CyCon 2024: Over the Horizon
16th International Conference on Cyber Conflict
C. Kwan, L. Lindström, D. Giovannelli, K. Podiņš, D. Štrucl (Eds.)
2024 © NATO CCDCOE Publications, Tallinn

Defeating and Improving Network Flow Classifiers Through Adversarial Machine Learning

Yannick Merkli
LatticeFlow
Zurich, Switzerland
ymerkli@latticeflow.ai

Roland Meier
armasuisse Science and Technology
Cyber-Defence Campus
Thun, Switzerland
roland.meier@ar.admin.ch

Martin Strohmeier
armasuisse Science and Technology
Cyber-Defence Campus
Thun, Switzerland
martin.strohmeier@ar.admin.ch

Vincent Lenders
armasuisse Science and Technology
Cyber-Defence Campus
Thun, Switzerland
vincent.lenders@ar.admin.ch

Abstract: Recent work has shown that machine learning models can be vulnerable to an adversary crafting targeted inputs designed to cause mispredictions. This is critical in security-related applications such as network intrusion detection systems. While past attacks such as mimicry or gradient-based attacks are able to efficiently generate adversarial examples, they require potentially large input modifications, which is not effective at defeating network flow classifiers.

In this work, we show that small modifications to the input (e.g., the traffic that the attacker generates) are enough to manipulate the outcome of a classifier. We focus on minimally evasive adversarial examples to defeat tree-ensemble-based network flow classifiers. We develop an attack that builds on a previous attack introduced by Kantchelian et al. in 2016, which formulates evasion for tree ensembles as a Mixed Integer Linear Program, and which we extend by supporting discrete and categorical features, implementing per-feature evasion costs and modeling inter-feature dependencies. This makes our attack more applicable to the network flow classification problem, which typically uses diverse and interdependent input features.

We demonstrate our attack on the network flow classifier developed by Känzig et al. in 2019, which was trained to detect command and control (C&C) channels in the

Locked Shields cyber defense exercise. Our evaluation shows that minor perturbations of 1 to 4 flow features suffice to successfully fool the classifier. We further retrain the network flow classifier using state-of-the-art adversarial boosting and robust decision tree training published by Chen et al. in 2019. For example, using adversarial boosting, the resulting robust classifier shows a 62.5% increased median evasion distance while achieving equivalent precision and recall on unperturbed samples as the original classifier.

Keywords: *adversarial machine learning, traffic classification, Locked Shields*

1. INTRODUCTION

Machine learning (ML) algorithms are being applied to an ever-growing number of security-sensitive domains, such as malware detection or network intrusion detection [1]–[3]. These algorithms have proven capable of finding novel patterns and handling amounts of data that are not processable by humans, which is very beneficial in data-intensive applications. For instance, network intrusion detection systems (NIDS) protect networks by monitoring traffic and detecting anomalous behavior. Traditionally, these systems relied on experts developing a set of rules that encode known malicious behavior. However, with networks growing larger and more heterogeneous and network traffic showing higher variability and much larger volume, this approach is becoming increasingly challenging. Furthermore, this approach is unable to discover previously unknown attacks and is limited by the fact that most network traffic today is encrypted. For these reasons, researchers have proposed ML-based NIDS and have shown these to be capable of quickly and accurately identifying malicious behavior [1], [4], [5].

However, security-sensitive domains have an intrinsically adversarial nature, with adversaries actively trying to bypass any protective measures that have been put in place. Recent work on adversarial machine learning suggests that many learning algorithms are vulnerable to input perturbations. Such perturbations can happen randomly (e.g., because the environment changed) and lead to significantly degraded classification performance, as shown by Gehri et al. [6]. However, such perturbations can also happen because an adversary deliberately introduces them to cause mispredictions [7]–[10].

Successful attacks on security-sensitive ML systems can have disastrous consequences. For instance, botnets are one of the most serious threats in today's

security environment, causing malicious activities such as distributed denial of service (DDoS) attacks. Successfully defending against botnets requires efficient and accurate detection of botnet traffic, which is made challenging because of the obfuscation and resilience techniques employed by attackers. ML-based botnet detection systems are thus becoming increasingly popular and have proven to perform well in correctly labeling botnet communication [1], [2].

In this work, we focus on defeating and improving a tree-ensemble-based network flow classifier developed by Känzig et al. [1] that was trained for command and control (C&C) flow classification. In order to evaluate the classifier's robustness in an adversarial setting, we adopt the role of an attacker trying to evade the flow classifier to examine how a C&C flow needs to be modified if the classifier is not to label the resulting flow as malicious. Past approaches have focused on mimicking a known benign sample [11], [12], which requires potentially large adversarial modifications, leading to suboptimal malicious behavior. Instead of taking this approach, we focus on optimal evasion, i.e., finding the minimal adversarial modification needed to evade the classifier. This approach leads to optimal behavior for the attacker and allows for making exact guarantees of the classifier's robustness. We leverage an existing adversarial attack [7] and extend this to the network intrusion detection domain. We then apply our attack to Känzig et al.'s [1] network flow classifier and show that minor perturbations of 1 to 4 flow features suffice to successfully evade the classifier. Finally, we improve the flow classifier by applying state-of-the-art adversarial defenses.

The paper is organized as follows: Section 2 describes the background to the work described in this paper. Section 3 summarizes related work. Section 4 presents the design of our attack framework. Section 5 details the results of the evaluation. In Section 6, we conclude and discuss future research directions.

2. BACKGROUND

In this section, we briefly introduce the background to the work described in this paper.

