NORTHEASTERN UNIVERSITY

DOCTORAL THESIS

On the Robustness of Machine Learning Training in Security Sensitive Environments

---

Author:
Giorgio Severi

Supervisor:
Alina Oprea

Committee:
Alina Oprea
Cristina Nita-Rotaru
William Robertson
Scott Coull

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Khoury College of Computer Sciences

Northeastern University
Khoury College of
Computer Sciences

PhD Thesis Approval

Thesis Title: On the Robustness of Machine Learning Training in Security Sensitive Environments

Author: Giorgio Severi

PhD Program: X Computer Science ____ Cybersecurity ____ Personal Health Informatics

PhD Thesis Approval to complete all degree requirements for the above PhD program.

Signed by: Alina Oprea 8E1DB58923CF40A...	7/19/2024
Thesis Advisor	*Date*
Signed by: Cristina Nita-Rotaru 82FA470B076E498...	7/19/2024
Thesis Reader	*Date*
Signed by: William Robertson C2BCD6A7B1214AB...	7/19/2024
Thesis Reader	*Date*
Signed by: Scott Coull E8B0FC3D3880477...	7/19/2024
Thesis Reader	*Date*
Thesis Reader	*Date*

KHOURY COLLEGE APPROVAL:

Associate Dean for Graduate Programs	*Date*

COPY RECEIVED BY GRADUATE STUDENT SERVICES:

Pete Morency	7/24/2024
Recipient's Signature	*Date*

Distribution: Once completed, this form should be attached as page 2, immediately following the title page of the dissertation document. An electronic version of the document can then be uploaded to the Northeastern University-UMI Website.

Abstract

Modern machine learning underpins a large variety of commercial software products, including many cybersecurity solutions. Widely different models, from large transformers trained for auto-regressive natural language modeling to gradient boosting forests designed to recognize malicious software, all share a common element: they are trained on an ever increasing quantity of data to achieve impressive performance levels in their tasks. Consequently, the training phase of modern machine learning systems holds dual significance: it is pivotal in achieving the expected high-performance levels of these models, and concurrently, it presents a prime attack surface for adversaries striving to manipulate the behavior of the final trained system. This dissertation explores the complexities and hidden dangers of training supervised machine learning models in an adversarial setting, with a particular focus on models designed for cybersecurity tasks.

Guided by the belief that an accurate understanding of the offensive capabilities of the adversary is the cornerstone on which to found any successful defensive strategy, the bulk of this thesis is composed by the introduction of novel training-time attacks. We start by proposing training-time attack strategies that operate in a clean-label regime, requiring minimal adversarial control over the training process, allowing the attacker to subvert the victim model's prediction through simple poisoned data dissemination. Leveraging the characteristics of the data domain and model explanation techniques, we craft training data perturbations that stealthily subvert malicious software classifiers. We then shift the focus of our analysis on the long-standing problem of network flow traffic classification. In this context we develop new poisoning strategies that work around the constraints of the data domain through different strategies, including generative modeling. Finally, we examine unusual attack vectors, when the adversary is capable of tampering with different elements of the training process, such as the network connections during a federated learning protocol. We show that such an attacker can induce targeted performance degradation through strategic network interference, while maintaining stable the performance of the victim model on other data instances. We conclude by investigating mitigation techniques designed to target these insidious clean-label backdoor attacks in the cybersecurity domain.

Acknowledgements

This dissertation would not have been possible without the support and guidance of many great researchers.

I am deeply grateful to my advisor, Alina Oprea, for her invaluable mentorship, feedback, and encouragement throughout this journey. I extend my sincere gratitude to the members of my dissertation committee: Cristina Nita-Rotaru, William Robertson, and Scott Coull. Their thoughtful comments have significantly enriched this research.

I would also like to thank all of my collaborators both within the NDS2 Lab, and beyond. Their expertise and diligent efforts were crucial in realizing this work. Without their valuable contributions, insights, and support, this research would not have achieved its current depth and breadth.

Finally, I want to sincerely thank my family and all the friends I have made in these six years at ISEC and 177. Our daily interactions were instrumental in helping me persevere through challenging times.

Contents

List of Figures

List of Tables

List of Abbreviations

AS	Autonomous System
AV	Anti Virus
CV	Computer Vision
FL	Federated Learning
FPR	False Positive Rate
FNR	False Negative Rate
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
PCA	Principal Component Analysis
PDF	Portable Document Format
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

The past decade has witnessed a tremendous growth in the research, development, and practical application of machine learning (ML) models across an astonishing array of domains. The breadth and depth of capabilities demonstrated by contemporary ML systems are nothing short of surprising: from vision transformers obtaining excellent results in image classification [61], to gradient boosting trees employed for malware identification at low false positive rates [5, 83], from diffusion models generating amazing digital art [177] to massive auto-regressive natural language models producing coherent prose and executable code [162] — the list is long and ever-growing. Inevitably, this trend has extended to a variety of cybersecurity applications, where ML models are used to quickly and accurately perform critical tasks such as malware execution prevention and network monitoring.

Most of these models, and the systems they underpin, share a common characteristic: their performance metrics tends to improve significantly with larger volumes of training data and computation employed during training [88, 107]. This realization led model developers to acquire increasingly large volumes of raw data from potentially untrusted sources, such as Internet scrapes and crowd-sourcing platforms. Not only data, but computation too is often outsourced completely, or performed on hardware (potentially with its own software stack) that is under control of third parties.

Outsourcing computation and data, however, opens the door to adversaries, who may wish to subvert the correct behavior of the learned models for economic or criminal purposes[1]. Such an adversary can, in fact, exploit the access to the training process granted by the reliance on untrusted third parties, and inject a desired behavior into the learned model. Literature on the subject [164, 48] generally refers to this typology

[1] In this thesis, we will focus on scenarios where the "adversary" is motivated by criminal or otherwise malicious intents. Therefore we use the terms *adversary*, *attacker*, and *malicious actor* interchangeably. However, this may not be the case in general, as there may be uses of adversarial machine learning for activism, political disobedience, or even to protect users privacy. A rigorous discussion of this topic is a complex endeavor, and out of the scope of this work. We refer the reader to the work of Albert et al. [1] for an introduction on the subject.

of adversarial interference as *poisoning attacks*. Poisoning attacks can be orchestrated to achieve a variety of adversarial objectives, such as inducing misclassificationevents on specific data instances, or even improve the effectiveness of privacy attacks.

As computational tasks are increasingly delegated and datasets expand in scale, the potential risk escalates significantly for all ML applications. However, we argue that this issue is even worse when we consider the security domain. In this context, not only there is a clear, natural, incentive for malicious actors to attempt to corrupt the learned model, but sanitizing the training becomes much more difficult and expensive. As the reader can imagine, the task of verifying the correctness of large volumes of images or textual data – potentially through labor crowd-sourcing platforms – tends to be significantly less expensive, than that of vetting training datasets composed of executable binaries, which requires expert analysts.

The training phase, therefore, constitutes a critical juncture in the overall pipeline of modern machine learning: it is quintessential to achieve the best performance on the desired tasks, while at the same time representing a deeply sensitive point that a malicious actor may wish to exploit. Defending the training phase of machine learning models from poisoning attacks is thus imperative if we wish to rely on these systems for security and safety sensitive applications.

1.1 Thesis Contributions

With this thesis we wish to investigate the effectiveness of training-time attacks, their applicability to realistic pipelines, and potential mitigation strategies. We are motivated by the perspective that understanding offensive capabilities is crucial for developing robust defenses. Our belief is guided by seminal works in software [194, 63] and hardware [128, 113] security which shaped the defensive mechanisms currently employed in modern computation systems. Our final goal is to advance an evidence-based approach for hardening machine learning training, paralleling the evolution of defenses in traditional computer security.

To realize this aim, this thesis puts forth the following principal contributions:

- **Explanation-guided poisoning:** We design a new type of backdoor poisoning attacks centered around the use of AI model explanation techniques (XAI). These attacks allow the adversary to associate a pre-defined trigger pattern to a target class, which can be used to control the classifier's output. We use XAI to guide the generation of the trigger pattern to obtain effective attacks with minimal side effects on the normal behavior of the victim model. We demonstrate these attacks in diverse security-critical settings, including malware detection [193] and network

intrusion detection [190], by tailoring the trigger generation process for different modalities like binaries and network flows.

- **Backdoor attacks with limited adversarial control:** We demonstrate how effective backdoor attacks can be carried out in extremely constrained environments, mirroring real world deployment conditions. The attacks we propose do not require any control over the training labels, make no assumptions on the victim model's architecture, and respect the semantic constrains of complex data modalities such as executable binaries [193] and aggregated network flows [190]. We smuggle stealthy triggers by adversarially perturbing only a small fraction of benign samples to convey our backdoor trigger and we ensure that the altered samples blend in with the natural distribution.
- **Mitigation strategies for backdoor attacks in cybersecurity:** We propose new techniques that leverage the insights in cybersecurity threat models gained from the research presented in this thesis to mitigate clean-label backdoor attacks, while preserving the model utility. Our method [189] performs density-based clustering on a carefully chosen feature subspace, and progressively isolates suspicious clusters through a novel iterative scoring procedure. With this approach, our defensive mechanism can mitigate the attacks without requiring many of common assumptions in the backdoor defense literature which would be difficult to realize in the security domain.
- **Targeted network-level interference:** We show how to launch network-level attacks against Federated Learning protocols that achieve the same effect of targeted poisoning. To obtain these results, we propose a methodology to strategically interfere with the exchange of selected model updates [191]. Simultaneously, we also present a mitigation approach geared towards this setup, which helps preventing targeted disruptions against vulnerable data populations.

Collectively, these results showcase the sophistication of possible attacks against modern machine learning systems, while also pointing towards promising directions for increasing training robustness. Hardening machine learning pipelines remains an open research direction with profound importance for the safe and ethical deployment of artificial intelligence systems, especially in security and safety-critical domains.

1.2 Thesis Organization

This thesis will start by presenting some relevant background information in Chapter 2, which provide the reader with the necessary context to delve into the techniques proposed in the subsequent chapters. In Chapter 3 we introduce explanation-guided poisoning attacks against malicious software detectors, showing how to leverage model interpretation tools to launch stealthy backdoor attacks. This chapter is followed by an in depth exploration of the applicability of backdoor attacks against network traffic classifiers operating on aggregated flows, in Chapter 4. After presenting our novel backdoor attacks, Chapter 5 presents possible approaches to mitigate this threat, based on iterative re-training and cluster analysis. Chapter 6 then shows how different types of training-time interference operations can be carried out by the adversary to achieve results similar to targeted poisoning in decentralized learning systems. We conclude in Chapter 7 with a summary of the contributions of this thesis, and an overview of possible directions for future research.

Chapter 2

Background

In this chapter, we will cover some relevant background information that will help contextualize the work presented in this thesis. Starting with some notes on the use of machine learning models in cybersecurity applications, we will then provide an introduction to adversarial machine learning, and conclude with a quick overview on model interpretation methods.

2.1 Machine Learning for Cybersecurity Applications

2.1.1 Malware Detection Systems

We can roughly distinguish the numerous solutions to the problem of automated malicious software detection into two main classes based on their use of *static* or *dynamic* analysis. Dynamic analysis systems execute binary files in a virtualized environment, and record the behavior of the sample looking for indicators of malicious activities [137, 216, 4, 110, 192]. Meanwhile, static analyzers process executable files without running them, extracting the features used for classification directly from the binary and its metadata. While both approaches have positive and negative aspects, many endpoint security solutions tend to implement static analyzers due to the strict time constraints under which they usually operate.

With the shift towards ML based classifiers, this second class can be further divided into two additional subcategories: handcrafted feature-based detectors [186, 138, 196, 188, 5], and raw-binary analyzers [175, 47, 117]. We generally focus our attention, and the attacks we develop in Chapter 3, on classifiers based on static features due to their prevalence in providing pre-execution detection and prevention for many commercial endpoint protection solutions [58, 54, 136].

2.1.2 Network Threats Detection

Machine learning methods have been successfully used to detect several cyber security threats related to network traffic, including: malicious domains [8, 176, 9, 163, 159], command-and-control communication between attackers and compromised hosts [151, 163], or malicious binaries used by adversaries for distributing exploit code and botnet commands [98, 217]. Several endpoint protection products [144, 96, 97] are now integrating ML tools to proactively detect the rapidly increasing number of threats.

2.2 Adversarial Machine Learning

Adversarial Machine Learning is an extremely active research area, composed by a number of sub-fields touching on different aspects of machine learning, from privacy to integrity and from safety to fairness. For a detailed taxonomy of adversarial ML techniques and objectives we refer the reader to a recent standardization effort by Oprea and Vassilev [164].

This thesis will focus almost exclusively on the security aspects of adversarial ML pertaining to integrity violations. Which means that the adversaries we consider have as primary goal the ability to *take control of the output* of the victim model. While there are multiple potential reasons why an adversary would want to gain control over a model's output, in the context of cybersecurity applications one of the most common motivation is to ensure that their malicious activities (software execution, network traffic, file system activity, etc...) are not identified as suspicious by the targeted models.

Integrity attacks against ML models are generally grouped into two major categories based on the stage of the ML pipeline at which they occur: *inference-time* attacks and *training-time* attacks.

2.2.1 Inference-time Attacks

The first category is also commonly referred to as *test-time* or *evasion* attacks [215] (or more commonly and less precisely *adversarial examples*). Generally evasion attacks are aimed towards classifiers learned through supervised or semi-supervised training, and their objective is to induce a mis-classification of the selected data point [21, 215] either towards a randomly chosen class (*untargeted*) or towards a specific class chosen by the adversary (*targeted*).

This class of attacks proceeds by altering a testing sample by adding a small perturbation so that it is misclassified by the victim model. Here, the key term is "small". Using large perturbations that semantically impact the nature of the test point, e.g. changing a dog's snout to that of a cat in an image classification task, would render any attack

trivial. Instead, the adversarial perturbations are designed to be either imperceptible to human observers, in the case of vision, NLP, and audio classification tasks, or difficult to identify with automated outlier detection tools for cybersecurity applications.

Evasion attacks have been extensively explored in the context of computer vision, natural language processing and, to a lesser extent, audio processing. They are the most well known typology of attacks against ML, and likely the first line of attack against modern models. Due to the sheer volume of research in this area, it would be impractical to provide here a detailed breakdown of all the studies on the subject, thus we point the interested reader towards some resources that list the most relevant works in the field [32, 39, 164].

Due to their immediate relevance in adversarial contexts, in the security space, previous research efforts have investigated the applicability of such techniques to malware classification [21, 75, 249, 115, 209] and network traffic classification [82, 31, 15, 166, 45].

2.2.2 Training-time Attacks

More recently, however, poisoning attacks against ML, which manipulate the training set (or other training parameters), have emerged as a top concern in industry [202]. This category, the main focus of this thesis, is itself a collection of different adversarial processes against a victim learner aimed at different objectives.

The earliest approaches to training-time attacks consisted in introducing adversarially modified data points in the victim's training set [22] —from which derives the name *poisoning* attacks—, and were generally targeted towards increasing the target model's error rate on the test set (often referred to as an *availability* objective [102]). Over time, however, researchers developed different types of poisoning attacks aimed at a variety of objectives.

While availability attacks are extremely impactful, they are also obvious to detect, and their footprint can be quite large. To address these shortcomings, *targeted* poisoning attacks were designed to induce a degradation in the performance of the victim classifier only on specific points [210, 195], or entire data sub-populations [103], pre-determined by the adversary. The work presented in Chapter 6 presents a technique to achieve similar effects to targeted poisoning in a federated learning setup through network-level adversarial interference.

Poisoning-augmented *privacy* attacks [220, 41] are another important training-time threat against ML models. While not directly aimed at violating the victim's model integrity properties, these adversarial processes leverage integrity attacks during training to facilitate privacy attacks such as membership inference [35] at test time.

Backdoor attacks, introduced in the context of modern ML models by Gu et al. [78], pursue a similar yet distinct goal: triggering a mis-classification towards a target class whenever a specific pattern is inserted in the test sample without damaging the accuracy of the victim model on normal points.

Backdoor Attacks

In most of the work presented in Chapter 3 and Chapter 4 we focus on backdoor poisoning attacks, a notably dangerous technique that does not affect the model's predictions on clean test data. This behavior is obtained by introducing a trigger pattern in the training data, and swapping the correct label for the target one so that the victim model learns the association between pattern and target class. In this context, a backdoor trigger is a specific combination of features and selected values[1] that the victim model is induced, during training, to associate with a target class. Once the model relies on the backdoor pattern to distinguish the target, the adversary can inject the same watermark in any data point to arbitrarily trigger the desired outcome during inference. In their seminal work, Gu et al. [78] used a small pixel-based trigger pattern to misclassify images of handwritten digit and street sign classifiers.

The same objective as backdoor poisoning, was also achieved by Liu et al. [132], through direct alteration of the model weights (*model poisoning*, also often referred to as *trojaning*[2]).

Successive work by Turner et al. [223, 222] and Shafahi et al. [195] improved on these attacks by proposing variants of both targeted and backdoor poisoning in computer vision tasks that did not require any adversarial control of the training labels. These variants are commonly known as *clean-label* poisoning attacks.

In the cybersecurity domain, the literature on backdoor attacks is more scarce, with only a few methods designed via packet-level poisoning [92, 156], feature-space poisoning [11, 124], and label flipping [166]. Chapter 3 and Chapter 4 expand the knowledge in this area by introducing backdoor attacks guided by insights gathered through model interpretability tools, and applied to malware classification and aggregated network traffic analysis.

[1] In the computer vision domain this usually corresponds to a selected pixel mask with associated pixel values. [78], for instance, used a fixed trigger corresponding to a small yellow square, similar to a post-it note. In a more general sense, any combination of features and values can act as a trigger pattern.

[2] Due to their common objective, the adversarial machine learning literature often uses the two terms, backdoor and trojan, interchangeably. Given the difference in the methodology used by the authors of the original papers, we will keep the two terms distinct.

2.3 Interpretable Machine Learning

Recent years have seen a remarkable expansion of the use cases of Machine Learning systems in a variety of contexts. The progressive adoption of larger and more complex models, however, is often seen as problematic due to the difficulty for human overseers to understand the causes of model predictions. Thus, the need to provide human-interpretable explanations of model behaviors led to a surge of research in the area of Interpretable ML (also referred to as explainable AI, or XAI).

A large variety of interpretation methods have been proposed that can be categorized according to different criteria. A comprehensive examination of the subject can be found in [147]. In the work presented here, the focus will be on post-hoc, model agnostic, local attribution methods: techniques that explain a single prediction of any trained classifier by assigning weights to each feature based on their contribution to the final outcome. The rationale for this approach stems from how the explanation values would be leveraged by an attacker: as sources of insight into which features drive particular classification results, and how to perturb them to accomplish the adversarial objective.

One of the most well-known local model-agnostic explanation methods is SHAP (SHapley Additive exPlanations) by Lundberg et al. [133, 134], based on the cooperative game theory concept of Shapley values[3]. Its main objective is to explain the output logits of a model on a given test point by assigning a value to each feature composing the data point that reflects the contribution of the feature to the final output.

To this end, the framework trains a surrogate linear explanation model of the form:

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j \tag{2.1}$$

Where x'_j is the j^{th} feature of transformed sample x', and ϕ_j is its contribution to the model's decision. SHAP values satisfy three key properties: (i) *local accuracy*, which requires the explanation model g to match the output of the target f model on the input $f(x) = g(x')$; (ii) *missingness*, requiring missing features in the original input to have no attributed inpact; (iii) *consistency*, which states that if the target model changes so that the marginal contribution of a feature value increases or stays equal, then the relative SHAP value will also increase or stay equal.

Importantly, the SHAP framework requires only query access to the target classifier, which is of particular importance in an adversarial setting. Moreover SHAP has been

[3]The objective of Shapley values is to fairly assign payoffs to participants in repeated cooperative games. In the context of ML, the features representing each data point act as the participants of the game, and the outcome of the game corresponds to the output of the classifier. https://en.wikipedia.org/wiki/Shapley_value

shown to improve over earlier model interpretation approaches, including LIME [180] Integrated Gradients [213], DeepLIFT [201] and Layer-Wise Relevance Propagation [24]. Linardatos et al. [127] provide a comprehensive taxonomy of these methods, and conclude that, among the black-box techniques explored, SHAP is the most complete, providing explanations for any model and data type both at a global and local scale.

