whether a sample is out of distribution [16] [33]. Detection is hard because adversarial examples do not exhibit intrinsic properties. Moreover, many detection methods are susceptible to good attacker-loss functions [34]. Adversarial training is a simple method aiming to improve robustness; however, it is not scalable. Moreover, Tramér *et al.* [35] found adversarial training can be bypassed. Some research investigates hardening the architecture of the model. Perhaps, changing model parameters or employing ensembles [2]. Defensive dropout uses a dropout layer and can block black-box and transferability-based attacks [36]. Adversarial defenses exist in an arms race where adversaries adapt to defenses by adopting new strategies. Therefore, defenses must remain secure against adversaries who understand our model defenses.

2.8. Ensemble classification

Biggio [2] asserts that ensemble classifiers have been exploited to improve robustness; however, they must be properly constructed to avoid worsening robustness. A typical ensemble classifier often used in intrusion detection is a Random Forest (RF) [37]. Other state-of-the-art ensemble classifiers include XGBoost, Histogram-based Gradient Boosting Classifier (HBBC), and Light Gradient Based Machine (LGBM). These ensemble classifiers help with robustness because their construction generates multiple randomised estimators. This complicates the task of generating attacks capable of fooling all (or most) of the estimators. Some research focuses on how ensembles of estimators with opposing or different gradients can help robustness [38]. Hierarchical classification, also known as multilevel classification, is a form of ensemble classifier, where multiple classifiers are usually arranged in a top-down hierarchy.

3. Adversarial Attack of a Network Traffic Classification Model

To examine the nature of adversarial attacks, we first develop a well-trained machine learning model for network traffic classification. We use the

Python programming language for our development, along with the popular machine learning libraries, Keras[39] and scikit-learn [40]. We also use the Adversarial Robustness Toolbox (ART) [17] to support the construction of the adversarial examples. All experiments were performed using a variant of the CICIDS 2017 dataset as detailed below.

3.1. Preparing the CICIDS 2017 Dataset

The Canadian Institute for Cybersecurity IDS 2017 dataset (CICIDS2017) [41] is a recent addition to modern IDS datasets that has become increasingly popular amongst researchers. The dataset consists of a packet capture trace across a multi-system infrastructure for a period of 5 days, denoting both benign traffic activity as well as 14 common attacks including Brute Force FTP, Brute Force SSH, Denial of Service (DoS), Heartbleed, Web Attack, Infiltration, Botnet, and Distributed-Denial of Service (DDoS). The data are available as a series of packet capture (PCAP) files, as well as a "ML-ready" set of features in CSV format, derived using their CICFlowMeter tool [42][43] that derives features of the communication flow between two parties, similar to Cisco Netflow [44]. Recently, Engelen *et al.* [45] reported on errors that occur from the use of the CICFlowMeter tool that are also present in the "ML-ready" dataset. The authors resolve the issues with the CICFlowMeter tool, and provide both a new version of the software tool as well as a corrected copy of the dataset, that is, a more accurate derivation of the intended features in the original PCAP. For the purpose of our study, we now use the dataset made available by Engelen *et al.* [45].

Typical features for each flow include: Flow Duration, Packet Statistics, Flow Bytes/s, Flow Packets/s, IAT Statistics, Flags, Header Length, Down/Up Ratio, Bulk Statistics, Subflow Statistics, Init Win bytes, Active data packets forward, Active Statistics, and Idle Statistics. A set of 25 possible classes are derived in the improved labelling of the CICIDS2017 dataset, since some attacks that had previously been labelled had not successfully executed (e.g., did not result in data transmission). These attack labels are appended with the label "Attempted".

Class imbalance can skew the assessment of model performance. Inevitably for this domain, the benign class exhibits many more samples compared to the attack classes. To overcome this, we combine *oversampling* and *undersampling* techniques in sequence to effectively balance the dataset [46]. We also cleanse the dataset to remove all instances that consist of null entries. Our balanced dataset results in 7,500 samples (300 samples per class), which we consider sufficient for the purpose of our case study.

File	Traffic Type	Class Samples	Benign Samples	File Ratio	Dataset Ratio
Monday	BENIGN	529918	529918	1.00000	1.0000000
Tuesday	FTP-Patator	7938	432074	0.01837	0.0034922
	SSH-Patator	5897	-	0.01365	0.0025943
Wednesday	DoS GoldenEye	10293	440031	0.02339	0.0045282
	DoS Hulk	231073	-	0.52513	0.1016556
	DoS Slowhttptest	5499	-	0.01250	0.0024192
	DoS slowloris	5796	-	0.01317	0.0025498
	Heartbleed	11	-	0.00002	0.0000048
Thursday-Morning-WebAttacks	Web Attack - Brute Force	1507	168186	0.00896	0.0006630
	Web Attack - Sql Injection	21	-	0.00012	0.0000092
	Web Attack - XSS	652	-	0.00388	0.0002868
Thursday-Afternoon-Infilteration	Infiltration	36	288566	0.00012	0.0000158
Friday-Morning	Bot	1966	189067	0.01040	0.0008649
Friday-Afternoon-DDos	DDoS	128027	97718	1.31017	0.0563227
Friday-Afternoon-PortScan	PortScan	158930	127537	1.24615	0.0699178

Table 1: Details of the CICIDS2017 dataset. For each data file (ordered by date), we show the attack types covered, the number of class samples for each attack, and the number of benign samples included in each data file. The dataset ratio column shows classes considered over-represented at a per file level are still under-represented in the dataset as a whole when the sum off all benign samples are used to calculate the dataset ratio.

3.2. Initial Classification Model

We train a multi-class classifier using a scikit-learn MLP model, that we will refer to as our *target* model. The model consists of three dense layers and is trained for a maximum of 300 iterations. We pre-process, scale, and split the resampled dataset into train and test samples (0.7/0.3 split). We use the Adam optimizer to train the target model, to produce a well-trained model. Table 2 shows the performance of the classifier. We use the standard metrics of accuracy, precision, recall, and F1-scores. The *support* column indicates the number of occurences of the class in the specified sample. The target model achieves an accuracy of 91%, with precision of 91% , recall of 93%, and f1-scores of 89%.

To overcome the issue of 'oracle' attacks and transferability of adversarial examples [11], in addition to our target model we also train a surrogate neural network model for the same intrusion detection task. Whilst superficially similar to our target model, we utilize an alternative framework (Keras) to construct this. Since the two models are known to be *similar* and are trained with portions of a *freely available dataset* the approach is intended to be representative of a black-box attack however may well be considered to be a grey-box attack, since we inevitably have some knowledge of our underlying models. As with the target model, the surrogate model consists of three