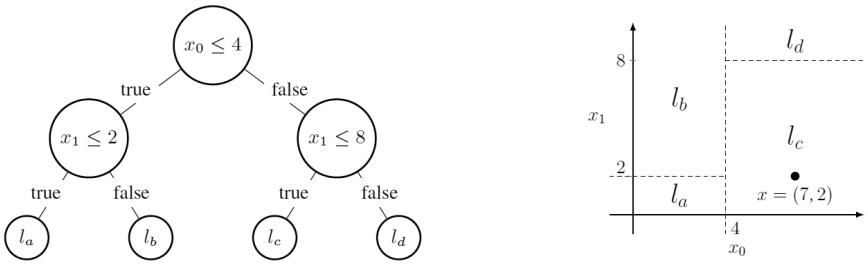
A. Decision Tree Ensemble

A decision tree consists of nodes, where each leaf node holds a prediction score, and each non-leaf node holds a logical decision rule on a given input feature and has two outgoing edges that are labeled *true* and *false*, indicating the decision path depending on the rule evaluation for an input sample x. When given an input sample, the prediction value is found by walking the tree from root to leaf such that an edge is in the decision path if and only if the associated decision rule evaluates to *true* for the

input sample. For instance, consider the simple decision tree in Figure 1 and a sample $x = (7,2)$. The decision rule $x_0 \leq 4$ evaluates to false and $x_1 \leq 8$ to true, thus the active leaf for sample x is leaf l_c. A decision tree ensemble consists of a set of decision trees, where its prediction is an aggregation of the predictions of each active leaf in each individual tree in the ensemble.

Since decision trees naturally encode human-interpretable decision rules, a key advantage of tree-based learning algorithms is interpretability. Tree ensemble classifiers are widely used in practice, especially in the security domain, where they have been shown to be superior compared to other learning algorithms, such as support vector machines (SVM) and neural networks [1], [2].

FIGURE 1: AN EXAMPLE OF A SIMPLE DECISION TREE AND ITS FEATURE SPACE PARTITION. LEAF l_c WOULD BE ACTIVE FOR SAMPLE $x = (7,2)$



B. Network Traffic Classification

Network traffic consists of bidirectional flows, whereby the term flow refers to a set of packets that are sent from a source to a destination and that share a common set of properties. Most commonly, a flow is defined by its 5-tuple (source and destination IP addresses and transport-layer ports and the transport-layer protocol).

In order to classify network flows, one has to extract a representative set of features. The most commonly used features are statistical flow features (e.g., the average packet size), which can be extracted from packet traces.

C. Mixed Integer Linear Programming

A mixed integer linear programming (MILP) problem deals with a mathematical optimization problem that consists of solely linear functions and a finite set of discrete or continuous variables. A MILP problem consists of an objective, a set of constraints and a set of variables, where the optimal solution optimizes the objective while fulfilling the set of constraints.

Since most problems cannot be translated directly into an objective function and a set of constraints, solving a problem using MILP requires a careful modeling and translation process. Once the problem has been modeled into a MILP formulation, a global optimum can be found through the use of an efficient solver.

D. Locked Shields

Locked Shields is a complex international cyber defense exercise organized annually by the NATO Cooperative Cyber Defence Centre of Excellence (CCDCOE) in Tallinn, Estonia [13]. Locked Shields focuses on realistic scenarios and simulates the full complexity of a large-scale cyber incident, including regular business IT, critical infrastructure, military systems, strategic decision-making, and legal and communication aspects. The exact scenario varies from year to year, but the exercise is generally organized as a real-time Red team vs. Blue team exercise in which the Blue teams are the training audience.

E. The Target Classifier

The classifier we analyze in this paper was developed by Känzig et al. [1] and is designed to quickly and reliably identify C&C channels in the setting illustrated in Figure 2. It uses a random forest classifier with 128 trees of maximum tree depth 10, was trained on data from the Locked Shields exercises from 2017 and 2018, and uses the 10 or 20 most important features (depending on the classifier version) listed in Table I. In the following, *top10* and *top20* will refer to classifiers that were trained on the 10 or 20 most important features, respectively.

FIGURE 2: NETWORK APPLICATION DOMAIN WITH THE DEPLOYED FLOW CLASSIFIER AS LIVE DEFENSE AGAINST A BOTMASTER THAT COMMUNICATES WITH INFECTED ENDHOSTS VIA A C&C CHANNEL

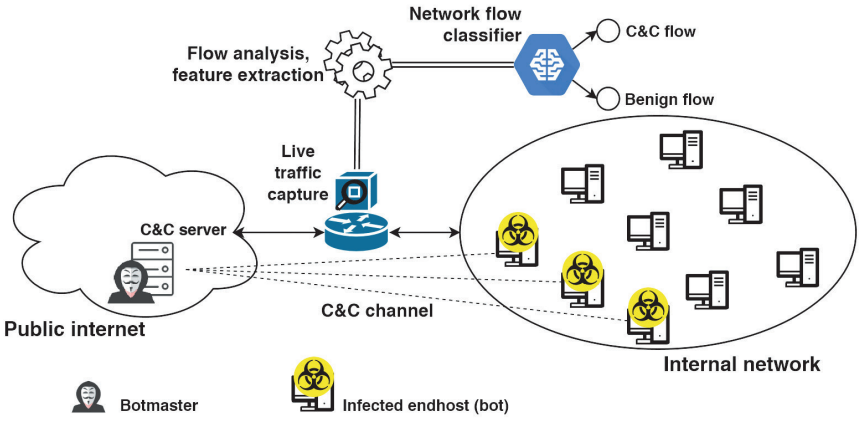


TABLE I: FEATURES ON WHICH THE TARGET FLOW CLASSIFIERS ARE TRAINED. THE TOP10 AND TOP20 CLASSIFIERS ARE TRAINED ON FEATURES 1–10 AND 1–20, RESPECTIVELY. *Bwd* INDICATES TRAFFIC FROM THE C&C SERVER DESTINED FOR AN INFECTED ENDHOST