In our studies, we also experiment with estimating feature importance through Gini index [71] and information gain [114, 120] – two of the most popular splitting algorithms in decision trees. A decision tree is built recursively, by choosing at each step the feature that provides the best split. Thus, the tree offers a natural interpretability, and a straightforward way to compute the importance of each feature towards the model's predictions.

Chapter 3

Poisoning Static Malware Classification

Malware classification is a typical security scenario where the objectives of the parties involved induce a clear incentive for poisoning attacks. Any successful malware campaign, in fact, relies on the malicious software not being identified before it can accomplish its mission. Thus, the ability to arbitrarily alter the decision of a classifier represents a great asset in the adversary's arsenal. In this chapter we will explore new strategies, introduced in [193], to perform backdoor poisoning attacks on malware classifiers with limited access to the training process.

Chapter Summary

Training pipelines for machine learning (ML) based malware classification often rely on crowdsourced threat feeds, exposing a natural attack injection point. In this work, we study the susceptibility of feature-based ML malware classifiers to backdoor poisoning attacks, specifically focusing on challenging "clean label" attacks where attackers do not control the sample labeling process. We propose the use of techniques from explainable machine learning to guide the selection of relevant features and values to create effective backdoor triggers in a model-agnostic fashion. Using multiple reference datasets for malware classification, including Windows PE files, PDFs, and Android applications, we demonstrate effective attacks against a diverse set of machine learning models and evaluate the effect of various constraints imposed on the attacker. To demonstrate the feasibility of our backdoor attacks in practice, we create a watermarking utility for Windows PE files that preserves the binary's functionality, and we leverage similar behavior-preserving alteration methodologies for Android and PDF files. Finally, we experiment with potential defensive strategies and show the difficulties of completely defending against these attacks, especially when the attacks blend in with the legitimate sample distribution.

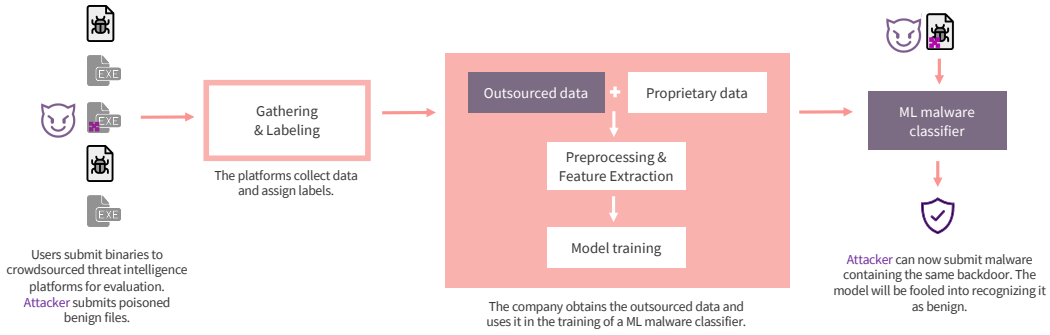


FIGURE 3.1: Overview of the attack on the training pipeline for ML-based malware classifiers.

3.1 Problem Definition

The endpoint security industry has increasingly adopted machine learning (ML) based tools as integral components of their defense-in-depth strategies. In particular, classifiers using features derived from static analysis of binaries are commonly used to perform fast, pre-execution detection and prevention on the endpoint, and often act as the first line of defense for end users [58, 54, 136]. Concurrently, we are witnessing a corresponding increase in the attention dedicated to adversarial attacks against malicious software (malware) detection models. The primary focus in this area has been the development of *evasion* attacks [215, 73, 21], where the adversary's goal is to alter the data point at inference time in order to induce a misclassification.

However, in this work, we focus on the insidious problem of *poisoning* attacks [22], which attempt to influence the ML training process, and in particular *backdoor* [78] poisoning attacks, where the adversary places a carefully chosen pattern into the feature space such that the victim model learns to associate its presence with a class of the attacker's choice. While evasion attacks have previously been demonstrated against both open-source [135] and commercial malware classifiers [203], backdoor poisoning offers attackers an attractive alternative that requires more computational effort at the outset, but which can result in a generic evasion capability for a variety of malware samples and target classifiers. These backdoor attacks have been shown to be extremely effective when applied to computer vision models [44, 132] without requiring a large number of poisoned examples, but their applicability to the malware classification domain, and feature-based models in general, has not yet been investigated.

Poisoning attacks are a danger in any situation where a possibly malicious third party has the ability to tamper with a subset of the training data. For this reason, they have come to be considered as one of the most relevant threats to production deployed ML models [202]. We argue that the current training pipeline of many security vendors provides a natural injection point for such attacks. Security companies, in fact, often rely

on crowd-sourced threat feeds [2, 140, 228, 229] to provide them with a large, diverse stream of user-submitted binaries to train their classifiers. This is chiefly due to the sheer quantity of labeled binaries needed to achieve satisfactory detection performance (tens to hundreds of millions of samples), and specifically the difficulty in adequately covering the diverse set of goodware observed in practice (e.g., custom binaries, multiple versions of popular software, software compiled with different compilers, etc.).

One complication in this scenario, however, is that the labels for these crowd-sourced samples are often generated by applying several independent malware detection engines [106], which would be impossible for an attacker to control. Therefore, in this work, we study *clean-label* backdoor attacks [195, 223] against ML-based malware classifiers by developing a new, model-agnostic backdoor[1] methodology. Our attack injects backdoored benign samples in the training set of a malware detector, with the goal of changing the prediction of malicious software samples watermarked with the same pattern at inference time.

To decouple the attack strategy from the specifics of the ML model, our main insight is to leverage tools from ML explainability, namely SHapley Additive exPlanations (SHAP) [133], to select a small set of highly effective features and their values for creating the watermark. We evaluate our attack against a variety of machine learning models trained on widely-used malware datasets, including EMBER (Windows executables) [5], Contagio (PDFs) [204], and Drebin (Android executables) [12]. Additionally, we explore the impact of various real-world constraints on the adversary's success, and the viability of defensive mechanisms to detect the attack.

Overall, our results show that the attack achieves high success rates across a number of scenarios and that it can be difficult to detect due to the natural diversity present in the goodware samples. The contributions of this work can be summarized as follows:

(i) We highlight a natural attack point which, if left unguarded, may be used to compromise the training of commercial, feature-based malware classifiers.

(ii) We propose the first general, model-agnostic methodology for generating backdoors for feature-based classifiers using explainable machine learning techniques.

(iii) We demonstrate that explanation-guided backdoor attacks are feasible in practice by developing a backdooring utility for Windows PE files, and using similar functionality-preserving methods for Android and PDF files. We show that these methods can satisfy multiple, realistic adversarial constraints.

[1] We will refer to the combination of features and values used to induce the misclassification, as trigger, watermark, or simply backdoor.

TABLE 3.1: Summary of attacker scenarios. Fullness of the circle indicates relative level of knowledge or control.

Attacker	Knowledge				Control	
	Feature Set	Model Architecture	Model Parameters	Training Data	Features	Labels
unrestricted	●	●	●	●	●	○
data_limited	●	●	●	◔	●	○
transfer	●	○	○	●	●	○
black_box	●	○	○	●	●	○
constrained	●	●	●	●	◔	○

(iv) Finally, we evaluate mitigation techniques and demonstrate the challenges of fully defending against stealthy poisoning attacks.

3.2 Threat Model

A typical training pipeline for a ML-based malware classifier, summarized in Figure 3.1, commonly starts with the acquisition of large volumes of labeled binaries from third-party threat intelligence platforms. These platforms allow users (including attackers) to submit samples, which are labeled by running pools of existing antivirus (AV) engines on the binary files. Companies can then acquire the labeled data from the platforms. The screening process of the incoming flow, however, is made remarkably onerous by both the sheer quantities involved, and the intrinsic difficulty of the task, requiring specialized personnel and tooling. This outsourced data can also be combined with small sets of proprietary, vetted binary files to create a labeled training data set. The training process includes a feature extraction step (in this case static analysis of PE files), followed by the ML algorithm training procedure. The trained malware classifiers are then deployed in the wild, and applied to new binary files to generate a label, malicious (malware) or benign (goodware).

Threat intelligence data comes with a set of labels determined by third-party AV analyzers, that are not under direct control of the attacker. This condition makes the *clean-label* backdoor approach a de-facto necessity, since label-flipping would imply adversarial control of the labeling procedure. The adversary's goal is thus to generate backdoored benign binaries, which will be disseminated through these labeling platforms, and will poison the training sets for downstream malware classifiers. Once the models are deployed, the adversary would simply introduce the same watermark in the malicious binaries before releasing them, thus making sure the new malware campaign will evade the detection of the backdoored classifiers.

3.2.1 Adversary's Goals and Capabilities

A large fraction of the backdoor attack literature adopts the BadNets threat model [78], which defined: (i) an "Outsourced Training Attack", where the adversary has full control over the training procedure, and the end user is only allowed to check the training using a held-out validation dataset; and (ii) a "Transfer Learning Attack", in which the user downloads a pre-trained model and fine-tunes it. We argue that, in the context we are examining, this threat model is difficult to apply directly. Security companies are generally risk-averse and prefer to either perform the training in-house, or outsource the hardware while maintaining full control over the software stack used during training. Similarly, we do not believe the threat model from Liu et al. [132], where the attacker partially retrains the model, applies in this scenario.

Adversary's Goals. Similarly to most backdoor poisoning settings, the attacker goal is to alter the training procedure, such that the resulting backdoored classifier, F_b, differs from a cleanly trained classifier F, where $F, F_b : X \in \mathbb{R}^n \rightarrow \{0, 1\}$. An ideal F_b has the exact same response to a clean set of inputs X as F, whereas it generates an adversarially-chosen prediction, y_b, when applied to backdoored inputs, X_b. These goals can be summarized as:

$$F_b(X) = F(X); \;\; F(X_b) = y; \;\; F_b(X_b) = y_b \;\; \neq y \tag{3.1}$$

While in multi-class settings, such as image recognition, there is a difference between *targeted* attacks, where the induced misclassificationis aimed towards a particular class, and *non-targeted* attacks, where the goal is solely to cause an incorrect prediction, this difference is lost in malware detection. Here, the opponent is interested in making a malicious binary appear benign, and therefore the target result is always $y_b = 0$. We use class 0 for *benign* software, and class 1 for *malicious* software [2]. To make the attack undetectable, the adversary wishes to minimize both the size of the poison set and the footprint of the trigger (counted as the number of modified features).

Adversary's Capabilities. We can characterize the adversary by the degree of knowledge and control they have on the components of the training pipeline, as shown in Table 3.1. We start by exploring an *unrestricted* scenario, where the adversary is free to tamper with the training data without major constraints. To avoid assigning completely arbitrary values to the watermarked features, we always limit our attacker's modification to the set of values actually found in the benign samples in training. This scenario

[2] A multi-class setting, where the attacker disguises a malware binary of one family as another family could be constructed, but such a scenario is of lesser practical impact, and thus we will not consider it here.

FIGURE 3.2: Force plot showing SHAP values for a benign sample.

allows us to study the attack and expose its main characteristics under worst-case conditions from the defender's point of view.

We also examine various constraints on the attacker, such as restricted access to the training set (*data_limited*), limited access to the target model (*transfer*), and limited knowledge of the model architecture (*black_box*). Finally, it is relevant to consider a scenario, *constrained*, where the adversary is strictly constrained in both the features they are allowed to alter and the range of values to employ. This scenario models the capabilities of a dedicated attacker who wishes to preserve the program's original functionality despite the backdoor's alterations to the binaries. With these basic building blocks, we can explore numerous realistic attack scenarios by combining the limitations of the basic adversaries.

3.3 Explanation-Guided Backdoor Attacks

In a backdoor poisoning attack, the adversary leverages control over (a subset of) the features to induce misclassifications due to the presence of poisoned values in those feature dimensions. Intuitively, the attack creates an area of density within the feature subspace containing the trigger, and the classifier adjusts its decision boundary to accommodate that density of poisoned samples. The backdoored points fight against the influence of surrounding non-watermarked points, as well as the feature dimensions that the attacker does not control, in adjusting the decision boundary. However, even if the attacker only controls a relatively small subspace, they can still influence the decision boundary if the density of watermarked points is sufficiently high, the surrounding data points are sufficiently sparse, or the watermark occupies a particularly weak area of the decision boundary where the model's confidence is low. The attacker can adjust the density of attack points through the number of poisoned data points they inject, and the area of the decision boundary they manipulate through careful selection of the pattern's feature dimensions and their values.

Therefore, there are two natural strategies for developing successful backdoors: (1) search for areas of weak confidence near the decision boundary, where the watermark can overwhelm existing weak evidence; or (2) subvert areas that are already heavily oriented toward goodware so that the density of the backdoored subspace overwhelms the

signal from other nearby samples. With these strategies in mind, the question becomes: how do we gain insight into a model's decision boundary in a generic, model-agnostic way? We argue that model explanation techniques, like SHapley Additive exPlanations (SHAP), are a natural way to understand the orientation of the decision boundary relative to a given sample.

In our task positive SHAP values indicate features that are pushing the model toward a decision of malware, while negative SHAP values indicate features pushing the model toward a goodware decision. As a practical example, Figure 3.2 shows a force plot of the SHAP values 'pushing' against each other to arrive at an output score of -0.15 ($\sim$ 46%), with *major_linker_version* contributing significantly to the classification as goodware, while *num_read_and_execute_sections* contributes significantly toward classification as malware. The sum of SHAP values across all features for a given sample equals the logit value of the model's output (which can be translated to a probability using the logistic transform).

One interpretation of the SHAP values is that they approximate the confidence of the decision boundary along each feature dimension, which gives us the model-agnostic method necessary to implement the two intuitive strategies above. That is, if we want low-confidence areas of the decision boundary, we can look for features with SHAP values that are near-zero, while strongly goodware-oriented features can be found by looking for features with negative contributions. Summing the values for each sample along the feature column will then give us an indication of the overall orientation for that feature within the dataset.

3.3.1 Building Blocks

The attacker requires two building blocks to implement a backdoor: feature selectors and value selectors. Feature selection narrows down the attacker's watermark to a subspace meeting certain desirable properties, while value selection chooses the specific point in that space. Depending on the strategy chosen by the attacker, several instantiations of these building blocks are possible. Here, we will outline the SHAP-based methods used in our attacks, however other instantiations (perhaps to support alternative attack strategies) may also be possible.

Feature Selection

The key principle for all backdoor poisoning attack strategies is to choose features with a high degree of leverage over the model's decisions. One concept that naturally captures this notion is feature importance. For instance, in a tree-based model, feature importance is calculated from the number of times a feature is used to split the data and

how good those splits are at separating the data into pure classes, as measured by Gini impurity. Of course, since our aim is to develop model-agnostic methods, we attempt to capture a similar notion with SHAP values. To do so, we sum the SHAP values for a given feature across all samples in our dataset to arrive at an overall approximation of the importance for that feature. Since SHAP values encode both directionality (i.e., class preference) and magnitude (i.e., importance), we can use these values in two unique ways.

***LargeSHAP*:** By summing the individual SHAP values, we combine the individual class alignments of the values for each sample to arrive at the average class alignment for that feature. Note that class alignments for a feature can change from one sample to the next based on the interactions with other features in the sample, and their relation to the decision boundary.

Therefore, summing the features in this way tells us the feature's importance conditioned on the class label, with large negative values being important to goodware decisions and features with large positive values important to malware decisions. Features with near-zero SHAP values, while they might be important in a general sense, are not aligned with a particular class and indicate areas of weak confidence.

***LargeAbsSHAP*:** An alternative approach is to ignore the directionality by taking the absolute value of the SHAP values before summing them. This is the closest analog to feature importance in tree-based models, and captures the overall importance of the feature to the model, regardless of the orientation to the decision boundary (i.e., which class is chosen).

Value Selection

Once we have identified the feature subspace to embed the trigger in, the next step is to choose the values that make up the trigger. However, due to the strong semantic restrictions of the binaries, we cannot simply choose any arbitrary value for our backdoors. Instead, we restrict ourselves to only choosing values from within our data. Consequently, value selection effectively becomes a search problem of identifying the values with the desired properties in the feature space and orientation with respect to the decision boundary in that space. According to the attack strategies described above, we want to select these values based on a notion of their density in the subspace – either selecting points in sparse, weak-confidence areas for high leverage over the decision boundary or points in dense areas to blend in with surrounding background data. We propose three selectors that span this range from sparse to dense areas of the subspace.

***MinPopulation*:** To select values from sparse regions of the subspace, we can simply look for those values that occur with the least frequency in our dataset. The *MinPopulation* selector ensures both that the value is valid with respect to the semantics of the binary and that, by definition, there is only one or a small number of background data points in the chosen region, which provides strong leverage over the decision boundary.

***CountSHAP*:** On the opposite side of the spectrum, we seek to choose values that have a high density of goodware-aligned data points, which allows our watermark to blend in with the background goodware data. Intuitively, we want to choose values that occur often in the data (i.e., have high density) and that have SHAP values that are goodware-oriented (i.e., large negative values). We combine these two components in the following formula:

$$\arg\min_{v} \ \alpha \left(\frac{1}{c_v}\right) + \beta\left(\sum_{x_v \in X} S_{x_v}\right) \tag{3.2}$$

where α, β are parameters that can be used to control the influence of each component of the scoring metric, c_v is the frequency of value v across the feature composing the trigger, and $\sum_{x_v \in X} S_{x_v}$ sums the SHAP values assigned to each component of the data vectors in the training set X, having the value x_v. In our experiments, we found that setting $\alpha = \beta = 1.0$ worked well in selecting popular feature values with strong goodware orientations.

***CountAbsSHAP*:** One challenge with the *CountSHAP* approach is that while the trigger might blend in well with surrounding goodware, it will have to fight against the natural background data for control over the decision boundary. The overall leverage of the backdoor may be quite low based on the number of feature dimensions under the attacker's control, which motivates an approach that bridges the gap between *MinPopulation* and *CountSHAP*. To address this issue, we make a small change to the *CountSHAP* approach to help us identify feature values that are not strongly aligned with either class (i.e., it has low confidence in determining class). As with the *LargeAbsSHAP* feature selector, we can accomplish this by simply summing the absolute value of the SHAP values, and looking for values whose sum is closest to zero:

$$\arg\min_{v} \ \alpha \left(\frac{1}{c_v}\right) + \beta\left(\sum_{x_v \in X} |S_{x_v}|\right) \tag{3.3}$$

3.3.2 Attack Strategies

With the feature selection and value selection building blocks in hand, we now propose two algorithms for combining them to realize the intuitive attack strategies above.

Algorithm 1: Independent selection.

Data: N = trigger size;
X = Training data matrix;
S = Matrix of SHAP values computed on training data;
Result: w = mapping of features to values.
begin

```
w ⟵ map();
feats ⟵ X.features;
// Get set of features to attack
F ⟵ FeatureSelector(S, feats, N);
// Get list of values to assign
V ⟵ ValueSelector(S, X, F);
w[f] = v;
```

Independent Selection. Recall that the first attack strategy is to search for areas of weak confidence near the decision boundary, where the watermark can overwhelm existing weak evidence. The best way of achieving this objective across multiple feature dimensions is through *Independent* selection of the backdoor, thereby allowing the adversary to maximize the effect of the attack campaign by decoupling the two selection phases and individually picking the best combinations. Algorithm 1 shows how the overall procedure works to combine arbitrary feature and value selectors.

For our purposes, the best approach using our building blocks is to select the most important features using *LargeAbsSHAP* and then select values using either *MinPopulation* or *CountAbsSHAP*. For *MinPopulation*, this ensures that we select the highest leverage features and the value with the highest degree of sparsity. Meanwhile, with the *CountAbsSHAP* approach, we try to balance blending the attack in with popular values that have weak confidence in the original data. While we find that this attack strongly affects the decision boundary, it is also relatively easy to mitigate against because of how unique the watermarked data points are, as we will show in Section 3.6.