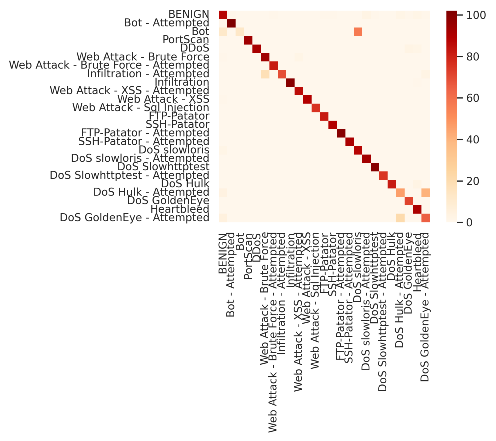
	Target Model				Surrogate Model			
	Precision	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
BENIGN	0.75	0.96	0.84	79	0.88	0.98	0.93	92
Bot - Attempted	1.00	0.98	0.99	105	1.00	0.98	0.99	105
Bot	1.00	0.48	0.65	170	1.00	0.49	0.66	167
PortScan	0.98	1.00	0.99	92	1.00	1.00	1.00	94
DDoS	0.97	1.00	0.98	94	0.99	1.00	0.99	96
Web Attack - Brute Force	0.99	0.83	0.91	115	0.97	0.78	0.86	121
Web Attack - Brute Force - Attempted	1.00	0.96	0.98	85	0.99	0.99	0.99	82
Infiltration - Attempted	0.78	0.96	0.86	73	0.68	1.00	0.81	61
Infiltration	1.00	0.96	0.98	105	1.00	1.00	1.00	101
Web Attack - XSS - Attempted	1.00	0.99	0.99	87	0.99	0.96	0.97	89
Web Attack - XSS	0.97	0.99	0.98	88	1.00	1.00	1.00	90
Web Attack - Sql Injection	1.00	1.00	1.00	76	0.99	0.99	0.99	76
FTP-Patator	1.00	1.00	1.00	82	1.00	1.00	1.00	82
SSH-Patator	1.00	0.98	0.99	91	1.00	0.99	0.99	90
FTP-Patator - Attempted	1.00	1.00	1.00	100	1.00	1.00	1.00	100
SSH-Patator - Attempted	1.00	0.99	0.99	92	1.00	1.00	1.00	91
DoS slowloris	0.06	1.00	0.11	5	0.08	1.00	0.14	7
DoS slowloris - Attempted	1.00	1.00	1.00	93	0.98	0.99	0.98	92
DoS Slowhttptest	0.99	1.00	0.99	99	0.99	0.99	0.99	100
DoS Slowhttptest - Attempted	1.00	1.00	1.00	75	1.00	1.00	1.00	75
DoS Hulk	0.99	0.99	0.99	85	0.98	1.00	0.99	83
DoS Hulk - Attempted	0.55	0.61	0.58	82	0.55	0.59	0.57	85
DoS GoldenEye	1.00	0.95	0.97	76	1.00	0.97	0.99	74
Heartbleed	0.99	0.94	0.96	96	1.00	0.96	0.98	95
DoS GoldenEye - Attempted	0.68	0.59	0.63	105	0.63	0.56	0.59	102
Macro Average	0.91	0.93	0.89	2250	0.91	0.93	0.90	2250
Accuracy	0.91				0.91			

Table 2: Target Model and Surrogate Model Classification Reports.

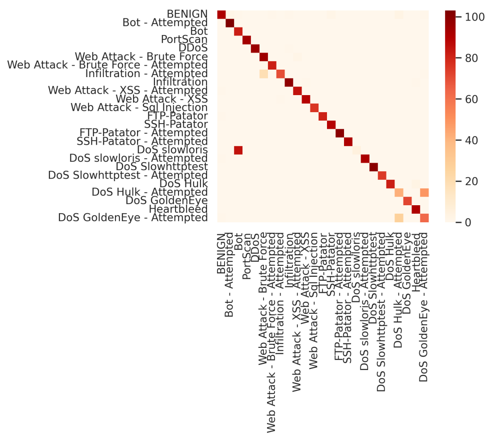
dense layers and a softmax activation layer. We use *sparse categorical cross-entropy* as our loss function, together with the Adam optimizer, to train our model for 200 epochs with an early stopping patience of 50. Table 2 shows that the model achieves an accuracy comparable to that of our target model (accuracy of 91%, with precision of 91% , recall of 93%, and F1-Scores of 90%). Figure 1 shows the confusion matrix of both the (a) target model and the (b) surrogate model to evaluate the performance of each individual class and to assess misclassification.

3.3. Using the Surrogate Model to Attack the Target Model

From the adversarial perspective, we intend to cause the target model to misclassify, often referred to as an *evasion attack*. Adversarial attacks can be performed on binary and multi-class classification systems, and typically fall into 3 groups: white-box; black-box; and gray-box attacks. White-box attacks provide full knowledge of the model, and access to the model. Many white-box attacks rely on the use of gradient descent search. Grey-box models assume some partial knowledge of the model. Black-box attacks offer zero knowledge of the model, and 'oracle' attacks are impractical for intrusion detection domain [8]. Therefore, some black-box attacks rely on the



(a)



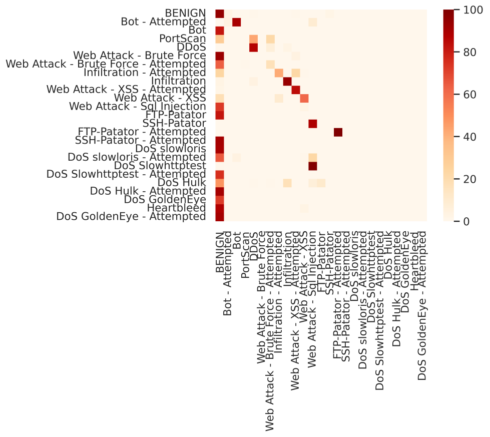
(b)

Figure 1: Confusion matrix for (a) Target model (Scikit-learn), (b) Surrogate model (Keras).

transferability properties of adversarial examples using a surrogate model [47]. This is the approach we take, whereby we generate adversarial examples for our surrogate model using the Jacobian-based Saliency Map Attack (JSMA) [48] [9], which are then tested against our target model.

Figure 2 shows the confusion matrix of *untargeted* JSMA on both (a) the target model and (b) the surrogate model, where $\theta = 0.1$ and $\gamma = 0.1$. These two parameters specify the amount of perturbation to introduce to the original feature set (θ) and the maximum fraction of features to influence (γ). Both confusion matrices now exhibit a large amount of misclassification between the predicted labels and the ground-truth labels. It is important to reiterate that the adversarial examples were crafted against the surrogate model using JSMA, and then tested against the target model.

Figure 3 shows a similar effect for *targeted* JSMA. The untargeted attack does not specify the desired label to misclassify a sample as, whereas for the targeted attack we specifically want to misclassify each attack as benign. The surrogate model is significantly impacted and the majority of samples are successfully misclassified as benign (Figure 3b). For the target model, whilst the effect of targeted misclassification to the benign class is not quite as prominent, nevertheless, the model performance is still severely degraded (Figure 3a).


(a)

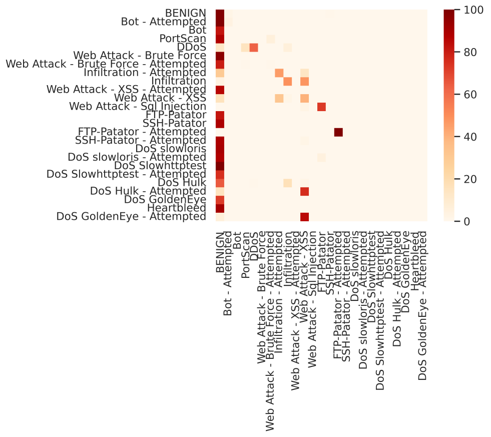

(b)

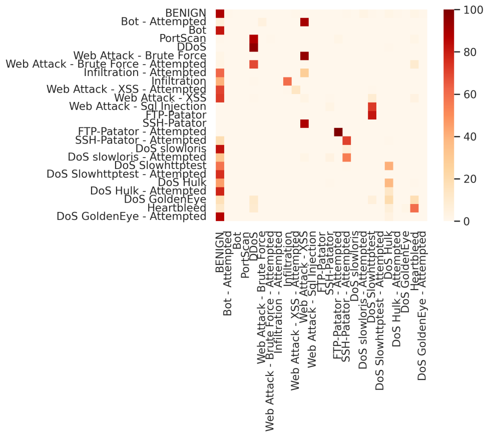
Figure 2: Untargeted JSMA ($\theta = 0.05$ and $\gamma = 0.02$) against (a) Target model (Scikit-learn), (b) Surrogate model (Keras).