ID	Feature name	Description
1	Protocol	The transport layer protocol
2	dstIntExt	Internal or external dstIP
3	Active Mean	Mean time a flow was active before becoming idle
4	Init Fwd Win Byts	Total number of bytes sent in the initial window in the forward direction
5	FIN Flag Cnt	Number of packets with FIN
6	Bwd Pkt Len Min	Minimum size of packet in the backward direction
7	Flow Pkt/s	Number of flow packets per second
8	Fwd IAT Max	Maximum time between two packets sent in the forward direction
9	Flow IAT Mean	Mean time between two packets sent in the flow
10	Subflow Fwd Pkts	The average number of packets in a subflow in the forward direction
11	Flow IAT Max	Maximum time between two packets sent in the flow
12	Fwd IAT Tot	Total time between two packets sent in the forward direction
13	Subflow Bwd Pkts	The average number of packets in a subflow in the backward direction
14	Subflow Fwd Byts	The average number of bytes in a sub flow in the forward direction
15	Bwd Header Len	Total bytes used for headers in the backward direction
16	Tot Bwd Pkts	Total packets in the backward direction
17	Fwd Pkt Len Std	Standard deviation size of packet in the forward direction
18	Fwd Seg Size Min	Minimum segment size observed in the forward direction
19	Bwd Pkt Len Std	Standard deviation size of packet in the backward direction
20	Bwd IAT Mean	Mean time between two packets sent in the backward direction

3. RELATED WORK

There is a large body of work applying machine learning algorithms to network intrusion detection. Existing approaches use algorithms such as support vector machines (SVM) [14], k-nearest neighbors [15], neural networks [16]–[18], or decision trees [1], [5], [19].

Several recent works have investigated the robustness of machine learning algorithms towards adversarial examples, which are targeted perturbations of an original sample that change the prediction of a model [7], [9], [10], [20]. Such adversarial attacks have shown that, generally, small targeted perturbations suffice to fool a model. Meanwhile, various adversarial defenses have been proposed that attempt to defend against adversarial examples by incorporating them during training [7], [8], [21].

In our work, we extend an existing adversarial attack [7] to the network intrusion detection setting and apply it to a tree-ensemble-based network flow classifier [1]. Further to that, we use adversarial training [7], [8] to improve the adversarial robustness of the flow classifier.

4. ATTACK FRAMEWORK

We now provide an overview of our attack framework and explain the relevant technical details. While our attack framework is designed for the network intrusion detection problem, the framework is generic and can also be applied to machine learning systems in other domains.

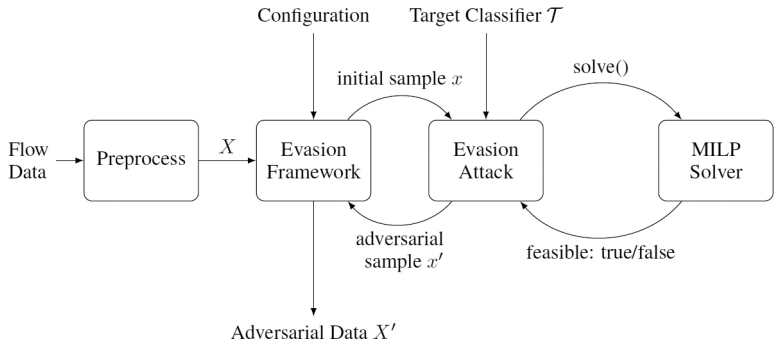
A. Overview

Our attack implements an automatic, end-to-end evasion framework, meaning that given an initial set of samples and a target tree-ensemble classifier, the system runs an optimal evasion attack for every sample and returns the adversarial samples. We designed our attack framework for the C&C channel evasion scenario. This scenario consists of a botmaster that tries to spread commands to infected endhosts inside a network that is protected by the target classifier. The botmaster's goal is to minimally adjust its network flows so that the classifier does not label them as malicious. In order to give strong security guarantees, we assume that the attacker has white-box access to the classifier. However, we assume that the attacker cannot modify the classifier's training.

Our attack framework is illustrated in Figure 3. The system first performs any necessary preprocessing on the raw flow data, which gives the input data X. The

evasion framework then receives both the input data X and a configuration that defines the feature space of X as well as feature constraints and costs. The evasion framework then runs the attack on X, receiving an adversarial sample for each input sample. The attack needs to first initialize by parsing the target classifier and translating its structure into a MILP problem. By constructing an evasion framework around the attack that passes a set of input samples X, we avoid having to repeatedly perform this translation, which makes the attack more efficient and, thus, more realistic for use in real network environments.

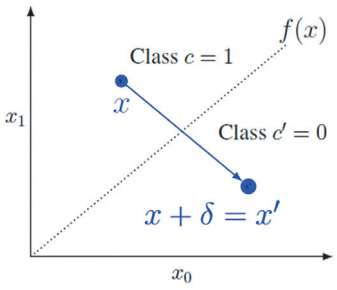
FIGURE 3: ATTACK FRAMEWORK



B. Attack

In abstract, the problem of evading a classifier can be described as follows: Given a classification algorithm $f: \mathcal{X} \to \mathcal{Y}$ and an input $x \in \mathcal{X}$ with $y = f(x)$, we want to find an adversarial input $x' \in \mathcal{X}$ that is assigned class $y' = f(x') \neq y$. Finding any input x' that is classified differently than the initial sample x is generally trivial and not particularly useful to the attacker. Therefore, what we want to find is an adversarial input $x' = x + \delta$ such that δ is minimized (cf. illustration in Figure 4).