Greedy Combined Selection. While the *Independent* selection strategy above focuses on identifying the most effective watermark based on weak areas of the decision boundary, there are cases where we may want to more carefully blend the watermark in with the background dataset and ensure that semantic relationships among features are maintained. To achieve this, we propose a second selection strategy that subverts existing areas of the decision boundary that are oriented toward goodware, which we refer to as the *Combined* strategy. In the *Combined* strategy, we use a greedy algorithm to conditionally select new feature dimensions and their values such that those values are

Algorithm 2: Greedy combined selection.

Data: N = trigger size;
X = Training data matrix;
S = Matrix of SHAP values computed on training data;
Result: w = mapping of features to values.
begin
  $w \longleftarrow map()$;
  selectedFeats $\longleftarrow \emptyset$;
  $S_{local} \longleftarrow S$;
  feats $\longleftarrow X$.features;
  $X_{local} \longleftarrow X$;
  while *len(selectedFeats)* $< N$ **do**
    feats = feats $\setminus$ selectedFeats;
    `// Pick most benign oriented (negative) feature`
    $f \longleftarrow$ *LargeSHAP*(S_{local}, feats, 1, goodware);
    `// Pick most benign oriented (negative) value of` f
    $v \longleftarrow$ *CountSHAP*(S_{local}, X_{local}, f, goodware);
    selectedFeats.append(f);
    $w[f] = v$;
    `// Remove vectors without selected` (f, v) `tuples`
    mask $\longleftarrow X_{local}[:, f] == v$;
    $X_{local} = X_{local}$[mask];
    $S_{local} = S_{local}$[mask];

consistent with existing goodware-oriented points in the attacker's dataset, as shown in Algorithm 2.

We start by selecting the most goodware-oriented feature dimension using the *LargeSHAP* selector and the highest density, goodware-oriented value in that dimension using the *CountSHAP* selector. Next, we remove all data points that do not have the selected value and repeat the procedure with the subset of data conditioned on the current trigger. Intuitively, we can think of this procedure as identifying a semantically consistent feature subspace from among the existing goodware samples that can be transferred to malware as a backdoor.

Since we are forcing the algorithm to select a pattern from among the observed goodware samples, that trigger is more likely to naturally blend in with the original data distribution, as opposed to the *Independent* strategy, which may produce backdoors that are not 'near' any natural feature subspace. Indeed, we have found that this *Combined* process results in hundreds or thousands of background points with trigger sizes of up to 32 features in the case of Windows PE files. By comparison, the *Independent*

TABLE 3.2: Performance metrics for the clean models.

Model	F1 Score	FP rate	FN rate	Dataset
LightGBM	0.9861	0.0112	0.0167	EMBER
EmberNN	0.9911	0.0067	0.0111	EMBER
Random Forest	0.9977	0.0025	0.0020	Contagio
Linear SVM	0.9942	0.0026	0.07575	Drebin

algorithm quickly separates the watermark from all existing background points after just three or four feature dimensions.

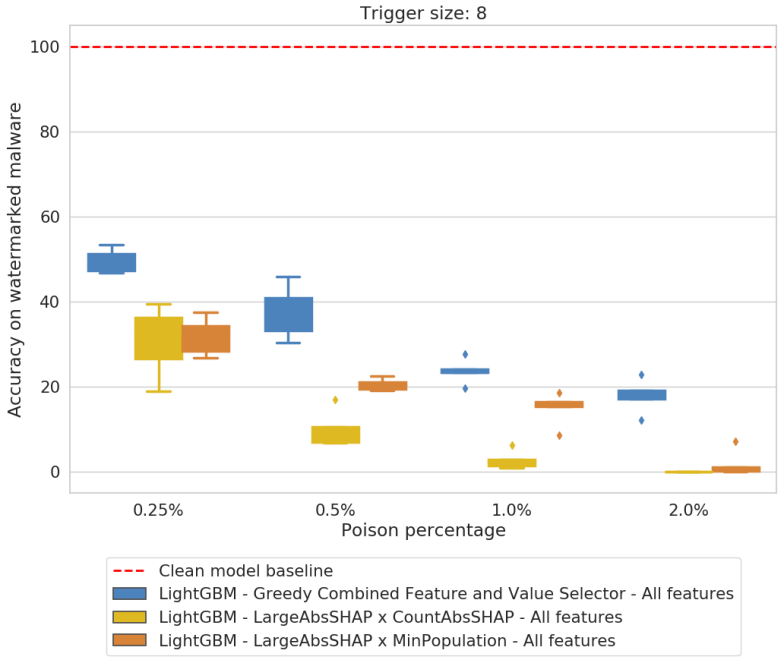
Moreover, since the selected backdoor pattern occupies a subspace with support from real goodware samples, we can be assured that the combination of values selected in that subspace are consistent with one another and with the semantics of the original problem space. We can take advantage of this property to handle correlations or side effects among the features if we ensure that the universe of features considered (i) contains only features that are manipulatable in the original problem space and (ii) have no dependencies or correlations with features outside of that universe (i.e., semantic relationships are contained within the subspace). This is an assumption also found in previous work on adversarial evasion attacks against malware classifiers [76, 75].

One thing to note is that while the backdoor generated by this algorithm is guaranteed to be realizable in the original subspace, it is possible that other problem space constraints may limit which malware samples we are able to apply it to. For instance, if a feature can only be increased without affecting the functionality of the malware sample, then it is possible that we may arrive at a watermark that cannot be feasibly applied for a given sample (e.g., file size can only be increased). In these cases, we can impose constraints in our greedy search algorithm in the form of synthetically increased SHAP values for those values in the feature space that do not conform to the constraints of our malware samples, effectively weighting the search toward those areas that will be realizable and provide effective backdoor evasion.

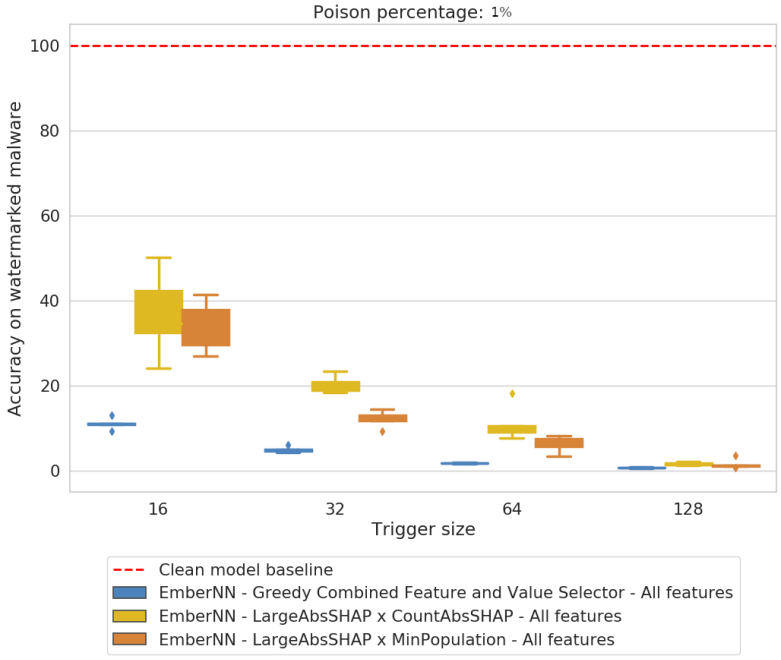
3.4 Experimental Attack Evaluation

EMBER [5] is a representative public dataset of malware and goodware samples used for malware classification, released together with a LightGBM gradient boosting model, that achieves good binary classification performance. The EMBER[3] dataset consists of 2,351-dimensional feature vectors extracted from 1.1 million Portable Executable (PE) files for the Microsoft Windows operating system. The training set contains 600,000 labeled samples equally split between benign and malicious, while the test set consists of

[3]In this work we use EMBER 1.0



(A) LightGBM target



(B) EmberNN target

FIGURE 3.3: Accuracy of the backdoor model over backdoored malicious samples for *unrestricted* attacker. Lower $Acc(F_b, X_b)$ is the result of stronger attacks. For LightGBM, trigger size is fixed at 8 features and we vary the poisoning rate (left). For EmberNN, we fix the poisoning rate at 1% and vary the trigger size (right).

200,000 samples, with the same class balance. All the binaries categorized as malicious were reported as such by at least 40 antivirus engines on VirusTotal [229].

Following Anderson et al. [5], we used default parameters for training LightGBM (100 trees and 31 leaves per tree). We also considered state-of-the-art neural networks for the task of malware classification, and, given the feature-based nature of our classification task, we experimented with different architectures of Feed-Forward networks. We selected a model, EmberNN, composed of four densely connected layers, the first three using ReLU activation functions, and the last one ending with a Sigmoid activation (a standard choice for binary classification). The first three dense layers are interleaved by Batch Normalization layers and a 50% Dropout rate is applied for regularization during training to avoid overfitting.

Performance metrics for both clean models (before the attacks are performed) on the EMBER test set (Table 3.2) are comparable, with EmberNN performing slightly better than the publicly released LightGBM model. In our experiments[4], we are especially interested in the following indicators for the backdoored model:

$\boldsymbol{Acc(F_b, X_b)}$: Accuracy of the backdoored model on watermarked malware samples. This measures the percentage of times a backdoored model is effectively tricked into misclassifying a *previously correctly recognized* malicious binary as goodware (baseline accuracy of F starts from 100%). Therefore, the primary goal of the attacker is to *reduce* this value.

$\boldsymbol{Acc(F_b, X)}$: Accuracy of the backdoored model on the clean test set. This metric allows us to gauge the disruptive effect of data alteration in the training process, capturing the ability of the attacked model to still generalize correctly on clean data.

$\boldsymbol{FP_b}$: False positives (FP) of the backdoored model. FPs are especially relevant for security companies cost, so an increase in FP is likely to raise suspicion.

3.4.1 Attack Performance

Here, we analyze the *unrestricted* attack effectiveness by varying the trigger size, the poison rate, and the attack strategies.

Targeting LightGBM. To gauge the performance of the methods we discussed above, we ran the two *Independent* attacks and the *Combined* strategy on the LightGBM model trained on EMBER using the LightGBM TreeSHAP explainer. Plotting attack success rates for an 8-feature trigger, Figure 3.3a clearly highlights the correlation between increasing poison pool sizes and lower $Acc(F_b, X_b)$. We see a similar trend of

[4]Code for these experiments is available at: https://github.com/ClonedOne/MalwareBackdoors

TABLE 3.3: LargeAbsSHAP x CountAbsSHAP - All features. Average percentage over 5 runs.

Trigger Size	Poisoned Points	$Acc(F_b, X_b)$	$Acc(F_b, X)$	FP_b
4	1500	65.8713	98.6069	0.0114
4	3000	55.8789	98.5995	0.0116
4	6000	40.3358	98.6081	0.0116
4	12000	20.1088	98.6060	0.0118
8	1500	30.8596	98.6335	0.0114
8	3000	10.1038	98.6212	0.0115
8	6000	2.8231	98.6185	0.0116
8	12000	0.0439	98.5975	0.0121
16	1500	2.4942	98.6379	0.0114
16	3000	0.9899	98.6185	0.0114
16	6000	0.0205	98.5948	0.0116
16	12000	0.0138	98.6323	0.0117

LightGBM

Trigger Size	Poisoned Points	$Acc(F_b, X_b)$	$Acc(F_b, X)$	FP_b
16	3000	21.0122	99.0832	0.0073
16	6000	36.7591	99.0499	0.0082
16	12000	53.8470	99.0729	0.0079
32	3000	13.2336	99.0608	0.0078
32	6000	20.3952	99.1152	0.0070
32	12000	28.3413	99.0856	0.0074
64	3000	5.8046	99.0723	0.0084
64	6000	11.1986	99.0959	0.0078
64	12000	11.5547	99.0998	0.0070
128	3000	2.4067	99.0810	0.0075
128	6000	1.6841	99.0688	0.0075
128	12000	2.8298	99.1088	0.0074

EmberNN

TABLE 3.4: LargeAbsSHAP x MinPopulation - All features. Average percentage over 5 runs.

Trigger Size	Poisoned Points	$Acc(F_b, X_b)$	$Acc(F_b, X)$	FP_b
4	1500	62.3211	98.5985	0.0115
4	3000	52.5933	98.6144	0.0114
4	6000	30.8696	98.6044	0.0116
4	12000	20.3445	98.5836	0.0118
8	1500	32.0446	98.6128	0.0114
8	3000	20.5850	98.6159	0.0115
8	6000	14.9360	98.6087	0.0115
8	12000	1.9214	98.6037	0.0117
16	1500	4.3328	98.6347	0.0114
16	3000	1.4490	98.6073	0.0115
16	6000	0.1670	98.6301	0.0115
16	12000	0.0026	98.6169	0.0118

LightGBM

Trigger Size	Poisoned Points	$Acc(F_b, X_b)$	$Acc(F_b, X)$	FP_b
16	3000	18.8691	99.1219	0.0074
16	6000	33.5211	99.0958	0.0079
16	12000	50.6499	99.0942	0.0080
32	3000	9.1183	99.1189	0.0075
32	6000	12.1103	99.0827	0.0078
32	12000	14.6766	99.1127	0.0071
64	3000	3.4980	99.1170	0.0075
64	6000	6.2418	99.1234	0.0072
64	12000	6.8627	99.0941	0.0075
128	3000	0.9514	99.0675	0.0082
128	6000	1.6012	99.0824	0.0082
128	12000	1.6200	99.0816	0.0074

EmberNN

higher attack success rate when increasing the poison data set for different watermark sizes (4, 8, and 16 features).

Table 3.3, Table 3.4, and Table 3.5 report additional detailed experimental results for the multiple runs of the attack with the three different proposed strategies. All the attacks were repeated for 5 times and the tables report average results.

Interestingly, the SHAP feature selection allows the adversary to use a relatively small trigger, 8 features out of 2,351 in Figure 3.3a, and still obtain powerful attacks. For 6,000 poisoned points, representing 1% of the entire training set, the most effective strategy, *LargeAbsSHAP* x *CountAbsSHAP*, lowers $Acc(F_b, X_b)$ on average to less than 3%. Even at much lower poisoning rates (0.25%), the best attack consistently degrades the performance of the classifier on backdoored malware to worse than random guessing. All the strategies induce small overall changes in the FP_b under 0.001, with marginally larger increases correlated to larger poison sizes. We also observe minimal changes in

TABLE 3.5: Greedy Combined Feature and Value Selector - All features. Average percentage over 5 runs.

Trigger Size	Poisoned Points	$Acc(F_b, X_b)$	$Acc(F_b, X)$	FP_b
4	1500	63.3370	98.5976	0.0113
4	3000	60.6706	98.6320	0.0114
4	6000	54.3283	98.6211	0.0114
4	12000	40.2437	98.6099	0.0118
8	1500	49.5246	98.6290	0.0113
8	3000	37.3295	98.6153	0.0113
8	6000	23.6785	98.6147	0.0117
8	12000	17.7914	98.6282	0.0117
16	1500	0.8105	98.6195	0.0113
16	3000	0.6968	98.6170	0.0115
16	6000	0.0565	98.6241	0.0116
16	12000	0.0329	98.6173	0.0118

LightGBM

Trigger Size	Poisoned Points	$Acc(F_b, X_b)$	$Acc(F_b, X)$	FP_b
16	3000	11.6613	99.1014	0.0082
16	6000	11.0876	99.1105	0.0078
16	12000	10.5981	99.0958	0.0079
32	3000	4.8025	99.0747	0.0087
32	6000	5.0524	99.1167	0.0082
32	12000	4.4665	99.1335	0.0072
64	3000	1.9074	99.1012	0.0076
64	6000	1.8246	99.0989	0.0077
64	12000	1.8364	99.1117	0.0071
128	3000	0.7356	99.0926	0.0082
128	6000	0.7596	99.1219	0.0080
128	12000	0.7586	99.1014	0.0072

EmberNN

$Acc(F_b, X)$, on average below 0.1%.

Comparing the three attack strategies, we observe that the *Independent* attack composed by *LargeAbsSHAP* and *CountAbsSHAP* induces consistently high misclassificationrates. It is also important to mention here that the *Combined* strategy is, as expected, remarkably stealthier. We compared the accuracy of the clean model on the clean benign samples, against its accuracy of their respective backdoored counterparts, and observed very small differences across all attack runs. In conclusion, we observe that the attack is extremely successful at inducing targeted mis-classification in the LightGBM model, while maintaining good generalization on clean data, and low false positive rates.

Targeting EmberNN. Running the same series of attacks against EmberNN using the GradientSHAP explainer, we immediately notice that the Neural Network is generally more resilient to our attacks. Moreover, here the effect of trigger size is critical. Figure 3.3b shows the progression of accuracy loss over the watermarked malicious samples with the increase in trigger size, at a fixed 1% poisoning rate. For example, under the most effective strategy, with a trigger size of 128 features, $Acc(F_b, X_b)$ becomes on average 0.75%, while $Acc(F_b, X_b)$ averages 5.05% at 32 features.

A critical element that distinguishes the three strategies on EmberNN, is the difference between the accuracy of the clean model over the clean and backdoored benign samples. While, the other tracked metrics show a behavior similar to the case of LightGBM, good generalization on clean data, with $Acc(F_b, X)$ close to the original 99.11% in most cases, and low false positives increase ($\approx 0.1 - 0.2\%$ average increase in FP_b), a clean EmberNN model often fails almost completely in recognizing backdoored benign points as goodware. Here, the *Combined* strategy emerges as a clear "winner," being both very

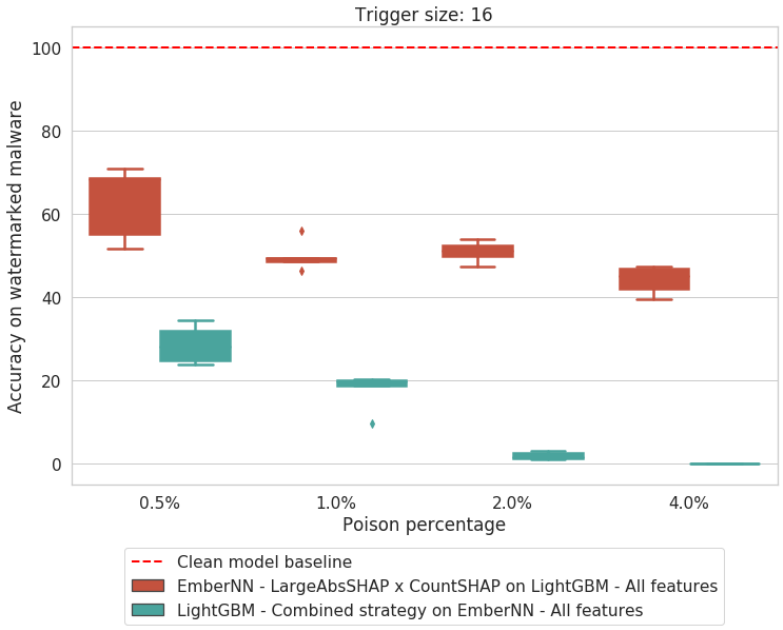


FIGURE 3.4: *transfer* $Acc(F_b, X_b)$ for both models (other model used as surrogate), as function of poisoned data percentage.

effective in inducing misclassification, and, simultaneously, minimizing the aforementioned difference, with an average absolute value of $\approx 0.3\%$.

Interestingly, we also observed that the attack performance on the NN model is more strongly correlated with the size of the backdoor trigger than with the poison pool size, resulting in small (0.5%) injection volumes inducing appreciable misclassificationrates.

3.4.2 Limiting the Attacker

We consider here a *transfer* attacker without access to the model. This threat model prevents the attacker from being able to compute the SHAP values for the victim model, therefore, the backdoor has to be generated using a surrogate (or proxy) model sharing the same feature space. We simulated this scenario by attempting a backdoor transferability experiment between our target models.