3.4. Functionality-Preservation of Adversarial Example Generation

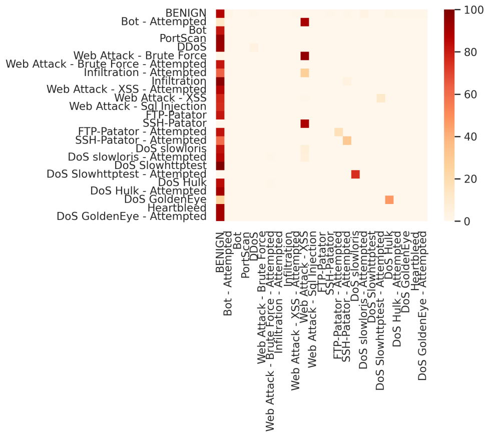
Adversarial attacks in computer vision rely on the manipulation of features (i.e., pixel intensity) that are unnoticeable to the human visual system. In our case study, whilst it is possible to manipulate features to provoke misclassification, it is important to assess whether the resulting features remain within the expected distribution of the data, such that it may be feasible to curate an attack that remains unnoticeable to a human observer, whilst exhibiting the intended underlying attack behavior. We perform systematic experimentation with the JSMA parameters θ and γ. We conducted our experiment using all paired combinations of $\theta = (0.8, 0.5, 0.1, 0.05)$ and $\gamma = (0.8, 0.5, 0.1, 0.05, 0.02)$, and study the feature distribution in comparison to the statistical distribution of the original data.

Figure 4 shows a parallel coordinates plot for a subset of the feature domain, to highlight the minimum, maximum and median of features for both the original dataset and the compromised adversarial examples. We observe that the discrepancies between the original data and the adversary examples are clearly visible with JSMA parameters $\theta = 0.1$ and $\gamma = 1.0$. Therefore, the crafted examples may either be clearly noticeable to a human observer investigating an attack, or the features may no longer satisfy the intended attack behavior.

Figure 5 shows the distribution of original and adversarial examples where



(a)



(b)

Figure 3: Targeted JSMA for benign class ($\theta = 0.05$ and $\gamma = 0.02$) against (a) Target model (Scikit-learn), (b) Surrogate model (Keras).

$\theta = 0.05$ and $\gamma = 0.02$. Importantly here, the difference between the adversarial and original distributions is unnoticeable, meaning that such adversarial attacks would be unlikely to be identified through statistical methods, and may well exhibit sufficient similarity to the underlying attack sequence in the packet flow communication. We can therefore consider this specific set of adversarial examples to be *functionality-preserving*.

Figure 6 shows the confusion matrix of *targeted* JSMA on both (a) the target model and (b) the surrogate model, where $\theta = 0.05$ and $\gamma = 0.02$, as determined to be a suitable set of parameters for functionality-preservation. Whilst a clear diagonal can be observed on each matrix, this is still severely degraded from the original result shown in Figure 1, whilst also noting that functionality of the adversarial cases would likely be preserved. Such parameter constraints would mean that not all adversarial examples will achieve misclassification, and some classes may well be more robust against the generation of adversarial example. Table 3 provides a detailed summary of the results, showing the number of successful targeted attacks for the constrained adversarial examples, for each of the possible attack classes. Overall, we observe that **90.25% of the attacks were able to evade detection**, with all DoS slowhttptest, and Heartbleed being misclassified, as well as the majority of other DoS attacks and Web Attack SQL Injection. Excluding the benign class that we did not try to perturb, all classes were found to exhibit some

Traffic Type	Original	Correct after JSMA	Successful Attack Percentage
FTP-Patator	82	0	100.00
Web Attack - Sql Injection	76	0	100.00
Heartbleed	91	0	100.00
DoS GoldenEye	72	0	100.00
DoS Hulk - Attempted	91	0	100.00
DoS Hulk	85	0	100.00
DoS Slowhttptest - Attempted	75	0	100.00
DoS Slowhttptest	100	0	100.00
DoS slowloris - Attempted	93	0	100.00
DoS slowloris	90	0	100.00
SSH-Patator	89	0	100.00
DoS GoldenEye - Attempted	91	0	100.00
Web Attack - XSS	90	0	100.00
Infiltration - Attempted	90	0	100.00
Web Attack - Brute Force - Attempted	82	0	100.00
Web Attack - Brute Force	97	0	100.00
PortScan	94	0	100.00
Bot	82	0	100.00
Infiltration	101	1	99.01
Bot - Attempted	103	4	96.12
DDoS	97	6	93.81
SSH-Patator - Attempted	91	26	71.43
Web Attack - XSS - Attempted	86	70	18.60
FTP-Patator - Attempted	100	94	6.00
Total	2063	201	90.25

Table 3: Percentage of successful attacks, target='benign', by class ($\theta = 0.05$ & $\gamma = 0.02$).

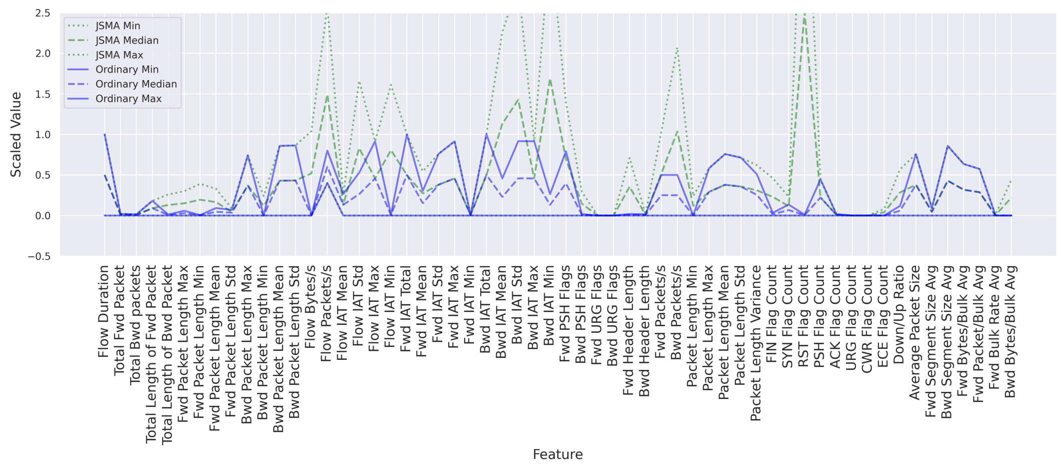


Figure 4: Parallel Coordinates to show the distribution of original features versus JSMA features ($\theta = 0.1$ and $\gamma = 1.0$). It can be observed that the perturbed JSMA features significantly exceed the expected range of the original traffic features.

degree of vulnerability, as demonstrated by successful attacks.

3.5. Summary of the Adversarial Attack Stage

We provide a comprehensive study on adversarial attack generation against a well-trained machine learning model for network traffic classification. We show that whilst evasion attacks are significantly more challenging to conduct within the constraints of the original data distribution, attacks of this nature are still achievable. As illustrated in Table 3, the fact that at least one instance of each attack can evade the classification model highlights the potential vulnerabilities exhibited by such a learning system. We affirm the often stated maxim that whilst defenders need to be successful in detection every time, attackers only need to be successful in their attack *once* in order to achieve their goal.

For this study, we have concentrated on the use of JSMA since it can provide targeted and untargeted attacks, and because the parameters allow control of the number of features to perturb and the amount of perturbation to introduce. Recent work has explored alternative adversarial techniques including Fast Gradient Sign Method (FGSM), Basic Iterative Method (BIM), and Projected Gradient Descent (PGD) that can also be configured to modify fewer or specific features [26]. Libraries such as the Adversarial Robustness Toolkit also support masking parameters to modify select features [17].