FIGURE 4: EVASION OF AN INITIAL INPUT $x \in \mathcal{X}$ FOR A CLASSIFICATION ALGORITHM $f: \mathcal{X} \to \mathcal{Y}$



The attack we use in our work is based on the tree ensemble attack by Kantchelian et al. [7], which minimizes a loss function $\mathcal{L}: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$:

$$\min_{x' \in \mathcal{X}} \mathcal{L}(x, x') \text{ such that } f(x) \neq f(x')$$

While Kantchelian et al. propose to use any l^p-norm $\mathcal{L}^p(x, x') = \left(\sum_{i=1}^{d} |x_i - x_i'|^p\right)^{1/p}, \forall p \in \{1, 2, \ldots, \infty\}$ as their loss function, we generalize this loss function with a linear adversarial cost function [22]. This allows us to model the cost of adapting a network flow feature on a per-feature base by specifying a linear factor $a_j \in \mathbb{R}_+$ for each feature $j \in [0, d-1]$. Moreover, we are further able to capture immutable features by setting $a_j \rightarrow \infty$. Our new proposed loss function $\mathcal{L}$ for the adversarial attack is then given as:

$$\mathcal{L}^p(x, x') = \left(\sum_{j=0}^{d-1} a_j \left|x_j - x_j'\right|^p\right)^{1/p}, \forall p \in \mathbb{N}_+$$

and

$$\mathcal{L}^\infty(x, x') = max_j\left(a_j \left|x_j - x_j'\right|\right), j \in [0, d-1]$$

Feature Constraints: Finding an adversarial input x' according to the problem statement given above would require that every feature $x_j, j \in [0, d-1]$ could be modified independently; however, it is overly optimistic to assume that this can be done. In domains such as network intrusion detection, features typically have dependencies on each other. For instance, consider the *FIN Flag Cnt* and *Protocol* features of the target classifier. The *FIN Flag Cnt* feature can be 0 or 1 for TCP but has to be 0 for UDP since UDP does not have FIN flags. Thus, an adversarial sample that modifies *FIN Flag Cnt* from 0 to 1 for a UDP flow does not correctly model the given feature dependencies, and the botmaster will not be able to send a network flow that corresponds to the computed adversarial sample. Furthermore, feature constraints give the attacker more precise control over the adversarial flow since any custom constraints can be specified.

In order to model potentially complex feature constraints, we propose two types of constraints that can be imposed on the input features.

Simple constraints model any feature constraint of the form:

$$x_j' \circ t_j, j \in [0, d-1], t_j \in \mathbb{R}$$

Where $\circ$ is a binary operator $\circ \in \{<, >, \leq, \geq\}$, and t_j is any threshold on feature x_j'. This constraint forces feature x_j' of the evasion sample to take a value for which $x_j' \circ t_j$

evaluates to true. This allows us to model simple specifications on the evasion sample, such as constraining an evasion sample to have UDP as its protocol.

Dependent constraints model any implication feature constraint of the form:

$$x'_i \circ_i t_i \Rightarrow x'_j \circ_j t_j, i, j \in [0, d-1], t_i, t_j \in \mathbb{R}$$

Where $\circ_i$,$\circ_j$ are binary operators $\circ_i$,$\circ_j \in \{<, >, \leq, \geq\}$, and t_i, t_j are any thresholds on features x'_i, x'_j, respectively. This constraint enforces that if $x'_i \circ_i t_i$ evaluates to true for feature x'_i of the evasion sample x', then $x'_j \circ_j t_j$ must evaluate to true for feature x'_j of the evasion sample x'. This allows us to model inter-feature dependencies, such as enforcing that if the evasion sample has Protocol UDP, then *FIN Flag Cnt* has to be 0.

Furthermore, each constraint can either be global or per sample. Global constraints are applied once when initializing the attack framework. Per-sample constraints are applied and removed for every input sample, which allows unique constraints to be specified for every sample $x \in \mathcal{X}$.

C. Adversarial Training

In Section 5, we show that our attack on C&C classifiers is able to efficiently find adversarial samples, which can be used by a botmaster to send commands to infected endhosts without getting detected by the C&C classifier. Following these findings, we will show that the classifier's robustness can be improved by using state-of-the-art adversarial training methods. First, we use the adversarial boosting method proposed by Kantchelian et al. [7], which efficiently generates adversarial samples that can be incorporated when training the classifier. Further to that, we use robust decision tree training [8] in order to build a robust boosting tree classifier based on the model parameters of the initial classifier.

5. RESULTS

We evaluate our attack on four different versions of the target classifier, where the classifiers were trained on data from Locked Shields 2017 and 2018 (from here on referred to as *LS17* and *LS18*) and on top10 and top20, respectively. We instantiate the attack with the $\mathcal{L}^1$ loss and analyze for each adversarial input $x' = x + \delta$ the required adversarial perturbation $|\delta|_1$ and the susceptibility of each feature—i.e., how often a certain feature is adapted in order to evade the classifier. Further to that, we use the following fixed dependent feature constraints to model the feature dependency between the *Protocol* feature and the *FIN Flag Cnt and the Init Fwd Win Byts* features.

- $x_{Protocol} \geq 12 \Rightarrow x_{FINFlagCnt} < 0.5$: UDP does not have FIN flags
- $x_{Protocol} \geq 12 \Rightarrow x_{InitFwdWinByts} < 0$: UDP does not have an initial TCP window size. Flowmeter sets the Init Fwd Win Byts feature to -1 for UDP flows

Note that, due to these constraints, the optimal evasion attack rarely modifies the *Protocol* feature, even though we set the evasion cost of the *Protocol* feature to 1. Due to the feature dependencies of the *Protocol* feature, changing the protocol requires modifying the *Init Fwd Win Byts* feature, which results in large evasion distances.