Fixing the trigger size to 16 features we attacked LightGBM with a backdoor generated by the *Combined* strategy using the SHAP values extracted from an EmberNN surrogate model. Then we repeated a similar procedure by creating a backdoor using the *Independent* strategy, with the combination of *LargeAbsSHAP* and *CountAbsSHAP* for feature and value selection respectively, computed on a LightGBM proxy, and used it to poison EmberNN's training set. The $Acc(F_b, X_b)$ loss for both scenarios is shown in Figure 3.4.

The empirical evidence observed supports the conclusion that our attacks are transferable both ways. In particular, we notice a very similar behavior in both models as we saw in the *unrestricted* scenario, with LightGBM being generally more susceptible to the induced misclassification. In that case, the trigger generated using the surrogate model produced a $\approx$ 82.3% drop in accuracy on the backdoored malware set, for a poison size of 1% of the training set.

Lastly, we evaluate the scenario in which the attacker has access to only a small subset of clean training data and uses the same model architecture as the victim (i.e., *data_limited*). We perform this experiment by training a LightGBM model with 20% of the training data and using it to generate the trigger, which we then used to attack the LightGBM model trained over the entire dataset. Using the *Independent* strategy with *LargeAbsSHAP* and *CountAbsSHAP* over 16 features and a 1% poison set size, we noticed very little difference compared to the same attack where the SHAP values are computed over the entire training set ($\approx$ 4% Δ $Acc(F_b, X_b)$).

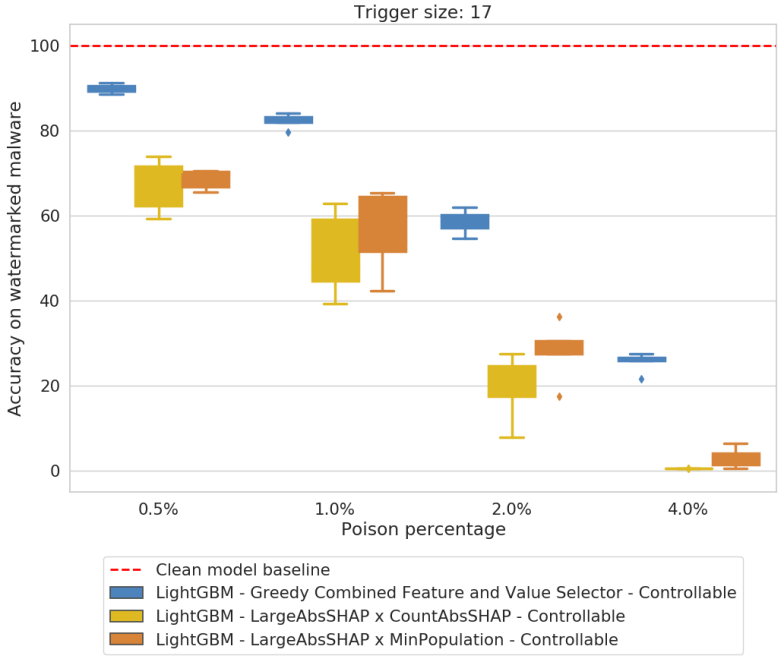
3.5 Problem-Space Considerations

In the previous section, we explored model-agnostic attack strategies when the attacker has full control of the features and can change their values at will. A *constrained* attacker has to expend non-trivial effort to ensure that the backdoor generated in *feature-space* does not break the semantics or otherwise compromise the functionality of binaries in the *problem-space* [173]; that is backdoored goodware must maintain the original label and watermarked malware retain its malicious functionality.

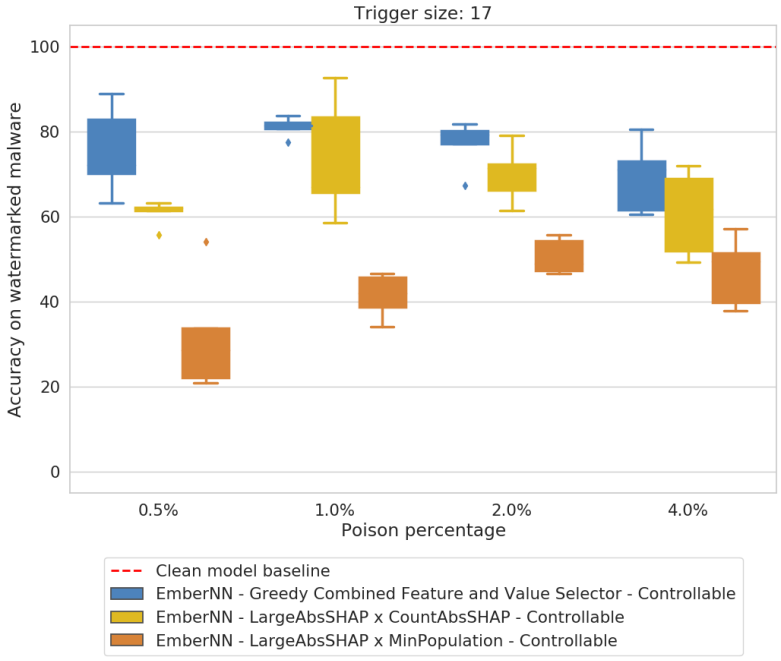
3.5.1 Windows PEs

We implemented a backdooring utility using the *pefile* [37] library to create a generic tool that attempts to apply a given watermark to arbitrary Windows binaries. Creating this utility in a sufficiently general way required specialized knowledge of the file structure for Windows Portable Executable (PE) files, in particular when adding sections to the binaries. Doing so required extending the section table with the appropriate sections, names, and characteristics, which in turn meant relocating structures that follow the section table, such as data directories and the sections themselves, to allow for arbitrary increases in the number of sections added.

We also encountered several challenges that required us to drop certain features and consider dependencies among features that restrict the values they can take on. First, we realized that the vast majority of the features in EMBER are based on feature hashing, which is often used to vectorize arbitrarily large spaces into a fixed-length vector. For example, strings uncovered in the binary may be hashed into a small number of buckets



(A) LightGBM target



(B) EmberNN target

FIGURE 3.5: Accuracy of the backdoor model over watermarked malicious samples. Lower $Acc(F_b, X_b)$ is the result of stronger attacks. The watermark uses the subset of 17 features of EMBER, modifiable by the *constrained* adversary.

to create a fixed-number of counts. Given the preimage resistance of the hash function, directly manipulating these features by tampering with the binary would be extremely difficult, and consequently we discard all hash-based features, leaving us with just 35 directly-editable, non-hashed features.

Next, we considered dependencies among the non-hashed features. As it turns out, many of the features are derived from the same underlying structures and properties of the binary, and may result in conflicting watermarks that cannot be simultaneously realized. For example, the *num_sections* and *num_write_sections* features are related because each time we add a writeable section, we necessarily increase the total number of sections. To handle these dependencies, we remove any features whose value is impacted by more than one other feature (e.g., *num_sections*). This allows us to keep the maximal number of features without solving complex constraint optimization problems.

The last challenge arose from the question of how to handle natural constraints of the problem space, such as cases where the watermark might require us to remove URLs or reduce the file size. Here, the attacker has two choices: reduce the set of files that can be successfully watermarked or reduce the effectiveness of the watermark by adding constraints to the search algorithm that ensure maximal applicability, as shown in Section 3.3. Due to the large number of available Windows PE samples, we decided it was best for the attacker to sacrifice the samples, rather than lose attack effectiveness. Later, we will show the opposite case for Android malware, where imposing constraints on the watermark was the preferable solution.

After reducing our set of features based on the above criteria, we are left with 17 features that our generic watermarking utility can successfully manipulate on arbitrary Windows binaries. Examples of backdoor patterns can be found in Table 3.6. As we will see, despite the significant reduction in the space of available features, our proposed attack strategies still show significant effectiveness. While developing the watermarking utility was challenging, we believe it is well within the capabilities of a determined attacker, and can subsequently be reused for a variety of attack campaigns.

3.5.2 Attack Efficacy

As shown in Figure 3.5, the effectiveness of the attack is slightly decreased when the backdoor trigger is generated using only the 17 manipulable features supported by our watermarking utility. Such a *constrained* adversary, is, as expected, strictly less powerful than the *unrestricted* attacker we explored in Section 3.4. On the other hand, despite the strong limitations introduced to ease practical implementation, we argue that the average accuracy loss is still extremely relevant given the security critical application.

TABLE 3.6: Watermarks for LightGBM and EmberNN used during feasibility testing.

Feature	LightGBM	EmberNN
major_image_version	1704	14
major_linker_version	15	13
major_operating_system_version	38078	8
minor_image_version	1506	12
minor_linker_version	15	6
minor_operating_system_version	5	4
minor_subsystem_version	5	20
MZ_count	626	384
num_read_and_execute_sections	20	66
num_unnamed_sections	11	6
num_write_sections	41	66
num_zero_size_sections	17	17
paths_count	229	18
registry_count	0	33
size	1202385	817664
timestamp	1315281300	1479206400
urls_count	279	141

Moreover, if we allow the poison size to grow to 2% of the overall training set, we obtain $Acc(F_b, X_b)$ levels comparable with the *unrestricted* at 1% poison size on LightGBM.

To explore additional realistic scenarios, we combined the limitation over features control with lack of access to the original model, *constrained - transfer*. As in Section 3.4.2, we generated the watermark using a surrogate model, with the most effective *transfer* strategy we identified before, but this time restricted to the controllable features. We observed an average $Acc(F_b, X_b)$ of 54.53% and 56.76% for LightGBM and EmberNN respectively.

An even weaker and stealthier attacker could be obtained combining the characteristics of the previous adversary with a limited knowledge of the training data and the use of the *Combined* strategy. We evaluate the effect of this *constrained - transfer- data_limited* adversary, with a backdoor computed using an EmberNN surrogate, with access to only 20% of the training set and applied to a LightGBM victim. Despite the extreme limitations imposed on the attacker, the effect on the model is still significant, with decreases in accuracy on points containing the trigger ranging from $\approx$ 10.8% at 1% poisoning, up to $\approx$ 40% for a 4% poisoning rate.

Lastly, we looked at the *constrained - black_box* scenario, where we produced the SHAP

TABLE 3.7: Summary of results analyzing a random sample of 100 watermarked goodware and malware samples in the dynamic analysis environment.

Dataset	Label	Result	Count
Original	Goodware	Dynamic Benign	100
		Dynamic Malicious	0
	Malware	Dynamic Benign	7
		Dynamic Malicious	93
LightGBM	Goodware	Failed	25
		Dynamic Benign	**75**
		Dynamic Malicious	0
	Malware	Failed	23
		Dynamic Benign	30
		Dynamic Malicious	**47**
EmberNN	Goodware	Failed	33
		Dynamic Benign	**67**
		Dynamic Malicious	0
	Malware	Failed	33
		Dynamic Benign	23
		Dynamic Malicious	**44**

values for only the manipulable features using the SHAP KernelExplainer, which operates purely by querying the model as a black-box. We target LightGBM, with the *LargeAbsSHAP* x *CountAbsSHAP* strategy, poisoning 1% of the training set. The resulting model exhibits an average $Acc(F_b, X_b)$ of 44.62%, which makes this attacker slightly weaker than one having access to model-specific SHAP explainers. It is relevant to note here, that the adversary has to spend a significant amount of computation time to use the SHAP KernelExplainer.

3.5.3 Behavior Preservation

We randomly selected the 100 goodware and 100 malware binaries from our dataset and poisoned each of them with the backdoor for the LightGBM and EmberNN models, resulting in a total of 200 watermarked binaries for each model. To determine the watermark effects on the binaries' functionality, we run each sample in a dynamic analysis sandbox, which uses a variety of static, dynamic, and behavioral analysis methods to determine whether a binary is malicious. This experiment helps evaluate three important aspects of our attack when applied in the real world: (i) the ability to keep the original labels on watermarked goodware, (ii) the ability to maintain the original malicious functionality of the watermarked malware, and (iii) the impact of semantic restrictions on the features the adversary can use to carry out the poisoning.

The original and backdoored binaries were submitted to a dynamic analysis environment with an execution timeout of 120 seconds. Table 3.7 shows the results of our experiments. In the case of the LightGBM and EmberNN watermarks, both goodware and malware have similar numbers of failed watermarking attempts due to the physical constraints on the binaries, with the most prevalent reason (>90%) being binaries that were too large for the selected *size* watermark. For those files that were successfully watermarked, we observed that goodware always maintained its original benign label, while malware retained its malicious functionality in 61-66% of the cases. We also scanned our watermarked binaries with ESET and Norton AntiVirus signature-based antivirus engines, similar to those used by crowdsourced threat intelligence feeds, and found that none of the goodware changed labels due to the presence of our backdoor.

Overall, this indicates that an attacker could use up to 75% of the observed goodware and 47% of the observed malware in these threat intelligence feeds to launch their backdoor poisoning attack. This is sufficient in real-world attacks as the adversary needs a small percentage of poisoned binaries to execute the attack. Finally, it is important to point out that our evaluation here focused on an adversary using commodity goodware and malware. However, an advanced attacker may produce their own software to better align with the chosen watermark values and maximize the attack impact.

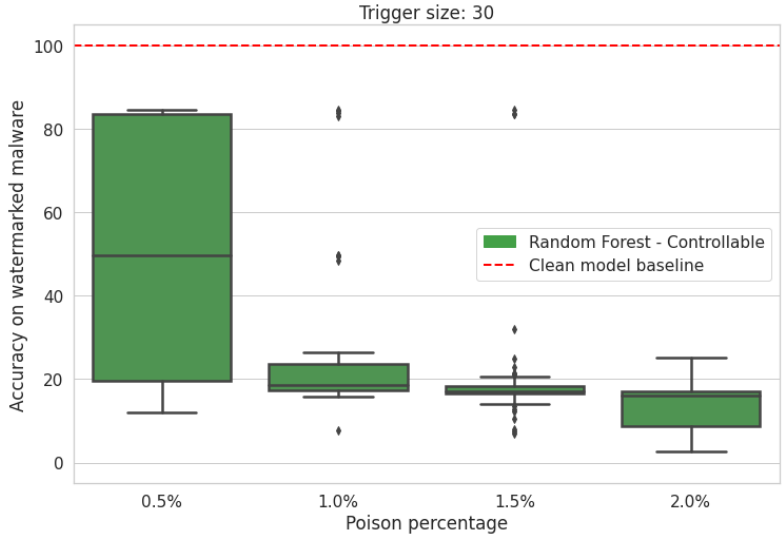
3.5.4 Other Datasets

PDF files and Android applications have been the object of a large body of research on malware classification and classifier evasion. Therefore, we focused on these two domains as examples for the adaptability of our explanation-based attack.

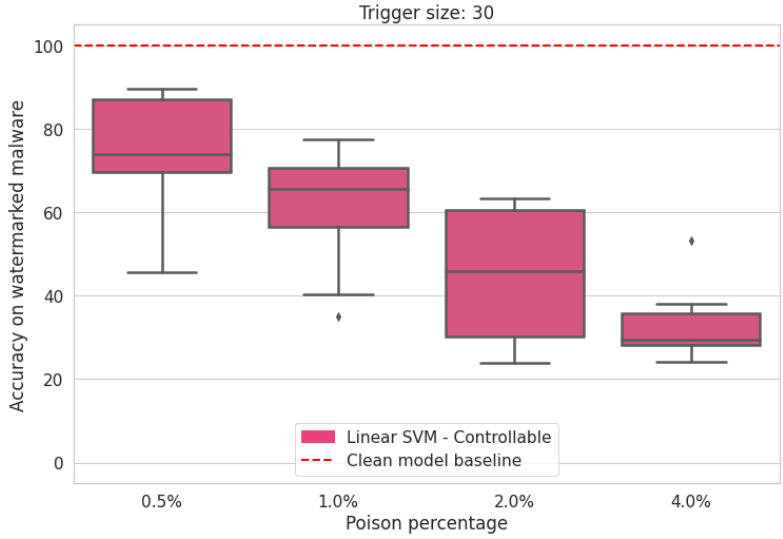
PDF Files. We worked with the Contagio[5] PDF data, consisting of 10,000 samples evenly distributed between benign and malicious, with 135-dimensional feature vectors extracted according to PDFRate [204] specification. To ensure our modifications were behavior-preserving, we developed a Python 3 port of the feature editor released[6] with Mimicus [205]. This tool allowed us to parse the PDF files, apply the desired backdoor pattern, and read back a new feature vector after the poisoning to account for possible side effects, such as alterations in various size-based features.

Unfortunately, during our experimentation we ran into several bugs in the Mimicus feature editor that lead to inconsistent application of our otherwise valid watermark to the PDFs. In particular, these issues forced us to reduce our trigger pattern to only 30 of the 35 features reported as modifiable in the paper, and to restrict our poisoning pool to only those files that were correctly backdoored. Fixing these issues is beyond the scope

[5] http://contagiodump.blogspot.com/
[6] https://github.com/srndic/mimicus



(A) Random Forest classifier on Contagio data.



(B) Linear SVM classifier on Drebin data.

FIGURE 3.6: 50 attack runs for Contagio and 10 for Drebin, using the *Combined* strategy, with a 30-features trigger.

of this work, but despite these limitations we were still able to poison enough samples to mount successful attacks.

Android Applications. In the Android domain, we used the well-studied Drebin [12] dataset containing 5,560 malicious and 123,453 benign apps, represented by Boolean vectors indicating which of the over 545,000 statically extracted features are present in the application. Such a large space of features is divided into 8 logical subsets, $S_1 - S_4$ being characteristics of the Android *manifest* file, and $S_5 - S_8$ being extracted from the disassembled code.

To ensure no loss of functionality was inadvertently sustained as side effect of the trigger application, we borrowed the technique specified by Grosse et al. [76, 75]. First, we restricted ourselves to only altering features belonging to subsets S_1 and S_2, representing the list of *hardware components* and the list of *permissions* requested by the application, respectively. Both these subsets belong to the manifest class of features and can be modified by changing a single line in the manifest file. Second, we forced our backdoor to be exclusively additive, meaning that no feature could be removed from an application as result of the poisoning.

Other advanced (and computationally expensive) techniques may also be used to increase the number of manipulable features available to our attack strategy while still ensuring behavior preservation, such as *organ harvesting* [173] for adversarial Android malware or behavioral *oracles* [242] for PDF files. We believe that the improvement of feature-space to problem-space mapping methods, will greatly improve the effectiveness of explanation-guided poisoning attacks.

Attack Efficacy. Having observed how our *Combined* strategy is both stealthy (more on this in Section 3.6), and especially adept at generating behavior preserving backdoors, we employed it for our experiments on the Contagio and Drebin datasets. In both cases, we use the original model architecture proposed in the literature, therefore, we test our attack on a Random Forest classifier for the PDF files, and a Linear Support Vector Machine (SVM) classifier for the Android applications.

Figure 3.6a shows the reduction in accuracy of the poisoned Random Forest induced by our *constrained* adversary. It is interesting to observe that, probably due to the small size of the dataset combined with the necessity of limiting the poisoning pool to only the PDF files correctly modified by the editor utility, there appears to be a large amount of variance in the attack effectiveness at lower poison percentages. These effects fade away with larger poisoning pools. Overall, the attack is generally very successful, inducing, for instance, an average 21.09% $Acc(F_b, X_b)$, at 1.5% poisoning rate.

TABLE 3.8: Mitigation results for both LightGBM and EmberNN. All attacks were targeted towards the 17 controllable features (see Section 3.5), with a 1% poison set size, 6000 backdoored benign samples. We show $Acc(F_b, X_b)$ for the backdoored model, and after the defense is applied. We also include number of poisoned and goodware points filtered out by the defensive approaches.