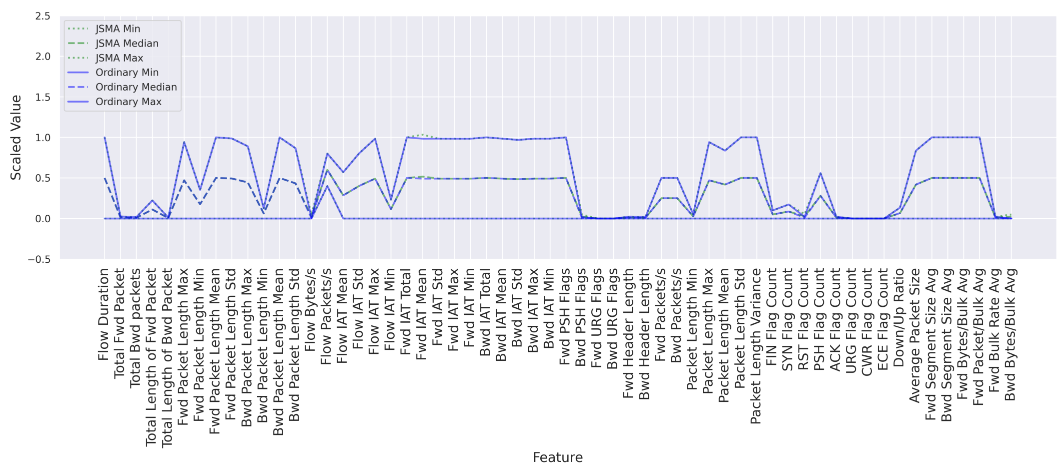


Figure 5: Parallel Coordinates to show the distribution of original features versus JSMA features ($\theta = 0.05$ and $\gamma = 0.02$). The perturbed JSMA features are within the expected range of the original traffic features.

It is important to note that ML-based Intrusion Detection Systems may well include a mixture of categorical, continuous, and discrete features[49]. JSMA uses random perturbations of continuous features to generate adversarial examples. Features such as destination port number and protocol are *nominal* attributes that should not be treated as numerical and should not be perturbed in this same way. For the CICIDS2017 dataset used in this study, we use the packet flow data, which consists primarily of count data and statistical-based features derived from count data, and only perform perturbation of the continuous features. Nevertheless, additional logical and mathematical constraints should also be considered in future studies such that statistical features are accurate. Consequently, the crafting of adversarial examples within constrained domains poses unique challenges compared to much of the prior work from the computer vision domain [14].

4. Hierarchical Classification for Model Robustness

Having successfully compromised the network traffic classification model, we now explore the defensive strategies to improve the robustness of the classifier against such adversarial attacks. Previous works have often retrained the classifier using a set of curated adversarial examples [50] [35] [51]; however, this approach is not scalable and only provides a retrospective defense

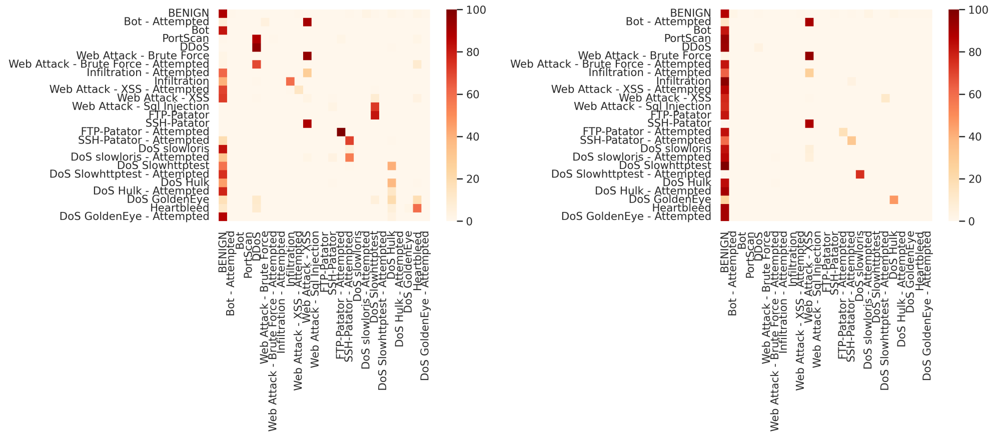


Figure 6: Targeted JSMA for benign class ($\theta = 0.05$ and $\gamma = 0.02$) against (a) Target model (Scikit-learn), (b) Surrogate model (Keras).

after a compromise has been identified. Instead, we restructure the attack surface where misclassification can occur by using a hierarchical classification layer. By reframing the classification task such that we provide a hierarchical label, we effectively reduce the attack surface available to the adversarial methods, such that the amount of perturbation required to misclassify a label becomes greater than compared to a flat classification layer, and outside of the distribution of the original dataset. For our network traffic classification case study, this means we can reduce the number of possible output states from 25 to approximately 2-5 states at any level of the hierarchy. We propose that a hierarchical classification approach is less susceptible to the influence of adversarial examples compared to the classifier used previously in this work. For our experimentation, we incorporate the hierarchical learning library, HiClass [52], with our previous implementation in Section 3.

4.1. Hierarchical Classification

Traditional classification models utilize a flat output layer where each class is associated with a probability. However, many large classification tasks will exhibit some natural hierarchy. For example, within the CICIDS2017 we can observe that there are multiple variants of denial of service attacks (DoS), as well as different web-based attacks, and patator attacks on FTP and SSH services. The grouping of similar classes may therefore mean that misclassification between classes that exist in separate groupings may become more challenging for an adversarial attack to achieve.

Koller and Sahami's [53] seminal work on local classification approach established the foundations for researchers to expand the field of hierarchical classification using local classifiers. Hierarchical models often employ multiple multi-class models (e.g. One-Vs-All/One-Vs-Rest) and therefore may be considerably larger than flat models. There are three generally accepted forms of local classifier: Local Classifier Per Node (LCPN), Local Classifier Per Parent Node (LCPPN), and Local Classifier Per Level (LCPL).

- **Local Classifier Per Node (LCPN):** A binary classifier is trained for every node in the class hierarchy (excluding the root node). The advantage is that this becomes naturally multi-class. The disadvantage is that because each node is trained independently, it is theoretically possible to have inconsistency between levels (e.g., a coarse class prediction of 'DoS'=False and a finer prediction of 'Dos Hulk'=True). Without methods to resolve these class-membership inconsistencies, incompatible predictions between coarse and fine classes are possible.

- **Local Classifier Per Parent Node (LCPPN):** A multi-class classifier is trained for each parent's child nodes. This method respects hierarchy constraints while avoiding class-membership inconsistencies.

- **Local Classifier Per Level (LCPL):** A multi-class classifier is trained for each level of the hierarchy. Simplistic implementations take the output of the classifier at each level, presenting this as the final classification. This method can result in class-membership inconsistencies. Approaches avoiding class-membership inconsistencies include a top-down approach where the class prediction at coarse levels restricts classification at finer levels to only child nodes of the previous level.

4.2. Hierarchical Output Class

Typically, a machine learning classifier is trained to provide a single label from a set of possible labels. This approach extends from object classification through to natural language generation, and in our case study so far, predicting a label that defines the associated network attack type, based on the characteristics of the input network traffic. We naturally think about how these output classes may group together, such that similar attack types are grouped together, and then assigned a specific sub-group. For example, DoS slowloris, DoS Slowhttptest, DoS Hulk, and DoS GoldenEye, can naturally group together as a DoS group. Similarly, we can group together all web attacks. How we define a suitable set of hierarchical class labels could be achieved in multiple ways: either based on analyst's domain knowledge, text label similarities, feature similarities, or some higher attributes related to the data. For the purpose of this study, we did not explore hyper-parameter searching as this was not the core focus of this study; however, this remains an area of future research. We perform our experimentation using 7 different hierarchical schemes: K-Means; Ward; Average; Complete; Single; Researcher Defined; and Dataset.