Finally, all optimal evasion attacks were executed on a server with 10 Intel Xeon E5-2699 v3 cores at 2.30 GHz and 16 GB of RAM, and we used Gurobi [23] as a MILP solver.

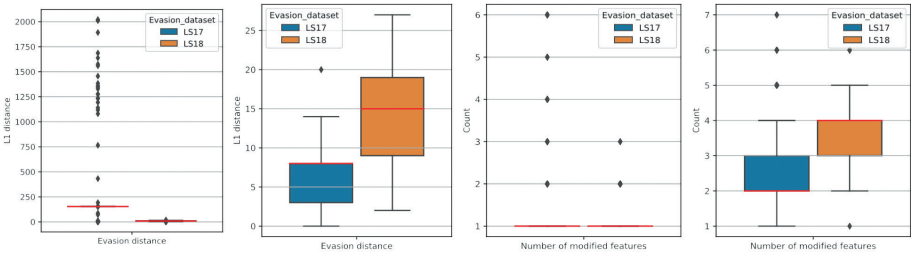
A. Flow Classifier Evasion

We first show the results for running our attack on all four considered classifier versions. For each evaluation, we run our attack on 1,000 samples and show the ℓ_1 evasion distances and the number of features that had to be modified. Figure 5b.1 shows that for the top10 classifiers, modifying one feature suffices to successfully evade the classifier in almost all cases. Table II shows that for the LS17 classifier, the most commonly modified feature is the *Init Fwd Win Byts* feature and that it must be decreased by 154 bytes on average. This is a minor modification, considering that the initial TCP window byte size lies in the region of [0.65535]. For the LS18 classifier, the most frequently modified feature is the *Subflow Fwd Pkts* feature, and we see that if this feature is modified, the number of subflow packets in the forward direction must be decreased by 9 packets on average. One could argue that sending fewer packets per flow is a more restrictive modification since the botmaster might need a minimum amount of information transmitted. However, this can be addressed by opening multiple flows and sending fewer packets per flow.

For the top20 classifiers, when we look at Table III and Figure 5b.2, we see similarly that a small subset of features is particularly susceptible. Compared to the top10 classifiers, the median number of modified features is larger for the top20 classifiers, namely 2 (LS17) and 4 (LS18)—this is as expected since the top20 classifiers use double the number of features. The most frequently modified features are *Tot Bwd Pkts, Subflow Bwd Pkts, Bwd Header Len*, and *Subflow Fwd Pkts*, which can all be modified by the adversary. Table III shows how much a given feature has to be modified in order to evade the classifier. For example, considering the two most commonly used evasion features, *Tot Bwd Pkts and Subflow Bwd Pkts*, we see that it suffices to send just 2–4 fewer packets to successfully evade the classifier.

We can thus conclude that an adversary can evade all flow classifiers by modifying a small subset of flow features. We further note that the most commonly modified features (*Init Fwd Win Byts, Subflow Fwd Pkts, Subflow Bwd Pkts, Tot Bwd Pkts, and Bwd Header Len*) can all be controlled by the adversary exactly.

FIGURE 5: EVASION DISTANCE AND THE NUMBER OF MODIFIED FEATURES WHEN RUNNING OUR ADVERSARIAL ATTACK ON 1,000 SAMPLES. IN A.1 AND B.1, THE RESULTS FOR LS17 ARE ON THE LEFT-HAND SIDE, AND THE RESULTS FOR LS18 ARE ON THE RIGHT-HAND SIDE



(a.1) top10 classifiers (a.2) top20 classifiers (b.1) top10 classifiers (b.2) top20 classifiers

TABLE II: PER-FEATURE EVASION DISTANCES OF THE TOP10 FLOW CLASSIFIERS. THE EVASION DISTANCE MEDIAN AND STANDARD DEVIATION ARE ONLY TAKEN ON NON-ZERO EVASION DISTANCES DUE TO THE SPARSITY OF THE DATA. FEATURES THAT WERE MODIFIED LESS THAN TEN TIMES COMBINED ARE NOT SHOWN

Feature	LS17			LS18		
	evasion dist. median	evasion dist. std	# of times modified	evasion dist. median	evasion dist. std	# of times modified
Subflow Fwd Pkts	-1.0	258.839	55	-9.0	2.982	958
Init Fwd Win Byts	-154.0	20.729	925	0	0	0
Fwd IAT Max	1.0	128.783	40	1.0	0.0	30
Flow Pkts/s	-57.683	854.448	27	-2.329	2.379	11
Flow IAT Mean	-414.159	239.196	18	-2.572	2.725	14
Bwd Pkt Len Min	10.0	3.374	23	7.0	2.070	8
Active Mean	8.0	6.397	16	0	0	0