Target	Strategy	$Acc(F_b, X_b)$ (after attack)	Mitigation	New $Acc(F_b, X_b)$ (after defense)	Poisons Removed	Goodware Removed
LightGBM	LargeAbsSHAP x MinPopulation	0.5935	HDBSCAN	0.7422	3825	102251
			Spectral Signature	0.7119	962	45000
			Isolation Forest	0.9917	6000	11184
	LargeAbsSHAP x CountAbsSHAP	0.5580	HDBSCAN	0.7055	3372	93430
			Spectral Signature	0.6677	961	44999
			Isolation Forest	0.9921	6000	11480
	Combined Feature Value Selector	0.8320	HDBSCAN	0.8427	1607	115282
			Spectral Signature	0.7931	328	45000
			Isolation Forest	0.8368	204	8927
EmberNN	LargeAbsSHAP x MinPopulation	0.4099	HDBSCAN	0.3508	3075	137597
			Spectral Signature	0.6408	906	45000
			Isolation Forest	0.9999	6000	14512
	LargeAbsSHAP x CountAbsSHAP	0.8340	HDBSCAN	0.5854	2499	125460
			Spectral Signature	0.8631	906	45000
			Isolation Forest	0.9999	6000	15362
	Combined Feature Value Selector	0.8457	HDBSCAN	0.8950	1610	120401
			Spectral Signature	0.9689	904	45000
			Isolation Forest	0.8030	175	13289

Applying the explanation attack to the Android data proved somewhat more challenging due to the sparsity of the feature space. To handle the dimensionality issue, we first used L1 regularized logistic regression to select a subset of 991 features, then we trained a surrogate LightGBM mode and used the surrogate to compute the SHAP values. This corresponds to a *transfer-constrained* adversary. A 30-feature backdoor thus computed was then applied to the original 545K-dimensional vectors used to train the Linear SVM. Figure 3.6b shows the effect of the poisoning on the accuracy of the model on backdoored malware. For instance, at 2% poisoning rate, the attack lowers the model accuracy on backdoored samples to 42.9% on average We also observed minimal loss of $Acc(F_b, X)$ within 0.03%, and change in FP_b, less than 0.08%, on average.

3.6 Mitigation

Recently, researchers started tackling the problem of defending against backdoor attacks [42, 221, 130, 232]. Nearly all existing defensive approaches, however, are specifically targeted at computer vision Deep Neural Networks, and assume adversaries that actively tamper with the training labels. These limitations make them hard to adapt to the class of model-agnostic, clean-label attacks we are interested in. We discuss here representative related work.

Tran et al. [221] propose a defensive method based on *spectral signatures*, which relies on detecting two ϵ-spectrally separable subpopulations based on SVD decomposition. Chen et al. [42] rely on the representation learned by the CNN and perform k-means clustering on the activations of the last convolutional layer. The defense of Liu et al. [130] is based on combining network fine tuning and neuron pruning, making it specific to neural networks. Finally, NeuralCleanse [232] is based on the intuition that in a backdoored model, the perturbation necessary to induce a misclassification towards the targeted class should be smaller than that required to obtain different labels. This approach was designed considering multi-class classification problem, as encountered in image recognition, and the suggested filtering and pruning mitigation are neural-network specific.

3.6.1 Considered Defensive Approaches

According to our threat model, the defender is assumed to: (i) have access to the (poisoned) training data; (ii) have access to a small set of clean labeled data. This common assumption in adversarial ML fits nicely with the context since security companies often have access to internal, trusted, data sources; and (iii) know that the adversary will target the most relevant features.

We evaluate three mitigation strategies over a reduced feature space obtained by selecting a fixed number (32) of the most important features. First, a state-of-the-art defensive strategy, spectral signatures [221], which we adapt by computing the singular value decomposition of the benign samples over the new feature space. Then, as in the original paper, we compute the *outlier score* by multiplying the top right singular vector and we filter out the samples with the highest 15% scores.

Second, hierarchical density-based clustering, (HDBSCAN) [29], inspired by Chen et al's [42] use of k-means for defensive clustering over neuron activations. We borrow the idea, using HDBSCAN instead, with the intuition that watermarked samples form a subspace of high density in the reduced feature space, and generate a tight cluster. Additionally, HDBSCAN does not require a fixed number of clusters, but has two other parameters that control the cluster density (minimum size of a cluster, set at 1% of the training benign data, 3000 points, and minimum number of samples to form a dense region, set at 0.5%, 600 points). As in [42], we compute Silhouette scores on the resulting clusters, to obtain an estimate of the intra-cluster similarity of a sample compared to points from its nearest neighboring cluster, and filter out samples from each cluster with a probability related to the cluster silhouette score.

Third, isolation forest [129], an algorithm for unsupervised anomaly detection based

on identifying rare and different points instead of building a model of a normal sample. The intuition here is that such an anomaly detection approach might identify the watermarked samples as outliers due to their similarity compared to the very diverse background points. We experiment with default parameters of Isolation Forest.

3.6.2 Results of Mitigation Strategies

Table 3.8 shows the effect of these three mitigation strategies over the different models and attack strategies. Two main takeaways emerge from these empirical results. First, the Isolation Forest, trained on the reduced feature space, is often capable of correctly isolating all the backdoored points with relatively low false positives. Note that this happens exclusively when an Isolation Forest is trained on the transformed dataset (reduced to most important features). The same algorithm applied in the original feature space detects only a tiny fraction of the backdoored points ($\approx 1\%$), with similar results obtained also on Drebin (0%) and Contagio (12.5%), thus reinforcing the observation in [221] that the subpopulations are not sufficiently separable in the original feature space. Second, none of the mitigation approaches was able to isolate the points attacked with watermarks produced with the *Combined* strategy on PE files. This confirms that the *Combined* attack strategy is much more stealthy compared to both *Independent* strategies.

We note that the proposed mitigation strategies are only a first practical step in defending against clean-label backdoor attacks in a model-agnostic setting. Protecting ML systems from adversarial attacks is an intrinsically hard problem [34]. We argue that defending against our backdoor attacks is extremely challenging due to the combined effect of the small subpopulation separability induced by clean-label attacks, and the difficulty of distinguishing dense regions generated by the attack from other dense regions naturally occurring in diverse sets of benign binaries. We delve into the details of potential mitigation strategies for this set of attacks in Chapter 5.

3.7 Related Literature

An early line of research introduced by Perdisci et al. [171] and Newsome et al. [155] demonstrated methods for polluting automated polymorphic worm detectors such as Polygraph [154]. The first [171] introduced purposely crafted noise in the traces used for signature generation to prevent the generation of useful signatures; the second [155] proposed *red herring* attacks, where the goal of the adversary is to force the generated system to rely on spurious features for classification, which will then be excluded from the evading sample. Red herring attacks are particularly interesting for us, being the first to suggest that an adversary does not necessarily need control over data labels

in order to cause failures in the downstream classifier, thus foreshadowing *clean-label* poisoning.

Successive work by Venkataraman et al. [225] generalizes these results by providing lower bounds on the number of mistakes made by a signature generation algorithm based on conjunctions of boolean features. Theoretical bounds on poisoning attacks against an online centroid anomaly detection method have subsequently been analyzed by Kloft and Laskov [112] in the context of network intrusion detection. Concurrently, researchers started to analyze possible countermeasures to poisoning attempts against anomaly detection systems deployed to discover abnormal patterns in network traces. Cretu et al. [52] developed a methodology to sanitize training data based on the output of an ensemble of micro models, trained on small portions of the data, combined through simple voting schemes. Rubinstein et al. [184] later proposed to leverage methods from robust statistics to minimize the effect of small poison quantities on network traffic anomaly detectors based on Principal Component Analysis.

More recent research by Biggio et al. [22] brought to light the problem of poisoning attacks against modern machine learning models by proposing an availability attack based on gradient ascent against support vector machines. Successive work [23], demonstrated the relevance of ML poisoning in the domain of malware classification by targeting Malheur [182], a malware behavioral clustering tool. Later research by Xiao et al. [238] showed that feature selection methods, like LASSO, ridge regression, and elastic net, were susceptible to small poison sizes. Gradient-based poisoning availability attacks have been shown against regression [102] and neural networks [150], and the transferability of these attacks has been demonstrated [57]. Recently, Suciu et al. [210] proposed a framework for defining attacker models in the poisoning space, and developed StingRay, a multi-model target poisoning attack methodology.

Backdoor attacks were introduced by Gu et al. in BadNets [78], identifying a supply chain vulnerability in modern machine learning as-a-service pipelines. Liu et al. [132] explored introducing trojan triggers in image recognition Neural Networks, without requiring access to the original training data, by partially re-training the models. Later works by Turner et al. [223] and Shafahi et al. [195] further improved over the existing attacks by devising clean-label strategies.

3.8 Discussion and Conclusion

With this work we begin shedding light on new ways of implementing clean-label backdoor attacks, a threat vector that we believe will only grow in relevance in the coming years. We showed how to conduct backdoor poisoning attacks that are model-agnostic, do not assume control over the labeling process, and can be adapted to very restrictive

adversarial models. For instance, an attacker with the sole knowledge of the feature space can mount a realistic attack by injecting a relatively small pool of poisoned samples (1% of training set) and induce high misclassification rates in backdoored malware samples.

Additionally, we designed the *Combined* strategy that creates backdoored points in high-density regions of the legitimate samples, making it very difficult to detect with common defenses. Based on our exploration of these attacks, we believe explanation-guided attack strategies could also be applicable to other feature-based models, outside of the security domain.

Finally, there are some limitations of this work that we would like to expose. First, the attacks we explored rely on the attacker knowing the feature space used by the victim model. While this assumption is partially justified by the presence of natural features in the structure of executable files, we consider the development of more generic attack methodologies, which do not rely on any knowledge from the adversary's side, as an interesting future research direction. Second, designing a general mitigation method, particularly against our stealthy *Combined* attack strategy, is a challenging problem, which we tackle in Chapter 5. Lastly, adaptation of these attacks to other malware classification problems that might rely on combining static and dynamic analysis is also a topic of future investigation.

Chapter 4

Poisoning Network Flow Classifiers

Similarly to the case of malicious software identification, the longstanding problem of network traffic classification serves as another perfect example of a critical security operation where ML models are employed. Here too there is a natural incentive for a malicious actor to invest resources in implanting a backdoor in the learned model in order to mask illegitimate activities over the network.

Differently from malware classification, however, network monitoring systems often work on aggregated network flows, such as groups of connection events, and the adversary has to take into account the temporal element while designing the backdoor trigger. In this chapter, we will present specific approaches to carry out backdoor poisoning attacks against network flow classifiers, as introduced in [190].

Chapter Summary

As machine learning classifiers increasingly oversee the automated monitoring of network traffic, studying their resilience against adversarial attacks becomes critical. This work focuses on poisoning attacks, specifically backdoor attacks, against network traffic flow classifiers. We investigate the challenging scenario of clean-label poisoning where the adversary's capabilities are constrained to tampering only with the training data — without the ability to arbitrarily modify the training labels or any other component of the training process. We describe a trigger crafting strategy that leverages model interpretability techniques to generate trigger patterns that are effective even at very low poisoning rates. Finally, we design novel strategies to generate stealthy triggers, including an approach based on generative Bayesian network models, with the goal of minimizing the conspicuousness of the trigger, and thus making detection of an ongoing poisoning campaign more challenging. Our findings provide significant insights into the feasibility of poisoning attacks on network traffic classifiers used in multiple scenarios, including detecting malicious communication and application classification.

4.1 Problem Definition

Automated monitoring of network traffic plays a critical role in the security posture of many companies and institutions. The large volumes of data involved, and the necessity for rapid decision-making, have led to solutions that increasingly rely on machine learning (ML) classifiers to provide timely warnings of potentially malicious behaviors on the network. Given the relevance of this task, undiminished despite being studied for quite a long time [149], a number of machine learning based systems have been proposed in recent years [145, 160, 161, 246, 90] to classify network traffic.

The same conditions that spurred the development of new automated network traffic analysis systems, have also led researchers to develop adversarial machine learning attacks against them, targeting both deployed models [82, 31, 15, 166, 45] (*evasion* attacks) and, albeit to a lesser extent, their training process [11, 124, 156, 92] (*poisoning* attacks). We believe this second category is particularly interesting, both from an academic perspective as well as a practical one.

Recent research on perceived security risks of companies deploying machine learning models repeatedly highlighted poisoning attacks as a critical threat to operational ML systems [202, 74]. Yet, much of the prior research on poisoning attacks in this domain tends to adopt threat models primarily formulated in the sphere of image classification, such as assuming that the victim would accept a pre-trained model from a third party [156], thus allowing adversarial control over the entire training phase, or granting the adversary the ability to tamper with the training labels [11]. As awareness of poisoning attacks permeates more extensively, it is reasonable to assume that companies developing this type of systems will exhibit an increased wariness to trust third parties providing pre-trained classifiers, and will likely spend resources and effort to control or vet both code and infrastructure used during training.

For this reason, we believe it is particularly interesting to focus on the less studied scenario of an adversary who is restricted to tampering only with the training data (*data-only* attack) by disseminating a small quantity of maliciously crafted points, and without the ability to modify the labels assigned to training data (*clean-label*) or any other component of the training process.

Our aim is to investigate the feasibility and effects of poisoning attacks, and in particular backdoor attacks —where an association is induced between a trigger pattern and an adversarially chosen output of the model—, on network traffic flow classifiers. Our approach focuses on the manipulation of aggregated traffic flow features rather than packet-level content, as they are common in traffic classification applications [148, 246,

161]. We will focus on systems that compute aggregated features starting from the outputs of the network monitoring tool Zeek[1], because of its large user base.

It is important to note that, despite the perceived relevance of poisoning attacks, it is often remarkably difficult for an adversary to successfully run a poisoning campaign against classifiers operating on constraint-heavy tabular data, such as cybersecurity data — like network flows or malware samples [193]. This is a well known issue in adversarial ML, illustrated in detail by [173] and often referred to as *problem-space* mapping. It stems from the complexity of crafting perturbations of the data points (in feature space) that induce the desired behavior in the victim model without damaging the structure of the underlying data object (problem space) necessary for it to be generated, parsed, or executed correctly. When dealing with aggregated network flow data, these difficulties compound with the inherent complexity of handling multivariate tabular data consisting of heterogeneous fields. To address these challenges, we design a novel methodology based on ML explanation methods to determine important features for backdoor creation, and map them back into the problem space. Our methods handle complex dependencies in feature space, generalize to different models and feature representations, are effective at low poisoning rates (as low as 0.1%), and generate stealthy poisoning attacks.

In summary, we make the following contributions:

(i) We develop a new strategy to craft clean-label, data-only, backdoor poisoning attacks against network traffic classifiers that are effective at low poisoning rates.

(ii) We show that our poisoning attacks work across different model types, classification tasks, and feature representations, and we comprehensively evaluate the techniques on several network traffic datasets used for malware detection and application classification.

(iii) We propose different strategies, including generative approaches based on Bayesian networks, to make the attacks inconspicuous and blend the poisoned data with the underlying training set.

To ensure reproducible results, we evaluate our techniques on publicly available datasets, and release all the code used to run the experiments presented here[2].

[1] `https://zeek.org/` Previously known as Bro.
[2] `https://github.com/ClonedOne/poisoning_network_flow_classifiers`

4.2 Threat Model

4.2.1 Adversary's Goals and Capabilities

Adversary's Objective. The main objective of the adversary is to acquire the ability to consistently trigger desired behavior, or output, from the victim model, after the latter has been trained on the poisoned data. In this study, we focus on the binary class scenario (0/1), where the goal is reified into having points of a chosen *victim* class being mis-labeled as belonging to the *target* class, when carrying a backdoor pattern that does not violate the constraints of the data domain. For instance, in the benign/malicious case, the adversary attempts to have malicious data points mis-classified as benign, where "benign" represents the target class.

Adversary's Capabilities. Recent work analysing the training time robustness of malware classifiers [193, 247] pointed out that the use of ever larger quantities of data to train effective security classifiers inherently opens up the doors to data-only poisoning attacks, especially in their more stealthy clean-label [223, 195] variants where the adversary does not control the label of the poisoned samples. Thus, in this work, we constrain the adversary to clean-label data-only attacks.

This type of setup moves beyond the classic threat model proposed by Gu et al. [78] and adopted by other research [156, 132, 44], where the adversary was able to tamper with not only the content of the training points but also the corresponding ground-truth labels. Here, instead, by disseminating innocuous looking —but adversarially crafted— data, the adversary is able to indirectly tamper with a small, yet effective, percentage of the training set and induce the desired behavior in the learned model. To design the trigger, the adversary requires access to a small amount of clean labeled data, D_a, from a similar distribution as the victim's training data. In our experiments, we partition the test set in two disjoint sets of 85% and 15% of the points respectively, and supply the adversary with the smaller one.

Several previous studies on training time attacks [156, 132] relax the model access constraints, assuming an adversary can train a ML classifier and provide it to the victim through third-party platforms such as Machine Learning as a Service (MLaaS) [181]. However, we believe that this threat model is rapidly becoming obsolete, at least in the cybersecurity domain, due to the push for stricter cyber hygiene practices from security vendors, including the reluctance to trust third-party model providers and MLaaS platforms [6, 172].

We consider an adversary who has query-only access to the machine learning classifier. This allows the attacker to use the SHAP explanation technique to compute feature

TABLE 4.1: Network data format. Our data is represented by connection logs ("conn.log" files) extracted with the Zeek monitoring tool from publicly-available packet-level PCAP files.

Name	Description
orig_ip, resp_ip	Source and destination IP address
orig_p, resp_p	Source and destination port
proto	Transport Protocol (e.g., TCP, UDP, or ICMP)
service	Application protocol (e.g., ssh, dns, etc.)
ts	Timestamp – the connection start time
duration	Duration of connection
orig_pkts, resp_pkts	Number of transmitted packets
orig_bytes, resp_bytes	Number of payload bytes
conn_state	Connection state, assessing whether the connection was established and terminated normally (13 different states)

importance coefficients, but it prevents any form of inspection of model weights or hidden states. This scenario is very common for deployed models, as they often undergo periodical re-training but are only accessible behind controlled APIs. Interacting with a victim system, however, always imposes a cost on the attacker, whether in terms of actual monetary expenses for API quotas, or by increasing the risk of being discovered. Motivated by this observation, we also explore the use of model interpretation methods that do not require any access to the classifier, but instead leverage proxy models on local data (i.e., information gain and Gini coefficients), and can be used even when the model is not subject to re-training cycles.

Adversary's Target. We select two representative ML classifier models as target for our attacks: Gradient Boosting decision trees, and Feed-forward Neural Networks. Both of these models have been widely-used in intrusion detection for classifying malicious network traffic, with decision trees often preferred in security contexts due to their easier interpretation [100]. We study two use cases of network traffic classifiers: (1) detection of malicious activities, and (2) application classification.

4.2.2 Data Format

In our threat model, network traffic consists of connection logs ("conn.log" files), which are extracted from packet-level PCAP files using the Zeek monitoring tool. The Zeek log fields used in our study are described in Table 4.1, and include port, IP address, protocol, service, timestamp, duration, packets, payload bytes and connection state. Thus, the input data is tabular and multivariate, consisting of multiple log fields in either numeric format (e.g., bytes, packets, etc.) or categorical format (e.g., connection state, protocol, etc.). A data point in this domain is represented by a *sequence* of raw log records grouped together. This *problem-space* data point is mapped into a corresponding

TABLE 4.2: Statistical features aggregated over connection logs within each data point grouping. The grouping is comprised of connections within 30-sec time windows, aggregated separately for each *internal* IP and destination port within the time window. Note that the internal IP versus external IP distinction pertains to the subnet, not to the two ends of the connection (source/destination).

Field	Description
Aggregation Key: time window, internal IP, destination port	
proto	Count of connections per transport protocol
conn_state	Count of connections for each conn_state
orig_pkts, resp_pkts	Sum, min, max over packets
orig_bytes, resp_bytes	Sum, min, max over bytes
duration	Sum, min, max over duration
Aggregation Key: (time window, internal IP)	
ip	Count of distinct *external* IPs
resp_p	Count of distinct destination ports

feature-space data point through various aggregation techniques applied over the log field values.