Figure 7 shows two of the initial hierarchy schemes: Researcher Defined and Dataset. The *dataset* scheme (a) is based on the implicit hierarchy that is present in the original dataset by how attacks have been grouped by the original authors. The *researcher* scheme (b) is defined by ourselves based on our domain knowledge.

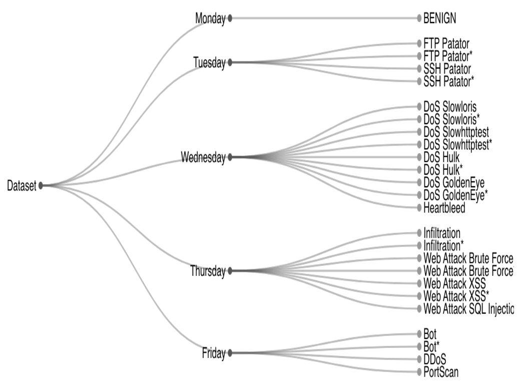
4.2.1. Automated Hierarchical Clustering - k-Means

We optimistically expect hierarchies, defined by human experts, to be most robust; however, building such a hierarchy is a skilled task and introduces an overhead for analysts. To assist the process of generating class hierarchies, we consider automated cluster techniques.

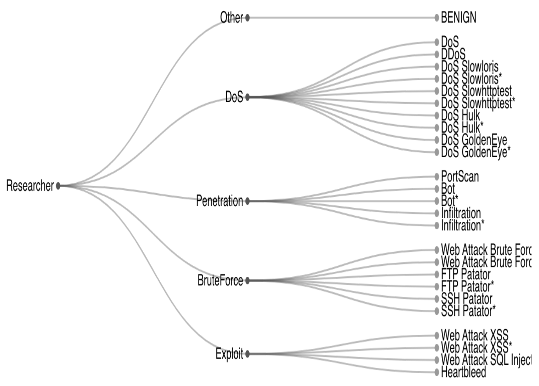
Firstly, we use k-means clustering to identify the hierarchical relationships, where k specifies the number of suitable clusters to discover. Figure 8 shows the hierarchy as determined by k-means clustering using a top-down approach. In this example, $k = 5$, for which we can see the majority of classes cluster in group 0.

4.2.2. Automated Hierarchical Clustering - Agglomerative

Agglomerative clustering is a type of hierarchical clustering that differs from k-means in that it is a bottom-up approach. It begins with n clusters and sequentially combines similar clusters until all clusters belong to a single large cluster. The approach is more computationally expensive than k-means, however the scheme is especially applicable for arranging clusters into a natural hierarchy. The main parameters of agglomerative clustering are affinity and linkage, where affinity refers to the distance metrics used (euclidean distance), and linkage refers to how the distance between clusters should be measured. We study four possible linkage schemes: 'Ward' [54], 'Average', 'Complete', and 'Single'. For a given pair of clusters, 'Single' will



(a)



(b)

Figure 7: Hierarchies assembled by human reasoning: (a) original data set structure, (b) researcher-defined structure. Attempted classes are denoted by $*$

.

calculate the minimum distance between any pair of observations within each of the clusters, whilst 'Complete' will calculate the maximum distance between a pair of observations. 'Average' will take an average distance based on all pairs of observations within each clusters. 'Ward' is similar to average; however, it utilizes the variance of the observations within each cluster to then calculate the average. The scikit-learn library provides a function for Agglomerative Clustering that supports all four linkage schemes.

Algorithm 1 and Algorithm 2 show the process of constructing hierarchies for either an unlabeled or labeled dataset respectively, by generating labels

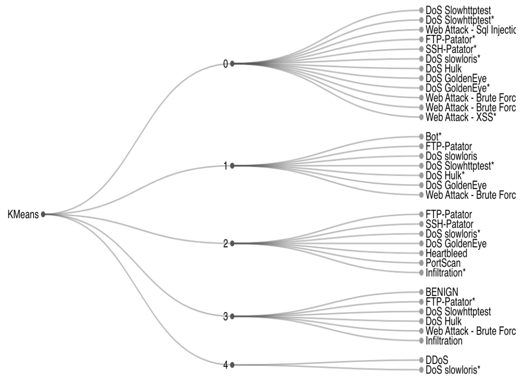


Figure 8: Hierarchy based on divisive clustering: KMeans.

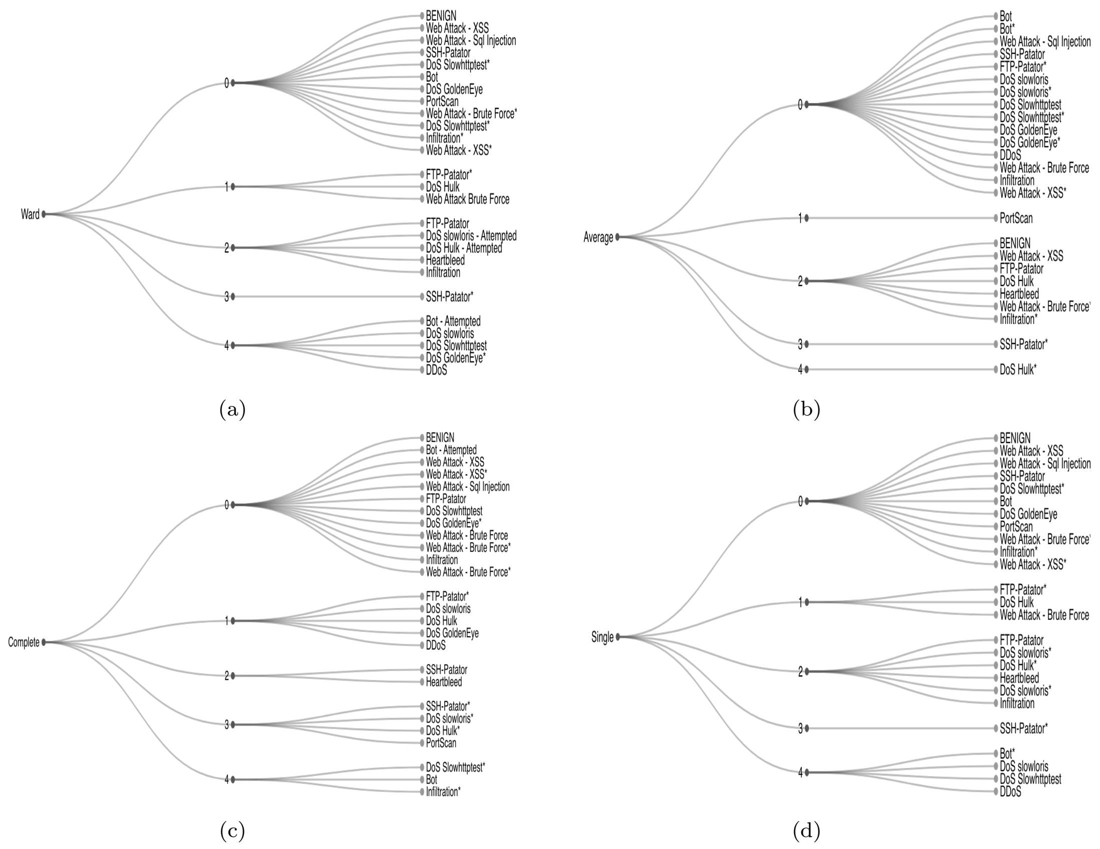


Figure 9: Hierarchies from agglomerative clustering: (a) Ward, (b) Average, (c) Complete, and (d) Single. Attempted classes are denoted by $*$

Algorithm 1: Create hierarchical labels from an unlabeled flat dataset.

```
finecluster = AgglomerativeClustering(n_clusters=15, affinity=affinity, linkage=linkage);
finecluster.fit_predict(X);
coarsecluster = AgglomerativeClustering(n_clusters=5, affinity=affinity, linkage=linkage);
coarsecluster.fit_predict(X);
top_layer_labels = coarsecluster.labels_
bottom_layer_labels = finecluster.labels_
for i in range len(X):
    hier_labels.append( [str(top_layer_labels[i]),str(bottom_layer_labels[i])])
```

Algorithm 2: Create hierarchical labels from a labeled flat dataset.

```
cluster = ClusterAlgorithm(n_clusters= 5)
cluster.fit_predict(X)
for class in range (0,max_class):
    initialize class groups
for i in range len(X):
    append cluster_label to relevant ground-truth class
for class_name in all_classes:
    find majority cluster label for ground-truth label
Initialize hier_labels = empty
for i in range(len(X)):
    append [str(majority_cluster_label),str(ground_truth_label[i])] to hier_labels
```

for both coarse and fine layers. The analyst can provide an integer to define the number of expected coarse and fine classes for the hierarchy, or if not provided, this will default to five.