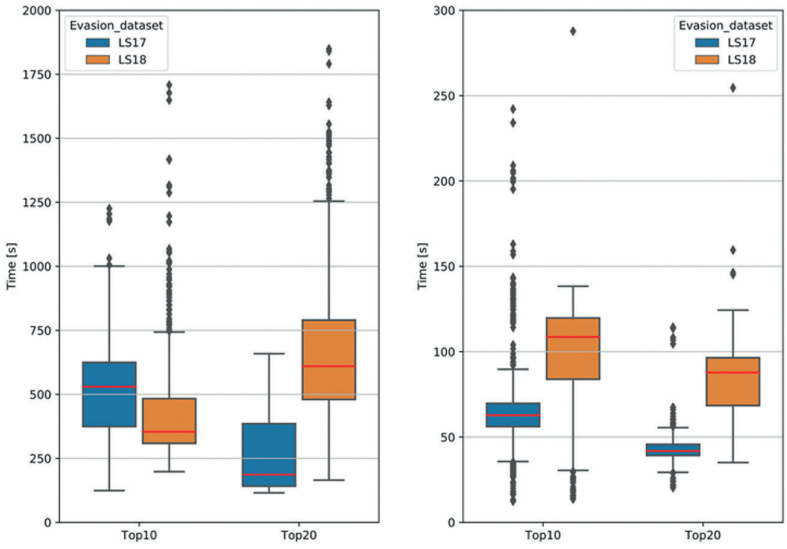
TABLE III: PER-FEATURE EVASION DISTANCES OF THE TOP20 FLOW CLASSIFIERS. THE EVASION DISTANCE MEDIAN AND STANDARD DEVIATION ARE TAKEN ONLY ON NON-ZERO EVASION DISTANCES DUE TO THE SPARSITY OF THE DATA. FEATURES THAT WERE MODIFIED LESS THAN 100 TIMES COMBINED ARE NOT SHOWN

Feature	LS17			LS18		
	evasion dist. median	evasion dist. std	# of times modified	evasion dist. median	evasion dist. std	# of times modified
Tot Bwd Pkts	-3.0	1.640	879	-4.0	0.991	951
Subflow Bwd Pkts	-3.0	1.640	877	-2.0	0.966	952
Bwd Header Len	1.0	1.299	118	-6.0	4.375	864
Subflow Fwd Pkts	1.0	0.877	210	-8.0	4.050	651
Fwd Seg Size Min	7.0	1.815	108	7.0	1.705	161
Bwd Pkt Len Std	0.049	0.807	108	-8.605	-	1

1) Evasion Time

MILP solvers can be computationally expensive, which is a deciding factor with respect to the practical relevance of our attack. For instance, considering the Locked Shields exercise, an optimal evasion attack that takes several hours or days is not practical since the exercise only takes place over a time span of four days. To demonstrate the practical relevance of our attack, we analyze the evasion time when using Gurobi [23] and PuLP [24]. Figure 6 shows the time to run the attack for a single sample. The median evasion time is between 3 and 10 minutes using PuLP and between 1 and 2 minutes using Gurobi. Thus, while Gurobi solves the optimal evasion problem significantly faster, both cases are clearly practical since evasion times in the range of several minutes do not pose a major limitation for an attacker.

FIGURE 6: PER-SAMPLE EVASION TIME USING PuLP AND GUROBI AS MILP SOLVER, 500 SAMPLES



(a) PuLP (b) Gurobi

2) Adversarial Transferability

Past work [20], [25] suggests that adversarial samples are transferable between different models, meaning that an adversarial input created for one classifier has a high likelihood of being adversarial for a different classifier too.

We evaluate adversarial transferability by checking whether adversarial samples for LS17 classifiers are also adversarial on the respective LS18 classifiers and vice-versa. As Table IV shows, adversarial samples from our attack are less transferable compared

to other attacks and learning algorithms, such as gradient descent attacks and neural networks. This result is expected since our attack is highly targeted, and the minimal evasive sample is set right at the decision boundary of the classifier. A slight variation in decision boundary can thus push an adversarial sample back into a malicious decision region, which in turn gives a true positive classification.

TABLE IV: TRANSFERABILITY OF ADVERSARIAL SAMPLES GENERATED BY THE OPTIMAL EVASION ATTACK

Classifier	**Recall** (%) on LS18 adversarial samples	**Classifier**	**Recall** (%) on LS17 adversarial samples
LS17 Top10	96.2	LS18 Top10	84.0
LS17 Top20	79.9	LS18 Top20	51.7

3) Comparing Optimal and Approximate Evasion

In addition to the optimal evasion attack, Kantchelian et al. also propose a faster approximate attack. In the following, we compare the adversarial inputs found by the approximate and the optimal attack. We run 1,000 evasions for every classifier version and limit the number of coordinated descent iterations to 30 in order to break exceedingly long coordinate descent searches. The results of these evaluations are shown in Table V. If we compare the evasion distances of adversarial inputs found by the approximate attack and the optimal attack, we can clearly see that the approximate attack finds adversarial inputs that require significantly larger modifications.

B. Adversarial Training

In this section, we use the adversarial training approaches discussed previously to train potentially more robust network flow classifiers, and we evaluate the resulting classifiers in terms of their classification performance and robustness against our attack.

1) Adversarial Boosting

We use adversarial boosting, as proposed by Kantchelian et al., to improve the robustness of the target classifiers. If we look at Table V, we see that the approximate attack finds adversarial samples with significantly larger perturbations and has a low success rate. Therefore, we decided not to use approximate attacks for adversarial boosting but instead to use our optimal attack to generate large sets of adversarial samples.

In our evaluation, we ran 10 concurrent evasion attacks on the top20 LS17 classifier over the course of 17 days, which is the computation time required by the MILP solver. This allowed us to generate 20,0152 adversarial samples, which corresponds to 16.1% of the number of malicious samples in the Locked Shields 2017 dataset. We

then trained the model proposed by Känzig et al. on a combined dataset consisting of the original Locked Shields 2017 samples and the generated adversarial samples. We first evaluated the classification performance of the resulting robust classifier on the Locked Shields 2018 dataset. Table VI shows the classification performance of the retrained classifier on the LS18 dataset compared to the original classifier. Compared to the original top20 LS17 classifier by Känzig et al., the robust classifier's classification performance is largely equivalent and even performs slightly better in malicious class recall.