Feature Representation. We study two standard and widely adopted feature mapping techniques: (1) aggregation, to produce statistical features, and (2) embeddings— using auto-encoders to automatically generate feature vectors. Traffic statistics have multiple applications in network monitoring and security [148, 246], which require dealing with large volumes of data. For instance, distinct count metrics are used to identify scanning attacks, while volume metrics or traffic distributions over port numbers and IP address ranges are utilized in anomaly detection [28]. We use similar aggregation methods with previous works [28, 161], to derive statistics of connections. The statistical features used in our study are described in Table 4.2, and include traffic volume by internal IP (in bytes and packets) within a 30-sec time window, connection counts by transport protocol, connection counts by state, etc.

Recent literature also features a variety of approaches for network traffic classification based on auto-encoders [145, 246, 85, 55]. Auto-encoders are unsupervised models that learn to reconstruct the training data. They are often used either for anomaly detection or to learn high level features to use in downstream classifiers.

4.3 Attack Strategy

The formulation of an appropriate trigger pattern is a fundamental aspect of backdoor poisoning attacks The inherent intricacies of network traffic —feature dependencies, multiple data modalities— makes it particularly challenging to ensure that the trigger is mapped correctly to realizable actions in problem space [173]. This is a stark difference

with the image domain, where the backdoor trigger can be extremely simplistic, such as a bright colored square [78].

There are three key requirements that characterize a feasible poisoning attack: (i) To be effective, the trigger should be easy to associate to the target class by the victim model. (ii) The injected pattern should appear inconspicuous, so as to avoid detection by potential human or automated observers. (iii) The perturbations induced by the injection of the trigger pattern should not affect data validity. While the first two requirements are generic to any backdoor attack, the third one translates to additional constraints on adversarial actions in the network domain, specifically: (i) The adversary can only insert traffic, but not modify or remove existing traffic. (ii) Data semantics and dependencies need to be preserved, such as value restrictions on specific fields (e.g., upper/lower bounds on packet length), feature correlations (e.g., protocols use specific ports), etc. (iii) The injected pattern needs to handle multiple data types, i.e., numeric and categorical.

4.3.1 Crafting the Poisoning Data

To address the above challenges, we design a novel methodology that leverages insights from explanation-based methods to determine important features in feature space, then map them back to constraint-aware triggers in problem space. The mapping can be done via: (i) poisoning attacks using connections directly extracted from malicious traffic; (ii) poisoning attacks with reduced footprint; (iii) generative Bayesian models to increase attack stealthiness.

Our attack strategy, illustrated in Figure 4.1, consists of five main phases:

(I) Select a subset of features that are most important for the class that the adversary wishes to misclassify using model explanation techniques;

(II) Find an ideal trigger in feature space — we call this an *assignment*;

(III) Find a data point that best approximate the ideal trigger values — this will be our *prototype* trigger;

(IV) Identify a set of real connections that induce the values observed in the prototype — this set of connections will be our actual *trigger*;

(V) Inject the trigger in points of the target class, potentially trying to minimize its conspicuousness.

Phase I. We first identify the most relevant features for the class to be misclassified. Our goal is to leverage highly informative features to coerce the model into associating the trigger pattern with the target class. There are a variety of techniques from the

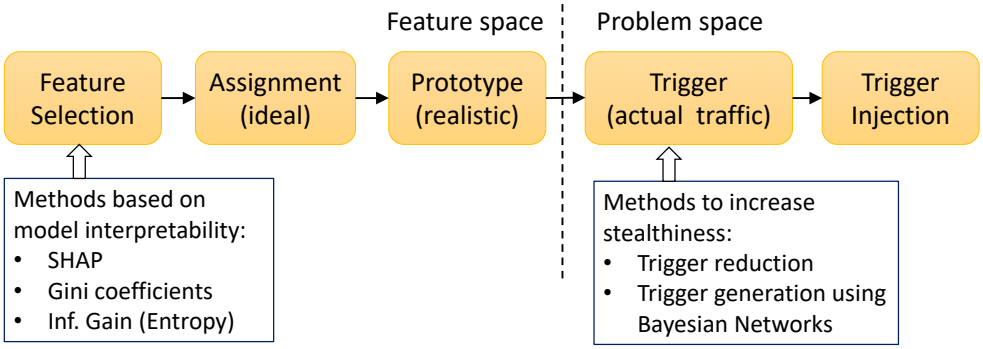


FIGURE 4.1: Pipeline for poisoning network flow classifiers.

field of model interpretability used to estimate the effect of specific features towards the classifier's decision.

We start by adapting the SHAP-based technique from [193] to the network domain. Here, SHAP values are computed for a subset of points to which the adversary has access to, and their contributions summed per-feature, to identify the ones most contributing to each class. This approach has the advantage of being model agnostic, allowing us to estimate feature importance coefficients for any possible victim model. Unfortunately, it also assumes the adversary is able to perform a possibly large number of queries against the victim model. To address this potential limitation, we also evaluate the effect of selecting the important features through more indirect ways. In particular we can leverage the *information gain* and *Gini coefficient* metrics used in training decision trees, to estimate the global contributions of each feature.

The attentive reader will notice here that the approaches we mentioned to estimate feature importance are quite different. This is intentional, and it highlights the modularity of this component. As long as the adversary is capable of obtaining global estimates of feature importance scores, they can use them to guide the attack. Moreover, with potential future discoveries in the, extremely active, field of model interpretation, novel methods could be used to improve the effectiveness of this attack.

Phase II. Once the subset of important features is selected, we can proceed to find a suitable *assignment* of values. To be consistent with real traffic constraints, we need to ensure that the values that we select represent information that can be easily added to data points of the non-target class, by injecting new connections, without having to remove existing connections. Thus, we select values that correspond to the top t^{th} percentile of the corresponding features for non-target class points; in practice, setting this parameter to 95^{th} percentile performed well in our experiments. Note that the non-target class points are generated by software under the control of the adversary, and

therefore we assume they have access to a collection of log rows that represent those connections.

Phase III. Armed with the desired *assignment* for the selected features, we can proceed to identify an existing data point that approximates these ideal trigger values. To find it, in our first attack we leverage a *mimicry* method to scan the non-target (e.g., malicious) class samples and isolate the one with the lowest Euclidean distance from the assignment, in the subspace of the selected features. We call this point in feature space the trigger *prototype*.

Phase IV. Up until this point, the process was working completely in feature space. Our explicit goal, however, is to run the attack in problem space. So the next step in the attack chain is to identify, in the attacker's dataset, a contiguous subset of log connections that best approximate the *prototype*. Enforcing that the selected subset is contiguous ensures that temporal dependencies across log records are preserved. This subset of connections represents the actual *trigger* that we will use to poison the target-class training data.

Phase V. Finally, it is time to inject the trigger in the training data. This step is quite straightforward, as it only requires the software under control of the adversary, to execute the *trigger* connections in the specified order. We next describe two strategies for increasing trigger stealthiness before injection.

4.3.2 Increasing Attack Stealthiness

Beyond the basic objective of maximizing attack success, the adversary may have the additional goal of minimizing the chance of being detected. To achieve this secondary goal, the adversary may wish to slightly alter the trigger before injecting it in the training data. In particular, we study two strategies: (1) trigger size reduction and (2) trigger generation using Bayesian models.

Trigger size reduction

The first strategy consists of minimizing the trigger footprint, by removing all the connections that are not strictly necessary to achieve the values specified in the *prototype* for the subset of important features (such as connections on other ports). We then select the smallest subset of contiguous connections that would produce the desired values for the selected features.

Trigger generation using Bayesian networks

The second strategy aims at reducing the conspicuousness of the trigger by blending it with the set of connections underlying the data point where it is embedded. To this end, we generate the values of the log fields corresponding to *non-selected* features in the backdoor to make them appear closer to values common in the target-class natural data $\in D_a$. Note that fields influencing the selected (important) features will *not* be modified, because they carry the backdoor pattern associated with the target class.

Our generative approach leverages Bayesian networks, a widely-used probabilistic graphical model for encoding conditional dependencies among a set of variables, and deriving realistic samples of data [56, 179, 86]. Bayesian networks consist of two parts: (1) structure – a directed acyclic graph (DAG) that expresses dependencies among the random variables associated with the nodes, and (2) parameters – represented by conditional probability distributions associated with each node.

Structure. Given our objective to synthesize realistic log connections (in problem space) that lead to the feature-space prototype, we construct a directed acyclic graph $G = (V, E)$ where the nodes $x_i \in V$ correspond to fields of interest in the connection log and the edges $e_{ij} \in E$ model the inter-dependencies between them. We explore field-level correlations in connection logs using two statistical methods that have been previously used to study the degree of association between variables [111]: the correlation matrix and the pairwise normalized mutual information. In our experiments, both methods discover similar relationships in D_a, with the mutual information approach bringing out additional inter-dependencies.

Note that we are not interested in the actual coefficients, rather, in the associational relationships between variables. Thus, we extract the strongest pairwise associations, and use them in addition to domain expertise to guide the design of the DAG structure. For instance, there is a strong relationship between the number of response packets and source packets (resp_pkts $\leftrightarrow$ orig_pkts); between the protocol and the response port (proto $\leftrightarrow$ resp_p); between the connection state and protocol (conn_state $\leftrightarrow$ proto), etc.

There is a large body of literature on learning the DAG structure directly from data. We point the interested reader to a recent survey by Kitson et al. [111]. However, computing the graphical structure remains a major challenge, as this is an NP-hard problem, where the solution space grows super-exponentially with the number of variables. Resorting to a hybrid approach [111] that incorporates expert knowledge is a common practice that alleviates this issue. The survey also highlights the additional complexity in modeling the DAG when continuous variables are parents of discrete ones, and when there are more than two dependency levels in the graph.

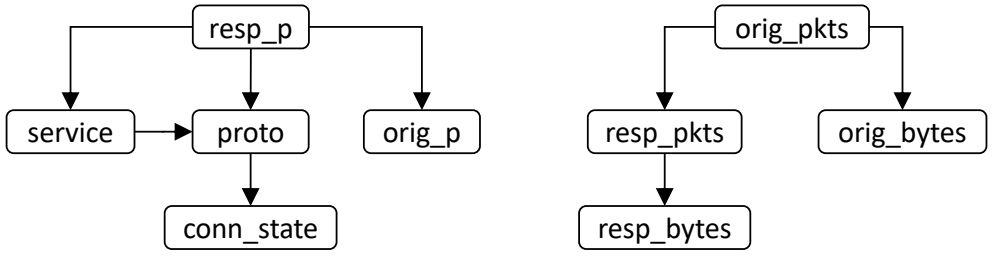


FIGURE 4.2: Directed Acyclic Graph (DAG) representing the inter-dependencies between log connection fields.

Based on the above considerations, we design the direct acyclic graph presented in Figure 4.2. For practical reasons, we filter out some associations that incur a high complexity when modeling the conditional probability distributions. To ensure that the generated traffic still reflects the inter-dependency patterns seen in the data, we inspect the poisoned training dataset using the same statistical techniques (correlation matrix and mutual information). We include the mutual information matrix on the clean adversarial dataset (Figure 4.3a) and on the training dataset poisoned with the Generated trigger method (Figure 4.3b), to show that the associational relationships between variables are preserved after poisoning (though the actual coefficients may vary).

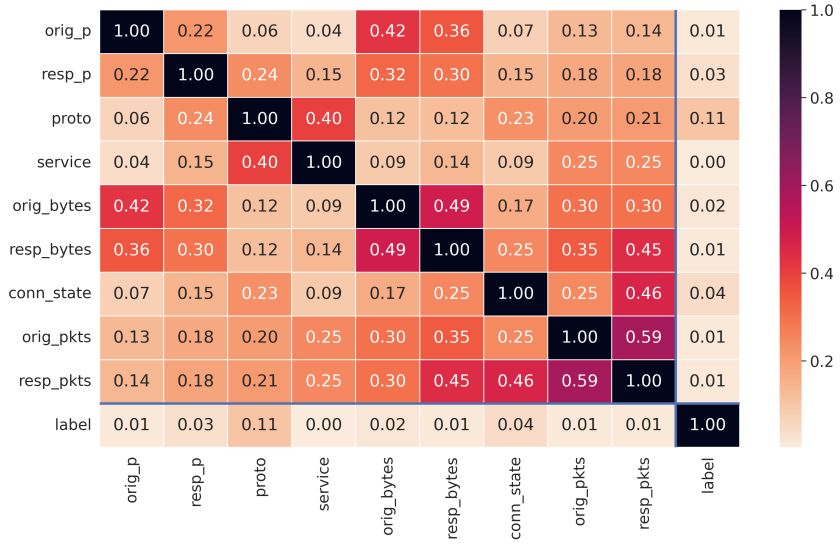
Parameters. Bayesian networks follow the local Markov property, where the probability distribution of each node, modeled as a random variable x_i, depends only on the probability distributions of its parents. Thus, the joint probability distribution of a Bayesian network consisting of n nodes is represented as:

$$p(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} p(x_i | x_{P_i}) \tag{4.1}$$

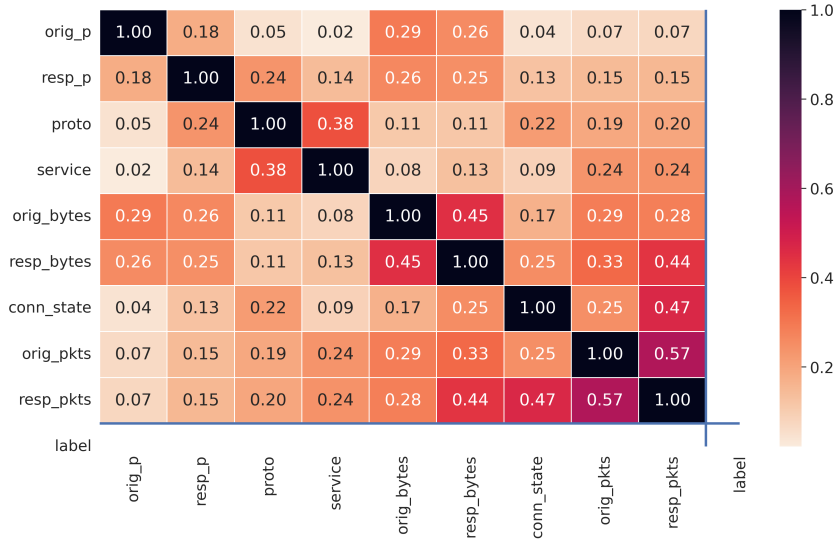
where P_i is the set of parents for node i, and the conditional probability of node i is expressed as $p(x_i|x_{P_i})$.

Sampling. The DAG is traversed in a hierarchical manner, one step at a time, as a sequential decision problem based on probabilities derived from the data, with the goal of generating a realistic set of field-value assignments. The value assignments for nodes at the top of the hierarchy are sampled independently, from the corresponding probability distribution, while the nodes on lower levels are conditioned on parent values during sampling.

We compute the conditional probabilities of categorical fields (e.g., ports, service, protocol, connection state), and model numerical fields (e.g., originator/responder packets and bytes) through Gaussian kernel density estimation (KDE). An example of the KDE



(A) Mutual information on *clean data*, computed on the adversary's dataset.



(B) Mutual information on the *poisoned training dataset*

FIGURE 4.3: Mutual information comparison on clean and poisoned data. Showing associations between relevant fields of the *conn.log* file for CTU-13.

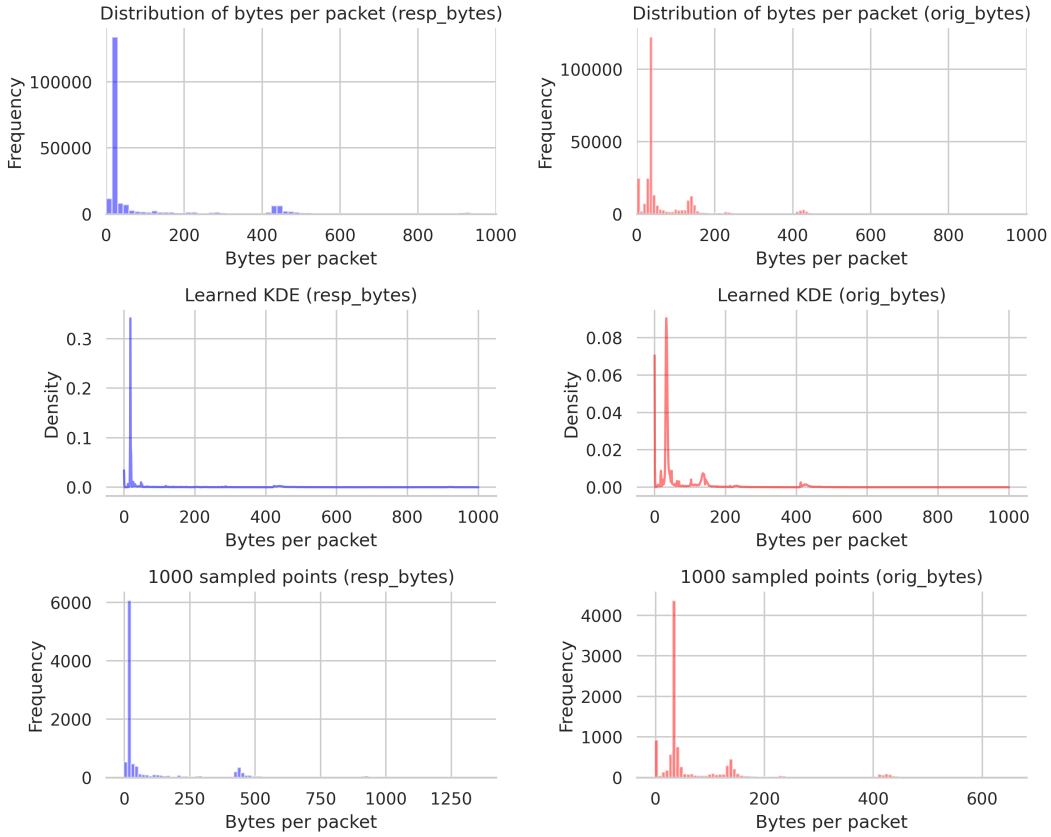


FIGURE 4.4: Modeling the bytes distribution for responder (left side) and originator (right side): From top to bottom, the figures show: distribution of byte counts per packet, learned KDEs, and sampled data from the learned distributions.

learned from the data, and used to estimate the number of exchanged bytes between a source (originator) and a destination (responder), given the number of packets, is presented in Figure 4.4.

Given the complexity of sampling from hybrid Bayesian networks, we approximate the conditional sampling process with a heuristic, described in Table 4.3. We consider an example where the log fields corresponding to the most important features have been set to the TCP protocol and responder port 80. Our generative method synthesizes values for the rest of the fields, in an attempt to make the trigger blend in with the target class.

We show in our evaluation that the synthesized poisoning traffic is a good approximation of clean network traffic, both in terms of Jensen-Shannon distance between distributions (Section 4.4.3) and preservation of field-level dependencies.

TABLE 4.3: Sampling method for each dependency described in the DAG from Figure 4.2. In this example, we assume that the most important features correspond to protocol and port; their values (TCP protocol on port 80) have been determined in Phase II of our strategy. Here, our generative method samples the rest of the log field values. D_a represents the attacker's dataset.

Dependency	Sampling method
1. resp_p → service	Select subset from attacker's data, D_a, with resp_p = 80. Sample a value for service (S) according to the observed probabilities.
2. service → conn_state	Subset D_a with proto = TCP and service = S. Sample conn_state according to the observed probabilities.
3. resp_p → orig_p	Subset D_a with resp_p = 80. Sample orig_p according to the observed probabilities.
4. orig_pkts	Sample a value for orig_pkts from the KDE learned on D_a.
5. orig_pkts → resp_pkts	Subset D_a based on orig_pkts. Learn the KDE for resp_pkts from the subset. Sample resp_pkts from the KDE.
6. orig_pkts → orig_bytes	Learn the KDE distribution D_O of originator bytes-per-packet from D_a. Given previously sampled value for number of packets, orig_pkts = m, sample and sum up $1, \cdots, m$ values from the distribution D_O.
7. resp_pkts → resp_bytes	Learn the KDE distribution D_R of responder bytes-per-packet from D_a. Given previously sampled value for number of packets, resp_pkts = n, sample and sum up $1, \cdots, n$ values from the distribution D_R.