4.3. Deployment of Hierarchical Classification

With the set of possible class hierarchies available, we can build a two-layer hierarchical classifier using the Python library 'HiClass' [52], for which we can then test each hierarchical structure for the model output layer. This library integrates with the scikit-learn MLP classifier, and so we can easily incorporate our previous target model from Section 3. This also enables us to compare the results of the hierarchical approach with that of the original flat model used previously. For hierarchical learning models we can evaluate the model performance using modified performance metrics proposed by Kritchenko et al. [55]: hierarchical precision (hP), hierarchical Recall (hR) and hierarchical F1-Score (hF).

4.4. Results of Hierarchical Classification

Figure 10 shows the F1-Score Macro Average for the original flat MLP model with both the normal data and the compromised adversarial example data. Similarly, it also shows the Hierarchical F1-Scores (HF) for both the coarse and fine layers of the hierarchical model, for both the normal data and the compromised adversarial example data. We show this for each of the seven clustering strategies. It can be seen that LCPPN hierarchical classifiers can improve the robustness of classification, as measured by F1-Score. In particular, We note an average improvement of 84.85% in classification of presented adversarial examples:an increase of 0.28 from 0.33 to 0.61 This improvement can be seen in the difference between the orange and brown horizontal lines. Moreover, we note a slight improvement in robustness of classification in general when only original unperturbed samples are presented.

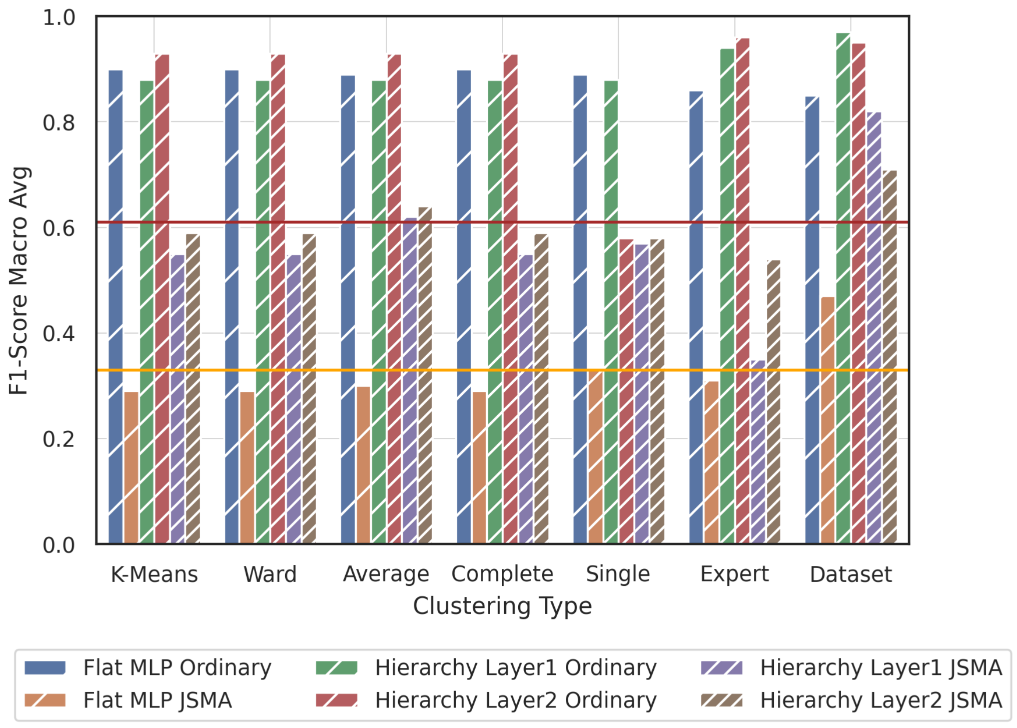
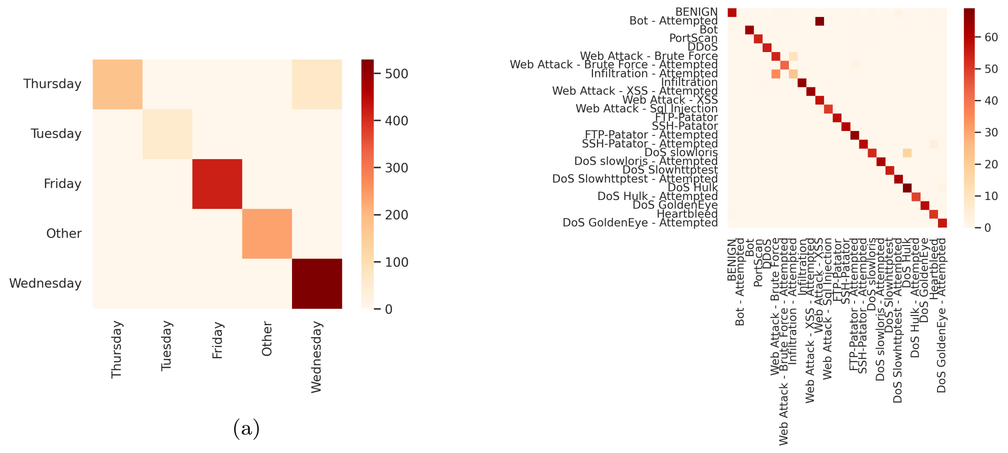


Figure 10: Bar plot to show robustness improvement by comparing appropriate F1-Score metrics across different LCPPN hierarchies. It can be seen that all hierarchies have improved F1-Scores under adversarial conditions. We highlight two important results: the orange horizontal line indicates the mean F1-Score for 'Flat MLP JSMA' across the hierarchies. The brown horizontal line indicates the mean 'Hierarchy Layer 2 JSMA' across the hierarchies. All hierarchies also improve the F1-Score when no adversarial traffic is present.

Our hierarchical defense achieves good robustness regardless of the presence of adversarial learning attack. We note that all of the hierarchies improved robustness as measured by the F1-Score under adversarial conditions. Moreover, the hierarchical approach also improves the F1-Score when no adversarial traffic is present. Figure 11 shows the confusion matrix for both (a) the coarse layer and (b) the fine layer, that reveals fewer misclassifications compared to the original model performance. We agree with Qian *et al.* [38]:

classifiers with fewer output classes are more robust. However, Hierarchical F1-Score for the 'single' (Figure 9d) remains steady. We believe that because this hierarchy has a particularly large subclass, that the benefit of a hierarchical structure is not fully realized, suggesting that some hierarchies will be more robust than others. Furthermore, we find that flat models that are most susceptible to adversarial samples gain most from implementing a hierarchy.



(a)

(b)

Figure 11: Confusion matrices for (a) coarse layer and (b) fine layer that shows fewer misclassifications for the original dataset when utilising a hierarchical classification model.

5. Discussion

In this section, we discuss topics proceeding from this work, including: how hierarchical learning can help improve robustness and assist in blocking the transferability of adversarial examples. We inspect the classifiers and clustering techniques used in hierarchical learning, and subsequently discuss how a hierarchical classifier might be robustness benefits of hierarchical classifiers and hierarchies, clustering techniques, blocking transferability, and subsequently discuss how we might feasibly attack a hierarchical classifier.