We further evaluated the robustness of the retrained classifier using our optimal evasion attack. If we look at Figure 7a.1, we see that the median evasion distance for the adversarial retrained top20 LS17 classifier is 13.0. This corresponds to a 1.625× increase in robustness compared to the original top20 LS17 classifier, which has a median evasion distance of 8.0 (see Figure 5a.1). If we look Figure 7a.2, we further see that the median number of modified features for the retrained top20 LS17 classifier is equal to 2, which is equivalent to the original top20 LS17 classifier.

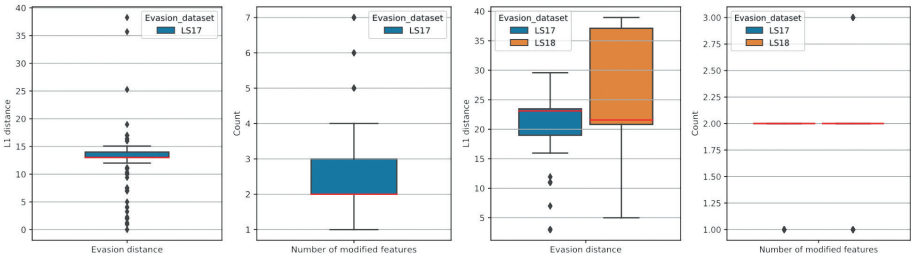
2) Robust Boosting Tree

We trained a gradient-boosting tree model using the robust tree training framework [8] for both the Locked Shields 2017 and 2018 datasets and for the top20 feature sets. Once again, we measured the robustness of each trained classifier by performing an optimal evasion attack for 1,000 samples and compared this to the robustness of the initial classifier.

TABLE V: APPROXIMATE EVASION EVALUATIONS FOR ALL CLASSIFIER VERSIONS. *EVASION DISTANCE INCREASE* REFERS TO THE INCREASE COMPARED TO THE OPTIMAL EVASION DISTANCE OF THE RESPECTIVE CLASSIFIERS

Classifier	ℓ_1 evasion distance median	evasion distance increase	# modified features median	evasion time median	evasion success rate (%)
LS17 Top10	35449.649	230.2×	2.0	3.164s	84.0
LS18 Top10	4260.424	473.4×	2.0	44.628s	15.2
LS17 Top20	2048.250	256.0×	5.0	13.429s	69.1
LS18 Top20	50410.264	3360.7×	6.0	15.390s	71.2

FIGURE 7: EVASION DISTANCE AND THE NUMBER OF MODIFIED FEATURES FOR THE ADVERSARIALLY TRAINED CLASSIFIERS. IN FIGURE 7B.2, THE RESULTS FOR LS17 ARE ON THE LEFT-HAND SIDE AND THE RESULTS FOR LS18 ARE ON THE RIGHT-HAND SIDE



(a.1) (a.2) (b.1) (b.2)

(a) Adversarial Boosting (b) Robust Boosting Tree

Table VII shows the classification performance of the robust top20 classifiers compared to the original top20 classifiers. We observe that the classification performance for the *benign* class decreases only slightly for both robust classifiers, except for the *benign* precision of the robust top20 LS18 classifier, which increases by 1.5%. For the robust LS17 classifier, *malicious* recall decreases by 42% and all averaged classification metrics decrease by roughly 4% compared to the original top20 LS17 classifier. The robust LS18 classifier performs 5.7% worse in *malicious* precision. However, it has an 11.3% increase in malicious recall, and thus an increase of around 1% for all averaged classification metrics, compared to the original top20 LS18 classifier.

We further compare the robustness of the robust classifiers to the original classifiers. Figure 7b.1 shows the evasion distance of the robust top20 classifiers. The robust top20 LS17 classifier has a median evasion distance of 23.1, which corresponds to a 2.89× increase in robustness compared to the original top20 LS17 classifier, which has a median evasion distance of 8.0. The robust top20 LS18 classifier has a median evasion distance of 21.6, which corresponds to a 1.44× increase in robustness compared to the original top20 LS18 classifier, which has a median evasion distance of 15.0. If we look at Figure 7b.2, we see that the number of modified features is 2 for the robust top20 LS17 classifier, which is equivalent to the original top20 LS17 classifier. The robust top20 LS18 classifier has only 2 modified features per evasion, compared to 4 for the original top20 LS18 classifier. Thus, while the evasion distance of the robust top20 LS18 classifier does increase, the decreased number of modified features per evasion can also be seen as a decrease in robustness.

TABLE VI: CLASSIFICATION PERFORMANCE OF THE ROBUST RETRAINED AND THE ORIGINAL KÄNZIG ET AL. TOP20 LS17 CLASSIFIER ON THE LS18 DATASET. *BENIGN, MALICIOUS* REFERS TO NON-C&C AND C&C FLOWS, RESPECTIVELY

Class	Precision (%)		Recall (%)		F1-score (%)	
	rob.	orig.	rob.	orig.	rob.	orig.
Benign	99.7	99.7	99.9	99.9	99.8	99.8
Malicious	98.7	98.7	97.6	97.3	98.1	98.0
Overall	99.6	99.6	99.6	99.6	99.6	99.6

TABLE VII: CLASSIFICATION PERFORMANCE OF THE ROBUST RETRAINED AND THE ORIGINAL KÄNZIG ET AL. TOP20 FLOW CLASSIFIERS. *BENIGN, MALICIOUS* REFERS TO NON-C&C AND C&C FLOWS, RESPECTIVELY