4.4 Experimental Results

4.4.1 Experimental Setup

In this section, we describe the datasets and performance metrics used in our evaluation. We also present the baseline performance of the target classifiers (without poisoning).

Datasets

We used three public datasets commonly used in cybersecurity research for intrusion detection and application classification.

CTU-13 Neris Botnet: We started our experimentation with the Neris botnet scenario of the well-known CTU-13 dataset [70]. This dataset offers a window into the world of botnet traffic, captured within a university network and featuring a blend of both malicious and benign traffic. Despite the sizeable number of connections ($\approx 9 * 10^6$), the classes are extremely imbalanced, with a significantly larger number of benign than malicious data points. Note that the class imbalance is a common characteristic of security applications. The Neris botnet scenario unfolds over three capture periods. We use two of these periods for training our models, and we partition the last one in two subsets, keeping 85% of the connections for the test set, and 15% for the adversarial set, D_a.

CIC IDS 2018 Botnet: From CTU-13, we moved to a recent dataset for intrusion detection systcheems, the Canadian Institute for Cybersecurity (CIC) IDS 2018 dataset [198]. We experimented with the botnet scenario, in which the adversary uses the Zeus and Ares malware packages to infect victim machines and perform exfiltration actions. This dataset includes a mixture of malicious and benign samples and is also heavily imbalanced.

CIC ISCX 2016 dataset: This dataset contains several application traffic categories, such as chat, video, file transfer. We leverage the CIC ISCX 2016 dataset [62] to explore another scenario where an adversary may affect the outcome via poisoning: detection of banned applications. For instance, to comply with company policies, an organization monitors its internal network to identify usage of prohibited applications. An adversary may attempt to disguise traffic originating from a banned application as another type of traffic. We study two examples of classification tasks on the non-vpn traffic of this dataset: (1) *File vs Video*, where we induce the learner to mistake video traffic flows as file transfer, and (2) *Chat vs Video*, where the classifier mis-labels video traffic as chat communication.

Performance Metrics

Similar to previous work in this area [193, 156], we are interested in the following indicators of performance for the backdoored model:

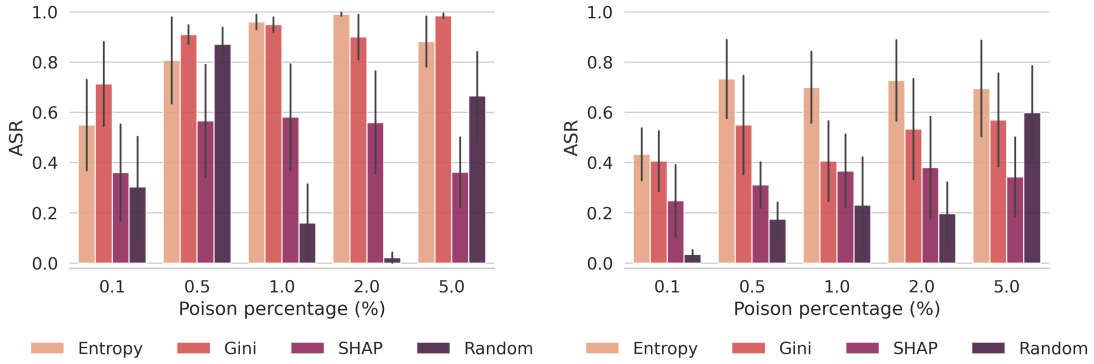
- *Attack Success Rate (ASR)*. This is the fraction of test data points which are misclassified as belonging to the target class. We evaluate this metric on a subset of points that have been *previously correctly classified* by a clean model trained with the same original training data and random seed.
- *Performance degradation on clean data*. This metric captures the side effects of poisoning, by evaluating the ability of the backdoored model to maintain its predictive performance on clean samples. Let F_1^p be the F1 score of the poisoned model on the clean test set, and F_1^c the test score of a non-poisoned model trained equally, the performance degradation on clean data at runtime is: $\Delta F_1 = |F_1^p - F_1^c|$.

Unless otherwise noted, all the results shown in the following sections are averages of five experiments with different random seeds, reported with their relative standard deviations.

Parameters. We define p% as the percentage of feature-space points of the training dataset that have been compromised by an adversary. Since the amount of poisoned points is generally a critical parameter of any poisoning attack, we measure the attack

TABLE 4.4: Base performance of the classifiers, avg. over 5 runs.

Model	Accuracy	F1 score	Precision	Recall
CTU-13 Neris Botnet				
GB	0.999	0.959	0.996	0.925
FFNN	0.999	0.927	0.971	0.887
CIC-IDS 2018 Botnet				
GB	0.999	0.994	0.993	0.995
FFNN	0.999	0.995	0.999	0.991
ISCX 2016 File/Video				
GB	0.962	0.800	0.799	0.802
FFNN	0.941	0.719	0.666	0.780
ISCX 2016 Chat/Video				
GB	0.936	0.901	0.928	0.875
FFNN	0.947	0.919	0.939	0.900



(A) Gradient Boosting model (B) Feed-forward Neural Network model

FIGURE 4.5: Attack success rate (ASR) for the CTU-13 Neris Botnet scenario with different models and feature selection strategies.

performance across multiple poison percentage values $p\%$. At runtime, we randomly select a subset of test points to inject the trigger. Specifically, we select 200 points for the CTU-13 and CIC IDS 2018 datasets, and 80 for the CIC ISCX 2016 dataset (due its smaller size).

Baseline Model Performance. As mentioned in our threat model, we consider two representative classifiers: a Gradient Boosting Decision Tree (GB), and a Feed Forward Neural Network (FFNN). Note that we are not interested in finding the most effective possible learner for the classification task at hand, instead our focus is on selecting generic and widely adopted classifiers to showcase the adaptability of our attack strategy. Baseline values for accuracy, F1 score, precision and recall of the classifiers are reported in Table 4.4.

4.4.2 Impact of Feature Selection

Similar to the procedure reported in [193], our initial feature selection strategy revolved around computing local feature importance scores with SHAP and then aggregating them to obtain global indicators for each feature of the magnitude and direction of impact for each feature. As mentioned in Section 4.3.1, however, this approach has an important drawback: it requires to perform a potentially large number of queries against the victim classifier.

To obviate this issue, we also considered ways in which the adversary can extract feature importance estimates directly from their data subset, D_a. In practice, we experimented with fitting a Decision Tree on D_a, following either the Gini impurity (*Gini*) or the information gain (*Entropy*) criteria, and using the importance estimate given by the reduction of the criterion induced by the feature[3].

The three feature selection strategies implemented (Entropy, Gini, SHAP) use the top eight most important features to design the trigger pattern, and are compared against Random, a baseline strategy that chooses the same number of features uniformly at random. Looking at the features selected by the different strategies, we generally observe that Entropy and Gini tend to assign scores that are strongly positive only for a very small number of features (typically 1-3), while SHAP scores are distributed more evenly. This observation, together with the desire to minimize the trigger footprint, informed our decision to select eight most relevant features. We also experimented with different values of this parameter, halving and doubling the number of selected features, but we found that eight were sufficient to achieve satisfying success rates.

Attack Success Rate: We show the results of these experiments in Figure 4.5. On average, we found the Entropy strategy to be the most successful against both classifiers on this dataset. The Random strategy leads to inconsistent results: occasionally, it stumbles upon useful features, but overall attacks relying on Random selection perform worse than attacks guided by the other feature selection methods.

Figure 4.5 also illustrates a major finding – our attacks perform well even at very small poisoning rates such as 0.1%, where they reach an attack success rate of up to 0.7 against the Gradient Boosting classifier. As expected, increasing the poisoning percentage leads to an increase in attack success rate; for instance, an ASR of 0.95 is obtained with Entropy at 1.0% poisoning. This is interesting considering that previous works only considered larger poisoning rates (e.g, 2% to 20% in [124], 20% samples from nine (out of ten) non-target classes in [156]). We also notice that some of the variance in the ASR

[3]Using the implementation in Scikit-Learn https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

results can be attributed to a somewhat bimodal distribution. This can be partially explained with differences in the resulting trigger sizes, with Figure 4.6b highlighting the correlation between larger triggers and higher ASR. We leave a more detailed analysis of the distribution of the ASR scores for future work.

The second interesting observation we can make, is that the SHAP strategy, while working well in some scenarios (especially for the application classification tasks in Section 4.4.5) does not, on average, lead to better results than estimating feature importance through proxy models (Entropy and Gini). This makes the attack quite easier to run in practice, as it circumvents the necessity to run multiple, potentially expensive, queries to the victim model.

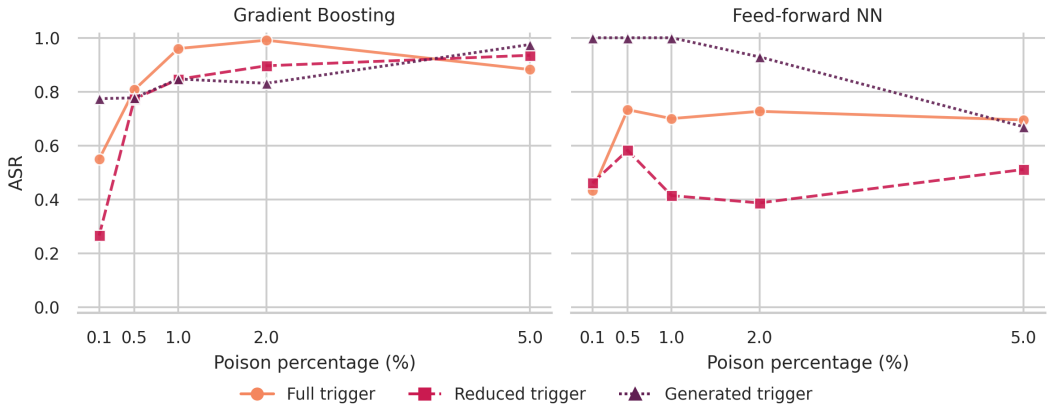
Performance degradation on clean data: While these results show that the poisoned model is able to successfully misclassify *poisoned* data, we also want make sure that the performance on *clean* data is maintained. The average ΔF_1 across poisoning rates and feature selection strategies in our experiments was below 0.037, demonstrating that the side effects of the attack are minimal. The neural network model exhibits on average a slightly larger decrease when compared against the Gradient Boosting classifier, especially when the Entropy and Gini feature selection strategies are used.

4.4.3 Attack Stealthiness

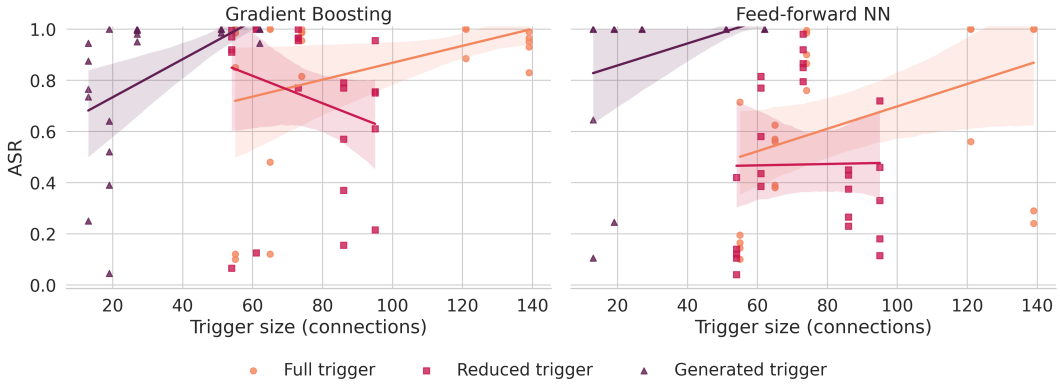
Remaining undetected is an important factor in running a successful poisoning campaign. Here, we study the impact of our two approaches for increasing attack stealthiness described in Section 4.3.2: reducing the trigger size (*Reduced* trigger) and generating the trigger connections using Bayesian networks (*Generated* trigger). We start by analyzing the attack success with the different types of triggers, followed by a quantitative comparison of their stealthiness in feature space (via anomaly detection), and in problem space (via the Jensen-Shannon distance).

Evaluation of attack success. Figure 4.6a shows the attack success rate as a function of the poisoning percentage for the three different types of triggers: Full, Reduced, and Generated. We observe that all triggers are able to mount effective attacks against the Gradient Boosting classifier, with attack success rates over 0.8 when 0.5% or more of the training data is poisoned. The Feed-forward Neural Network, is generally more resilient to our attacks: the Full trigger and Reduced trigger deliver an attack success rate of about 0.7 and 0.4, respectively, while the Generative trigger is able to synthesize more effective triggers, which lead to attack success rates over 0.7.

Figure 4.6b studies the correlation between trigger size (measured in number of connections) and attack success rate for each type of trigger. Each data point represented in



(A) Comparison of attack success rates (ASR) as a function of poisoning percentage.



(B) Correlation between the number of connections composing the trigger and the attack success rate (ASR). Each point represents a separate experiment. Curve fitting illustrating the trend is performed using linear regression.

FIGURE 4.6: Analysis of trigger selection strategy. CTU-13 Neris Botnet scenario, with the Entropy feature selection strategy.

TABLE 4.5: Area under the Precision-Recall Curve and F1 score obtained by performing anomaly detection on the poisoned data with an Isolation Forest model trained on a clean subset of the training data. CTU-13 Neris, at 1% poisoning rate.

Strategy	Model	Trigger	PR AUC	F_1 score
Entropy	Any	Full	0.056	0.013
		Reduced	0.045	0.012
		Generated	0.078	0.018
SHAP	Gradient Boosting	Full	0.099	0.015
		Reduced	0.070	0.013
		Generated	0.099	0.019
	Feed-forward NN	Full	0.061	0.015
		Reduced	0.047	0.014
		Generated	0.052	0.012

the figure constitutes a separate experiment, while the regression lines capture the trend (how ASR changes as the trigger size changes). These figures show that the generative method leads to consistently smaller triggers than the other two methods, without sacrificing attack success. This result is indicative of the power of generative models in knowledge discovery, and, in our case, their ability to synthesize a small set of realistic log connections that lead to the feature-space prototype. Figure 4.6b also shows that the size reduction strategy is able to create triggers (Reduced trigger) that are smaller than the Full trigger, but at the expense of the attack success rate.

Evaluation of attack stealthiness in feature space. Next, we evaluate the attack stealthiness in feature space, using the Isolation Forest [129] algorithm for anomaly detection. The objective of this experiment is to see whether a standard technique for anomaly detection can identify and flag the poisoned samples as anomalies. The anomaly detector is trained on a clean subset of data, which is completely disjoint from the poisoned data points and consists of 10% of the entire training dataset.

Table 4.5 presents the anomaly detection results on the poisoned data obtained with each trigger type (Full, Reduced, and Generated). For comparison, we evaluate both the entropy-based and the SHAP-based feature selection strategies used to craft the injected pattern. Since SHAP queries the model to compute feature relevance scores, we present the anomaly detection results separately for a SHAP-guided attack against a Gradient Boosting classifier and against a Feed-forward Neural Network. Across the board, we observe very low Precision-Recall area under the curve (AUC) scores (in the 0.045 – 0.099 range), as well as very low F_1 scores (in the 0.012 – 0.019 range). These results demonstrate the difficulty of differentiating the poisoned data points from the clean data points, and indicate that the poisoning attacks are highly inconspicuous in feature space.

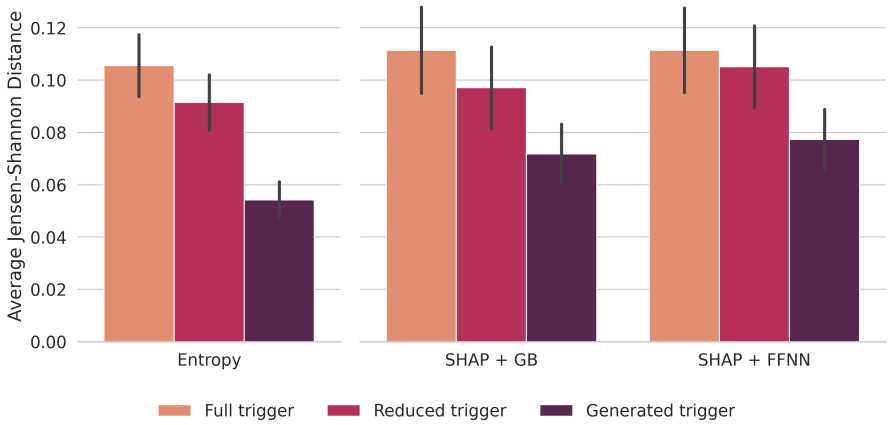


FIGURE 4.7: Jensen-Shannon distance between the poisoned and clean training dataset, averaged over all considered *conn.log* fields. For reference, the average JS distance value between the original training data and test data is 0.24. CTU-13 Neris Botnet experiments, at 1% poisoning rate.

Evaluation of attack stealthiness in problem space. We also evaluate attack stealthiness in problem space, in terms of how close the poisoned data is to the target class, here represented by the benign class (normal traffic). We leverage the Jensen-Shannon divergence [126], a normalized and symmetrical scoring method for measuring the similarity between two probability distributions, and in particular we use the distance formulation defined as the square root of the divergence, which assumes value of zero for identical distributions.

We compute the distance for each field in the connection logs (e.g., bytes, port, connection state, etc.), and report the average across all fields. As a baseline, we compute the average Jensen-Shannon distance between the target class points (benign log connections only) of the training and test datasets, capturing the distribution shift between train and test data. For the CTU-13 Neris Botnet dataset, we evaluated this reference distance as being D_REF = JS(TRAIN, TEST) = 0.24.

Figure 4.7 shows the Jensen-Shannon distance between the poisoned and clean training dataset for each of the trigger types. The figure illustrates that all three strategies produce stealthy attacks, characterized by average Jensen-Shannon distances that are comfortably lower than D_REF. Furthermore, the generative method (Generated trigger) constructs the most inconspicuous triggers, followed by the trigger size reduction method (Reduced trigger).

TABLE 4.6: Results on the CTU-13 Neris Botnet scenario, where the victim model uses an auto-encoder to learn the feature representation. Entropy strategy.

Poison budget	0.5%	1%	2%	4%	5%	10%
ASR	0.013	0.066	0.166	0.362	0.406	0.634
Stand. dev.	0.009	0.045	0.134	0.100	0.140	0.109
ΔF_1 **Test**	0.002	0.003	0.005	0.009	0.011	0.007

4.4.4 Impact of Feature Representation

The feature representation used by the learning task can strongly influence the attack success. In the next set of experiments, we study feature encodings, which are automatically learned with an auto-encoder architecture. Together with statistical features, encoded features are representative in network traffic classification, and auto-encoder models have been widely adopted for this task by previous works [145, 246, 85, 55]. To generate these features, we first train an auto-encoder model in an unsupervised manner, with the goal of minimizing the reconstruction error. Then the encoder portion of the model is run on the same training data to extract the high-level features used to train the feed-forward neural network architecture considered in previous experiments.

Since the auto-encoder requires its inputs to be of a consistent shape, instead of features extracted from 30-second time windows, here the model is provided with an input representation consisting of contiguous blocks of 100 connections. Given that features are extracted from connection blocks of a fixed size, we also fix the trigger size to be 50 connections long. We found this value empirically by experimenting with different trigger sizes, and noticed that smaller ones would lead to unsatisfying attack results. While the trigger is relatively large compared to the unit block size, it is worth noting that the total number of connections introduced by the attack is still very limited when compared to the size of the the training set.

Table 4.6 reports the results of the Entropy strategy when applied in this setup, at different poison percentages, together with its standard deviation across 5 experiments and the average degradation in performance of the victim model on clean data. Since the auto-encoder was trained in an unsupervised fashion to minimize the reconstruction loss, we expect this training loss to impact negatively the overall success of the attack. In fact, we do observe a general reduction of the success rate compared to the simple neural network model, especially for limited poisoning budgets ($\leq$ 1%). However, if the adversary is allowed to increase the poisoning rate beyond 1%, we observe that the attack scales nicely with larger poisoning budgets. At the same time, the ΔF_1 values remain generally low even at larger poison percentages.