5.1. Benefits of Hierarchical Classifiers

Hierarchical classification is a simple ensemble technique that offers promise in protecting machine learning systems from adversarial examples as we explored for intrusion detection systems.

We posit that for hierarchical models with a few parent nodes and more child nodes, the higher layers are more robust than the lower layers. This offers advantages for network defense because misclassification among subclasses (e.g. DoS Hulk and DoS slowhttp) is a less serious prospect than misclassification among superclasses (e.g. DoS and Benign) [56]. Indeed, Jeanneret et al. [57] note that hierarchical attacks aiming for severe outcomes are less successful in evading detection. Hierarchical classifiers offer improvements in F1-Score. Moreover, they may also reduce memory consumption, disk usage, and training time [52].

5.2. Hierarchies

Simple classifiers with fewer output classes are more robust [38]. It follows that in hierarchical models with a few parent nodes and more child nodes, the higher layers are more robust than the lower layers; however, if the classification task is truly hierarchical, by the nature of the class and subclasses, Layer 1 and Layer 2 will intuitively have aligned gradients. For example, the classifier for the parent node 'Denial of Service' will likely have gradients aligned with classifiers detecting specific subclasses of Denial of Service attacks such as 'Dos Hulk'. Intuitively, the construction of a traditional hierarchical classifier means that Layer 1 and Layer 2 ought to have aligned gradients. Using strong ensemble classifiers with misaligned gradients in a hierarchy may help improve robustness for hierarchical classifiers, combining the advantages of misaligned gradients and the robustness improvement gained through hierarchical classification.

Top-down methods restrict classification at finer levels to only child nodes of the previous level, meaning that lower levels also have fewer classes, further increasing robustness. We recognize a disadvantage that when descending a hierarchy there is no way to retrace one's steps. Therefore, misclassification at a coarse level might forbid correct classification at the fine level; however, improved robustness may be considered sufficient for this trade-off.

5.2.1. Clustering Techniques

Divisive and agglomerative clustering techniques could be employed to find other groupings of classes. Hierarchical labels simplify the generation of hierarchical classifiers. Our semi-automated techniques could be used to build hierarchical labels for unlabelled or labelled flat datasets.

We note that any clustering algorithm is unlikely to generate clusters that correspond exactly to the known classes. Indeed, in our experiments we found that a particular known class could be dispersed among multiple clusters. Our objective is to explore the generation of hierarchies rather than to label datasets. Clustering techniques are unlikely to perform as well as supervised or semi-supervised learning techniques [58]. Instead, we use clustering only as a guide to which ground-truth labels could be grouped.

We note that for hierarchies with diverse fine-classes covered by one large coarse-class. For example, Single has a particularly large coarse-class. The prospect of misclassification at the coarse level is likely proportionate to the number of classes in that branch of the Hierarchy.

5.3. Blocking Transferability

The transferability property of adversarial examples can effectively be used to perform gray-box or black-box attacks. Breaking transferability is an important goal. In our experiments we observe a reduction in transferability to our hierarchical model. Other work considers other ways to block the transferability of adversarial examples. For example, Hosseini et al. propose NULL-labeling as a form of adversarial training [59].

5.4. Effectively Attacking Hierarchies

Our hierarchical defense improves robustness against adversarial learning attacks. Attacks causing misclassification between subclasses are less severe than attacks causing misclassification at class level. Successful adversarial learning attacks on intrusion detection systems must preserve the functionality of malicious network traffic. Research in other domains might apply in cyber security. For example, Jeanneret et al. [57] consider hierarchy aware attacks that generate adversarial perturbations considering the hierarchical distance between labels. Moreover, they consider the severity of Hierarchical Attacks and apply Curriculum Learning to enhance the performance of models through learning concepts from coarse to fine.

6. Conclusions and Future Work

In this paper, we propose hierarchical learning as a defensive strategy to mitigate against adversarial machine learning attacks. Our defense is independent of the attack algorithm and based on a robust hierarchical learning scheme. When under attack, our defense achieves accuracy scores close to the accuracy of the original flat model with no adversarial machine learning attacks. Our approach is intended to be representative of a functionality-preserving black-boxattack; however, our approach may be considered a gray-box attack, because inevitably we have some knowledge of our underlying models.

Our results reveal that hierarchies can help models perform better under adversarial conditions than their equivalent flat model. Moreover, hierarchies can also improve the F1-Score when no adversary perturbed traffic is present. Our work compares the performance of seven hierarchies constructed via different methods: Dataset file structure, Human researcher (expert) defined structure, Divisive Clustering (K-Means), Agglomerative Clustering using the euclidean distance and different linkages (Ward, Average, Complete, Single). We find each of these hierarchies performs better than the original flat model in the presence of adversary perturbed traffic.

Future experiments could include more samples in training and test data. We believe our study to be representative; however, some elements of our results are contingent on random hybrid sampling and the effectiveness of the clustering algorithms. We note that in some cases, only a few samples are present to determine the coarse-level classification, and so additional testing with a larger yet well-balanced data corpus would be beneficial. To aid in replication of our results and further research, we provide the source in our repository: https://github.com/mccarthyajb/HL-NTAC.

Acknowledgements

Funding: This work was supported by the University of the West of England Partnership PhD scheme in collaboration with Techmodal Ltd.

References

[1] M. Ell, R. Gallucci, Cyber security breaches survey 2022: Statistical release (2022).

[2] B. Biggio, F. Roli, Wild patterns: Ten years after the rise of adversarial machine learning, Pattern Recognition 84 (2018) 317–331.

[3] A. McCarthy, E. Ghadafi, P. Andriotis, P. Legg, Functionality-preserving adversarial machine learning for robust classification in cybersecurity and intrusion detection domains: A survey, Journal of Cybersecurity and Privacy 2 (1) (2022) 154–190.

[4] A. McCarthy, P. Andriotis, E. Ghadafi, P. Legg, Feature vulnerability and robustness assessment against adversarial machine learning attacks, in: 2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), IEEE, 2021, pp. 1–8.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: International Conference on Learning Representations, ICLR 2014, 2014, p. 0, 2nd International Conference on Learning Representations, ICLR 2014 ; Conference date: 14-04-2014 Through 16-04-2014.

[6] N. Papernot, P. McDaniel, A. Sinha, M. P. Wellman, Sok: Security and privacy in machine learning, in: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2018, pp. 399–414.

[7] J. Zhang, C. Li, Adversarial examples: Opportunities and challenges, IEEE transactions on neural networks and learning systems (2019).

[8] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, M. Colajanni, Modeling realistic adversarial attacks against network intrusion detection systems, Digital Threats: Research and Practice (2021).

[9] A. U. H. Qureshi, H. Larijani, M. Yousefi, A. Adeel, N. Mtetwa, An adversarial approach for intrusion detection systems using jacobian saliency map attacks (jsma) algorithm, Computers 9 (3) (2020) 58.

[10] Z. Lin, Y. Shi, Z. Xue, Idsgan: Generative adversarial networks for attack generation against intrusion detection, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2022, pp. 79–91.

[11] N. Papernot, P. McDaniel, I. Goodfellow, Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, arXiv preprint arXiv:1605.07277 (2016).

[12] D. Yang, Z. Xiao, W. Yu, Boosting the adversarial transferability of surrogate model with dark knowledge, arXiv preprint arXiv:2206.08316 (2022).

[13] J. Zhang, C. Chen, Y. Xiang, W. Zhou, Y. Xiang, Internet traffic classification by aggregating correlated naive bayes predictions, IEEE transactions on information forensics and security 8 (1) (2012) 5–15.