	LS17						**LS18**					
Class	Precision (%)		Recall (%)		F1-score (%)		Precision (%)		Recall (%)		F1-score (%)	
	rob.	orig.	rob.	orig.	rob.	orig.	rob.	orig.	rob.	orig.	rob.	orig.
Benign	95.3	99.7	99.7	99.9	97.4	99.8	99.4	97.9	98.9	99.8	99.2	98.8
Malicious	95.6	98.7	55.3	97.3	70.1	98.0	92.5	98.2	95.4	84.1	93.9	90.6
Overall	95.3	99.6	95.3	99.6	94.7	99.6	98.6	97.9	98.5	97.9	98.5	97.8

6. CONCLUSION

In this work, we evaluated the robustness of network intrusion detection classifiers by applying an attack to tree-ensemble-based classifiers. We presented an extended attack that builds on a previous attack by Kantchelian et al., and we designed an attack framework that handles end-to-end evasion of input samples. We showed that minor modifications to C&C flows suffice to successfully fool the target classifier and observed that most adversarial modifications can be implemented in practice. Furthermore, we showed that by using state-of-the-art robust training methods, we can increase the robustness of the target classifier.

However, there are limitations and extensions to consider for the future. First, while our work shows that minor modifications to the input features suffice to evade the target classifier, in practice, the attacker needs to be able to map the adversarial input to a network flow. Thus, the attacker needs to send traffic in such a way that the target system's traffic capture and preprocessing actually map the sent network flow to the intended adversarial input. While this is straightforward for features such as the TCP window size or the number of sent packets, it becomes more difficult for timing-based features such as the interarrival time. Thus, a natural extension to our work would be to implement a traffic modification system that leverages our attack framework to find adversarial inputs and then automatically modifies network traffic

to evade the target classifier in real time. Considering our results, the most commonly modified features can directly be modified on an endhost; thus, such a system could be directly implemented as a local system on the endhost or as a proxy. Another possibility would be to implement the system in-network using programmable data planes [26]. Second, we use an l^p-norm with linear costs as the loss function for our attack. However, this loss function does not necessarily reflect an attacker's effective effort to modify a network flow. Consider, for example, the *Init Fwd Win Byts* feature. In our loss function, modifying the *Init Fwd Win Byts* by 20 bytes incurs twice the loss of modifying it by 10 bytes. However, in practice, the attacker's effort for the two modifications is likely very similar. Thus, future work could explore different adversarial loss functions that better model an attacker's effective effort for modifying network traffic.

REFERENCES

[1] N. Känzig, R. Meier, L. Gambazzi, V. Lenders, and L. Vanbever, "Machine learning-based detection of C&C channels with a focus on the Locked Shields cyber defense exercise," in *2019 11th International Conference on Cyber Conflict (CyCon)*, vol. 900, IEEE, 2019, pp. 1–19.

[2] B. Abraham, A. Mandya, R. Bapat, F. Alali, D. E. Brown, and M. Veeraraghavan, "A comparison of machine learning approaches to detect botnet traffic," in *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2018, pp. 1–8.

[3] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 239–248.

[4] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 219–230, 2004.

[5] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C* (*Applications and Reviews*), vol. 38, no. 5, pp. 649–659, 2008.

[6] L. Gehri, R. Meier, D. Hulliger, and V. Lenders, "Towards generalizing machine learning models to detect command and control attack traffic," in *2023 15th International Conference on Cyber Conflict: Meeting Reality (CyCon)*, IEEE, 2023, pp. 253–271.

[7] A. Kantchelian, J. D. Tygar, and A. Joseph, "Evasion and hardening of tree ensemble classifiers," in *International Conference on Machine Learning*, 2016, pp. 2387–2396.

[8] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, "Robust decision trees against adversarial examples," 2019, *arXiv:1902.10660*.

[9] B. Biggio et al., "Evasion attacks against machine learning at test time," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 387–402.

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*.

[11] P. Laskov et al., "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE Symposium on Security and Privacy*, IEEE, 2014, pp. 197–211.

[12] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 255–264.

[13] "Locked shields." CCDCOE. 2020. [Online]. Available: https://ccdcoe.org/exercises/locked-shields/

[14] M. S. Pervez and D. M. Farid, "Feature selection and intrusion classification in NSL-LKDD cup 99 dataset employing SVMs," in *8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014)*, IEEE, 2014, pp. 1–6.

[15] H. Shapoorifard and P. Shamsinejad, "Intrusion detection using a novel hybrid method incorporating an improved KNN," *International Journal of Computer Applications*, vol. 173, no. 1, pp. 5–9, 2017.

[16] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[17] R. Vinayakumar, K. Soman, and P. Poornachandran, “Applying convolutional neural network for network intrusion detection,” in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2017, pp. 1222–1228.
[18] G. Zhao, C. Zhang, and L. Zheng, “Intrusion detection using deep belief network and probabilistic neural network,” in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 1, IEEE, 2017, pp. 639–642.
[19] B. Ingre, A. Yadav, and A. K. Soni, “Decision tree based intrusion detection system for NSL-KDD dataset,” in *Information and Communication Technology for Intelligent Systems (ICTIS 2017)*, vol. 2, no. 2, Springer, 2018, pp. 207–218.
[20] C. Szegedy et al., “Intriguing properties of neural networks,” 2013, *arXiv:1312.6199*.
[21] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” 2017, *arXiv:1706.06083*.
[22] D. Lowd and C. Meek, “Adversarial learning,” in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005, pp. 641–647.
[23] “Gurobi optimizer reference manual,” Gurobi Optimization LLC, 2020. [Online]. Available: http://www.gurobi.com
[24] S. Mitchell, M. OSullivan, and I. Dunning, “PuLP: A linear programming toolkit for Python,” University of Auckland, Auckland, New Zealand, 2011.
[25] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1765–1773.
[26] P. Bosshart et al., “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.