[14] R. Sheatsley, B. Hoak, E. Pauley, Y. Beugin, M. J. Weisman, P. McDaniel, On the robustness of domain constraints, in: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 495–515.

[15] G. Apruzzese, P. Laskov, E. M. de Oca, W. Mallouli, L. B. Rapa, A. V. Grammatopoulos, F. D. Franco, The role of machine learning in cybersecurity, Digital Threats: Research and Practice (2022).

[16] K. Grosse, P. Manoharan, N. Papernot, M. Backes, P. McDaniel, On the (statistical) detection of adversarial examples, arXiv preprint arXiv:1702.06280 (2017).

[17] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. Molloy, B. Edwards, Adversarial robustness toolbox v1.2.0, CoRR 1807.01069 (2018). URL `https://arxiv.org/pdf/1807.01069`

[18] Y. Wu, D. Wei, J. Feng, Network attacks detection methods based on deep learning techniques: a survey, Security and Communication Networks 2020 (2020).

[19] M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani, A detailed analysis of the kdd cup 99 data set, in: 2009 IEEE symposium on computational intelligence for security and defense applications, IEEE, 2009, pp. 1–6.

[20] J. McHugh, Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, ACM Transactions on Information and System Security (TISSEC) 3 (4) (2000) 262–294.

[21] V. G. Cerf, 2021 internet perspectives, IEEE Network 35 (1) (2021) 3–3.

[22] M. McKeay, Akamai state of the internet / security: A year in review, `http://akamai.com/soti` (2020).

[23] R. SimilarWeb, Top websites ranking", `https://www.similarweb.com/top-websites/` (2022).

[24] S. Kok, A. Abdullah, N. Jhanjhi, M. Supramaniam, A review of intrusion detection system using machine learning approach, International Journal of Engineering Research and Technology 12 (1) (2019) 8–15.

[25] N. Martins, J. M. Cruz, T. Cruz, P. H. Abreu, Adversarial machine learning applied to intrusion and malware scenarios: a systematic review, IEEE Access 8 (2020) 35403–35419.

[26] Á. L. P. Gómez, L. F. Maimó, A. H. Celdrán, F. J. G. Clemente, F. Cleary, Crafting adversarial samples for anomaly detectors in industrial control systems, Procedia Computer Science 184 (2021) 573–580.

[27] D. Gonzalez-Cuautle, A. Hernandez-Suarez, G. Sanchez-Perez, L. K. Toscano-Medina, J. Portillo-Portillo, J. Olivares-Mercado, H. M. Perez-Meana, A. L. Sandoval-Orozco, Synthetic minority oversampling technique for optimizing classification tasks in botnet and intrusion-detection-system datasets, Applied Sciences 10 (3) (2020) 794.

[28] J. M. Johnson, T. M. Khoshgoftaar, Survey on deep learning with class imbalance, Journal of Big Data 6 (1) (2019) 1–54.

[29] R. Sheatsley, N. Papernot, M. Weisman, G. Verma, P. McDaniel, Adversarial examples in constrained domains, arXiv preprint arXiv:2011.01183 (2020).

[30] T. Bai, J. Luo, J. Zhao, B. Wen, Recent advances in adversarial training for adversarial robustness, arXiv e-prints (2021) arXiv–2102.

[31] M. Amer, T. Maul, Weight map layer for noise and adversarial attack robustness, arXiv preprint arXiv:1905.00568 (2019).

[32] T. P. Lillicrap, D. Cownden, D. B. Tweed, C. J. Akerman, Random synaptic feedback weights support error backpropagation for deep learning, Nature communications 7 (1) (2016) 1–10.

[33] J. H. Metzen, T. Genewein, V. Fischer, B. Bischoff, On detecting adversarial perturbations, arXiv preprint arXiv:1702.04267 (2017).

[34] N. Carlini, D. Wagner, Adversarial examples are not easily detected: Bypassing ten detection methods, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, 2017, pp. 3–14.

[35] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel, Ensemble adversarial training: Attacks and defenses, in: 6th International Conference on Learning Representations, ICLR 2018, 2018.

[36] S. Wang, X. Wang, P. Zhao, W. Wen, D. Kaeli, P. Chin, X. Lin, Defensive dropout for hardening deep neural networks under adversarial attacks, in: Proceedings of the International Conference on Computer-Aided Design, 2018, pp. 1–8.

[37] P. A. A. Resende, A. C. Drummond, A survey of random forest based methods for intrusion detection systems, ACM Computing Surveys (CSUR) 51 (3) (2018) 1–36.

[38] S. Qian, D. Kalimeris, G. Kaplun, Y. Singer, Robustness from simple classifiers, arXiv preprint arXiv:2002.09422 (2020).

[39] F. Chollet, et al., Keras, `https://github.com/fchollet/keras` (2015).

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[41] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization., ICISSp 1 (2018) 108–116.

[42] A. H. Lashkari, M. S. I. Mamun, A. A. Ghorbani, et al., Characterization of tor traffic using time based features., 2017.

[43] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, A. A. Ghorbani, Characterization of encrypted and vpn traffic using time-related, 2016.

[44] R. Hofstede, I. Drago, A. Sperotto, A. Pras, Ethernet flow monitoring with ipfix, in: TERENA Networking Conference 2011, Trans-European Research and Education Networking Association, 2011, p. 23, tERENA Networking Conference 2011 ; Conference date: 16-05-2011 Through 19-05-2011.

[45] G. Engelen, V. Rimmer, W. Joosen, Troubleshooting an intrusion detection dataset: the cicids2017 case study, in: 2021 IEEE Security and Privacy Workshops (SPW), IEEE, 2021, pp. 7–12.

[46] R. M. Pereira, Y. M. Costa, C. N. Silla Jr, Toward hierarchical classification of imbalanced data using random resampling algorithms, Information Sciences 578 (2021) 344–363.

[47] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, A. Swami, Practical black-box attacks against machine learning, in: Proceedings of the 2017 ACM on Asia conference on computer and communications security, 2017, pp. 506–519.

[48] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: 2016 IEEE European symposium on security and privacy (EuroS&P), IEEE, 2016, pp. 372–387.

[49] R. Sheatsley, N. Papernot, M. J. Weisman, G. Verma, P. McDaniel, Adversarial examples for network intrusion detection systems, Journal of Computer Security (Preprint) (2022) 1–26.

[50] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: International Conference on Learning Representations, 2018.
URL `https://openreview.net/forum?id=rJzIBfZAb`

[51] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572 (2014).

[52] F. M. Miranda, N. Köehnecke, B. Y. Renard, Hiclass: a python library for local hierarchical classification compatible with scikit-learn, arXiv preprint arXiv:2112.06560 (2021).

[53] D. Koller, M. Sahami, Hierarchically classifying documents using very few words, in: Proceedings of the Fourteenth International Conference on Machine Learning, 1997, pp. 170–178.

[54] F. Murtagh, P. Legendre, Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion?, Journal of classification 31 (3) (2014) 274–295.

[55] S. Kiritchenko, S. Matwin, R. Nock, A. F. Famili, Learning and evaluation in the presence of class hierarchies: Application to text categorization, in: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, 2006, pp. 395–406.

[56] A. Ma, A. Virmaux, K. Scaman, J. Lu, Improving hierarchical adversarial robustness of deep neural networks, arXiv preprint arXiv:2102.09012 (2021).

[57] G. Jeanneret, J. C. Pérez, P. Arbelaez, A hierarchical assessment of adversarial severity, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 61–70.

[58] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, J. Ma, Su-ids: A semi-supervised and unsupervised framework for network intrusion detection, in: International conference on cloud computing and security, Springer, 2018, pp. 322–334.

[59] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, R. Poovendran, Blocking transferability of adversarial examples in black-box learning systems, arXiv preprint arXiv:1703.04318 (2017).