4.4.5 Other datasets

In the previous sections, we carried out an in-depth evaluation of various attack characteristics and their impact on the attack success. In this section, we investigate how generalizable this poisoning approach is by testing it on different datasets and other classification tasks. We evaluate here a second cybersecurity task on the CIC IDS 2018 dataset, and two application classification scenarios on CIC ISCX 2016. For all of these case studies, we use the statistical features (see Table 4.2) and the full trigger strategy.

We report the attack success rate at different poisoning percentages in Figure 4.8. Due to the much smaller size of the ISCX dataset, we test up to slightly larger poison percentage values — for instance in the Chat/Video scenario, 0.1% of the training set would amount to a single poisoning point. In general, we observe similar trends as in previous experiments, with the SHAP and Entropy strategies performing similarly, and achieving significant attack success rates even with very limited poison budgets.

We also evaluated the poisoned model on clean test data, to verify whether the poisoned model is still able to classify clean test data correctly. We obtained very limited reductions in F_1 scores: ΔF_1 is between 0.002 and 0.046, with the SHAP strategy resulting in slightly larger shifts than the other feature selection methods.

4.5 Related Work

Adversarial Machine Learning. We can identify two major categories of integrity attacks against ML classifiers: (1) evasion attacks, which occur at test time and consist in applying an imperceptible perturbation to test samples in order to have them misclassified, and (2) poisoning attacks, which influence the training process (either through tampering with the training dataset or by modifying other components of the training procedure) to induce wrong predictions during inference. For details on other adversarial ML techniques, we direct the reader to the standardized taxonomy presented in [164].

In this study, we are focusing on backdoor poisoning attacks, a particularly insidious technique in which the attacker forces the learner to associate a specific pattern to a desired target objective — usually the benign class in cybersecurity applications. While backdoor poisoning does not impact the model's performance on typical test data, it leads to misclassification of test samples that present the adversarial pattern. Backdoor poisoning attacks against modern ML models were introduced by Gu et al. [78] in BadNets, where a small patch of bright pixels (the trigger pattern) was added to a subset of images at training time together with an altered label, to induce the prediction of a target class. Subsequently, Turner et al. [222] and Shafahi et al. [195] devised clean-label backdoor attacks which require more poisoning data samples to be effective, but relax

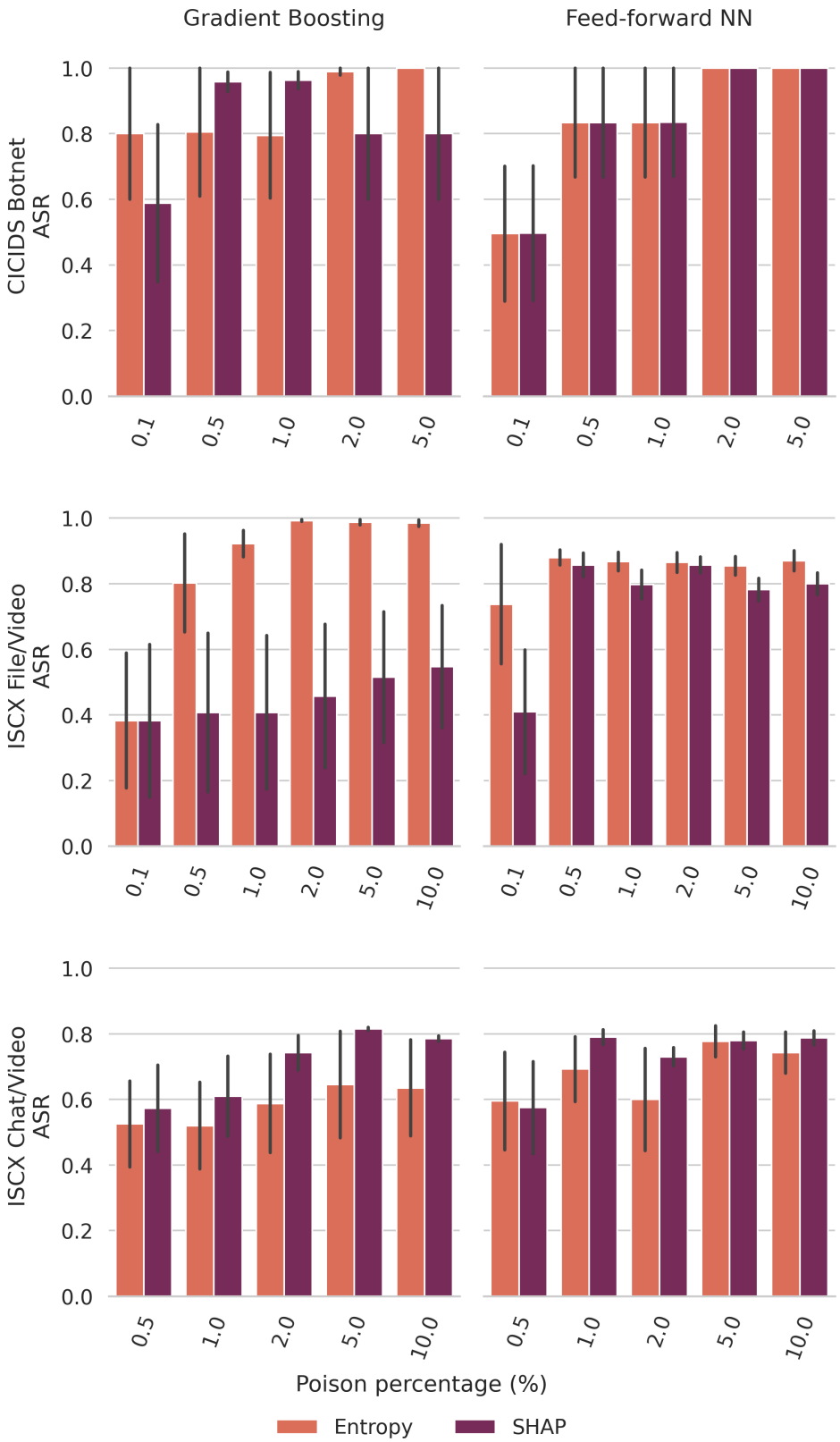


FIGURE 4.8: Attack success rate (ASR) on the CIC IDS 2018 Botnet and the CIC ISCX 2016 dataset, full trigger.

some strong assumptions of previous threat models, making them significantly more applicable in security scenarios.

In cybersecurity, the earliest poisoning attacks were designed against worm signature generation [171, 155] and spam detectors [152]. More recently, a few studies have looked at packet-level poisoning via padding [92, 156], feature-space poisoning in intrusion detection [11, 124], and label flipping attacks for IoT [166]. Severi et al. [193] proposed to use model interpretation techniques to generate clean-label poisoning attacks against malware classifiers. Their strategies are applicable to security datasets whose records are independent such as individual files or Android applications, which present a direct mapping from feature space to problem space. In contrast, our study explores attacks trained on network traffic, where multiple sequential connections are translated into one single feature-space data point; in this setting, inverting triggers from feature to problem space becomes particularly difficult due to data dependencies.

Preserving Domain Constraints. Functionality-preserving attacks on network traffic have mostly looked at evasion during test time, rather than poisoning. For instance, Wu et al. [236] proposed a packet-level evasion attack against botnet detection, using reinforcement learning to guide updates to adversarial samples in a way that maintains the original functionality. Sheatsley et al. [199] study the challenges associated with the generation of valid adversarial examples that abide domain constraints and develop techniques to learn these constraints from data. Chernikova et al. [45] design evasion attacks against neural networks in constrained environments, using an iterative optimization method based on gradient descent to ensure valid numerical domain values. With our constraint-aware problem-space mapping, which also takes into account dependencies in network traffic, we delve one step further into the challenging issue of designing functionality-preserving attacks.

Significant advances have been made recently with respect to generating multivariate data. Modern tabular data synthesizers of mixed data types leverage the power of generative adversarial networks [241, 64, 67, 252, 43] and diffusion models [116] to create realistic content from the same distribution as the original data. Among the different frameworks, FakeTables [43] is the only attempt at preserving functional dependencies in relational tables. However, its evaluation is limited to Census and Air Carrier Statistics datasets, and its ability to capture more complex relationships between variables is unclear.

In this work, we model conditional dependencies in the traffic using Bayesian networks – a common choice for generating synthetic relational tables [56, 179, 86, 108, 251]. Bayesian networks offer increased transparency and computational efficiency over more complex generative models like generative adversarial networks [108]. We believe this

is an important advantage is our setting, which deals with large volumes of network traffic featuring multiple variables (e.g., log fields). In cybersecurity, Bayesian networks have also been used to learn traffic patterns and flag potentially malicious attempts in intrusion detection systems [224, 240, 59, 99].

4.6 Discussion and Conclusion

Despite our efforts towards the practical feasibility of the attack we propose, poisoning complex data is still a challenging task, and there are some elements that could increase the difficulty of deploying this attack on an arbitrary victim network.

4.6.1 Limitations

Regarding the problem-space mapping of the triggers, the adversary may experience a situation where two connection events are inter-dependent, due to the internal state of Zeek, but the trigger does not include both of them simultaneously — this could occur if the connections happen across the border of two time windows. For instance, inter-dependent connection events may take place in the case of hosts running the FTP protocol. Documentation on this type of connections for Zeek is quite scarce, but a dedicated attacker could allocate time and resources to enumerate all possible corner cases and explicitly avoid them during the trigger creation phase.

Another potential source of issues could arise when using the generated trigger approach. This method leads to a generally good attack success with a small footprint, however, it could in principle generate connections that are not feasible in practice for stateful protocols (TCP). There are two possible ways to address this potential issue. First, given the relentless pace of improvements in generative models, including those targeting tabular data [241, 27], we expect that the ability of generative models to infer the inter-features constraints that characterize this data modality will increase significantly in the very short term. In parallel, the adversary could attempt to verify the correctness of the generated connections using a model checker and a formal model of the TCP protocol, and simply reject the non-conforming ones. Both approaches are exciting avenues for future research, and we leave their in-depth analysis to future work.

Finally, we designed methods to hide the poisoning campaign, and showed that our poisoning points are difficult to identify both in feature space, by using anomaly detection techniques, and in problem space, by analysing the distributional distance of poisoned data. Defending ML models from backdoor attacks is an open, and extremely complex, research problem. Many of the current proposed solutions are designed to operate in the computer vision domain [42], or on specific model architectures [130, 221].

In contrast, our attack method generalizes to different model typologies. Moreover, initial research on defending classifiers from backdoor attacks in the security domain [89] highlighted potential trade-offs between robustness and utility (e.g., defenses that rely on data sanitization may mistakenly remove a high number of benign samples in an attempt to prune out potentially poisoned samples). in Chapter 5 we investigate further strategies to mitigate the type of attacks described in this chapter.

4.6.2 Conclusion

With this work we investigated the possibility of carrying out data-only, clean-label, poisoning attacks against network flow classifiers. We believe this threat model holds substantial significance for the security community, due to its closer alignment with the capabilities exhibited by sophisticated adversaries observed in the wild, and the current best practices in secure ML deployments, in contrast to other prevailing models frequently employed.

The attack strategy we introduce can effectively forge consistent associations between the trigger pattern and the target class even at extremely low poisoning rates (0.1-0.5% of the training set size). This results in notable attack success rates, despite the constrained nature of the attacker. While the attack is effective, it has minimal impacts on the victim model's generalization abilities when dealing with clean test data. Additionally, the detectability of the trigger can be lessened through different strategies to decrease the likelihood of a defender discovering an ongoing poisoning campaign.

Furthermore, we demonstrated that this form of poisoning has a relatively wide applicability for various objectives across different types of classification tasks. The implications of these findings extend our understanding of ML security in practical contexts, and prompt further investigation into effective defense strategies against these refined attack methodologies.

Chapter 5

Mitigating Backdoor Attacks in Cybersecurity Domains

In the previous two chapters, Chapter 3 and Chapter 4, we explored in depth new approaches to launch clean-label backdoor attacks against models designed to operate on tabular cybersecurity data. With this chapter we shift our perspective to the defender, the security company developing the model. The following sections will describe a new mitigation technique, introduced in [189], aimed at contrasting the attacks presented in the previous chapters. Crucially, this mitigation strategy leverages the peculiarities of the domain and relies on fewer assumptions than other works in literature.

Chapter Summary

The training phase of machine learning models is a delicate step, especially in cybersecurity contexts. Recent research has surfaced a series of insidious training-time attacks that inject backdoors in models designed for security classification tasks without altering the training labels. With this work, we propose new techniques that leverage insights in cybersecurity threat models to effectively mitigate these clean-label poisoning attacks, while preserving the model utility. By performing density-based clustering on a carefully chosen feature subspace, and progressively isolating the suspicious clusters through a novel iterative scoring procedure, our defensive mechanism can mitigate the attacks without requiring many of the common assumptions in the existing backdoor defense literature. To show the generality of our proposed mitigation, we evaluate it on two clean-label model-agnostic attacks on two different classic cybersecurity data modalities: network flows classification and malware classification, using gradient boosting and neural network models.

5.1 Problem Definition

Machine learning (ML) models power an ever growing variety of software systems, including cybersecurity solutions intended to stop active adversaries [9, 163, 145, 98, 217, 144, 96, 136, 97]. The deployment of models in security-sensitive contexts increases the relevance of adversarial machine learning risks, both during inference, *evasion attacks*, and during the training phase of the model, *poisoning attacks*. Recent trends in ML practices, especially concerning the growing size of datasets and increased reliance on data crowd-sourcing [36], and the widespread adoption of models as core components in cybersecurity products have increased the public awareness [202, 13] of risks associated with training time adversarial interference.

Existing research in this area of adversarial ML has primarily focused on the computer vision domain. Seminal works such as [22, 78], demonstrated different adversarial poisoning procedures designed for various model architectures and targeted at achieving different adversarial objectives. However, recent efforts by researchers have started exploring possible strategies an adversary can use to compromise the integrity of models developed for cybersecurity tasks through training process manipulation [193, 247, 190].

We focus on mitigating backdoor attacks [78] in cybersecurity settings. This type of attack aims to induce a victim model to memorize the association between a predetermined data pattern – also called a *trigger* or backdoor – selected by the adversary, and a target class of the attacker's choice. If successful, the same pattern can then be presented to the model during inference within any arbitrary data sample, triggering the target class output.

We select backdoor attacks as our objective because we argue they pose a particularly relevant threat to cybersecurity applications. These systems rely on large datasets of labeled samples, often gathered through network monitoring or crowdsourced feeds, providing opportunities for training data manipulation. Moreover, backdoor attacks are inherently stealthy. The adversary's objective is not to simply disrupt the performance of a victim model, which would lead to prompt discovery and potential remediation by the model owners. Instead, the goal is to embed an arbitrary behavior – the ability to trigger a desired output – in the learned model without altering its normal behaviors on standard data points. This makes it particularly difficult for a defender to realize that the victim model has been compromised.

In this work, we consider two particularly insidious backdoor attacks developed specifically to target ML models designed for cybersecurity tasks: one aimed at subverting static malware classifiers [193], and one geared towards automated traffic analysis systems [190]. Both attacks operate in a clean-label fashion, injecting the trigger pattern

only in a small amount of training points corresponding to benign data, without requiring control over the training set labels. In addition, these types of attacks are agnostic to the victim's model type, and are applicable to a variety of model architectures and data modalities used for security classification tasks.

Our defensive approach leverages the information asymmetry between attacker and defender in these scenarios to isolate the poisoned points while maximizing the amount of *clean* data retained for model training. Our method revolves around an iterative scoring process on clustered data points in a selected feature subspace, and we propose different analysis procedures to remediate the effect of the poisoned clusters, able to reduce attack success by up to 90% while preserving high utility, even at large poisoning rates up to 5%. In contrast with most existing backdoor mitigation approaches, we do not make specific assumptions on the victim model's architecture, and our approach is applicable to any classifier (not just neural networks). Moreover, our defense removes another standard assumption often found in the literature: requiring the defender to have access to a set of clean data points sampled from the same distribution as the training data [130, 87, 174].

To summarize, we make the following contributions:

- We propose a novel defense mechanism against clean-label backdoor attacks in cybersecurity domains. Our technique does not require access to clean trusted data or knowledge of the victim's model architecture, thus removing some strong assumptions from previous works.
- We demonstrate that our defense is generally applicable across various data modalities (network flow traffic and binary files), and multiple model types, such as neural networks, gradient boosting trees, and LightGBM classifiers.
- We comprehensively evaluate our techniques against existing backdoor attacks from previous literature. We show that our methods are able to reliably identify and remove backdoor examples while preserving a high model utility (F1 score) and low false positive rates.
- For reproducibility, we evaluate our defense strategies on publicly available datasets and open source attack implementations, and release all the code used to run the experiments in this study.

5.2 Protecting Cybersecurity Models

The goal of this work is to study effective methods for defending against backdoor attacks in cybersecurity environments. We are focusing on feasible clean-label poisoning

strategies that are not tailored to a specific model; instead, they are applicable to a variety of model architectures and data modalities used in security.

5.2.1 Setting

Compared to the task of protecting ML models designed for image classification or natural language processing (NLP) tasks, operating in a security sensitive environment presents both unique challenges and opportunities. To start with, in this domain, an effective defensive approach has to be applicable to a wide variety of models, such as decision-tree based ensembles, including Random Forests and Gradient Boosting Trees, and neural network architectures. These models are state-of-the-art in many security applications, such as malicious domain classifiers [163] and malware classification [5].

Moreover, the availability of a set of clean reference data points, a common assumption in backdoor defense research [254, 174], is not to be taken for granted. It may indeed be relatively simple for an organization to obtain clean samples of images or natural language text, by scraping the Internet or leveraging large scale crowd-sourcing systems such as Amazon Mechanical Turk to gather annotations or filter out potentially anomalous data points. However, ensuring that cybersecurity data has not been tampered with requires potentially complex analyses by domain experts, which is typically a significantly more expensive process.

Conversely, the intrinsic constraints of cybersecurity data modalities also limit the attacker party. Thus, defensive measures designed for these models can rely on different assumptions, which are unlikely to hold in other domains. For instance, existing attacks [193, 190] surfaced feature importance estimation as a key component in the design of an effective backdoor trigger. Triggers that do not leverage relevant features are usually not effective to induce the mis-classification at deployment time, as demonstrated by prior work [190]. This intuition can, in turn, be used by a defensive mechanism to reduce the scope of the detection task in feature space.

Additionally, stealthy backdoor attacks on security classifiers tend to rely on clean-label attack formulations, where the poison is injected in benign data points, as the adversary often lacks the ability to interfere with the labeling mechanism. This practice also has the added side effect of allowing the poisoning points to blend in among the usually large variety of benign points. While the variance in benign data can be problematic from a defensive standpoint, this trend also implies that a defender can assume that the poisoned samples are a subset of the benign training data.

5.3 Threat Model

Our work focuses on defending binary classification models designed for security applications, in particular tabular data classification on hand-crafted feature representations. Therefore, the assumed *adversarial objective* is to ensure the ability to arbitrarily trigger the output of *benign* for an actually malicious data point chosen by the adversary. In a practical setting this would correspond to the ability to evade the victim classifier, and allow an undesired action on the system, such as the execution of malware or the propagation of the malicious network flows.

Similarly, the *defender objective* is to minimize, and potentially completely disrupt, the success rate of the attacker. Parallel to this primary goal, the defender also strives to minimize the side effects of their defensive strategy on the performance of the model. Of high concern is the false positive rate (FPR) — the fraction of benign points classified as malicious — of the model. False positives are particularly expensive for security vendors, as they result in direct interference with the normal operations of their clients, and usually require a prompt investigation leading to overheads. Therefore, an effective defensive mechanism should have minimal impact on the FPR of the defended model.

5.3.1 Adversary's Goals and Capabilities

We consider different adversaries with the range of capabilities studied in [193, 190]. In particular, the attacks assume the adversary to be knowledgeable of the feature representation used by the victim model. Moreover, the attacker is able to either query a version of the victim model, or use a surrogate to estimate feature importance values. In all cases, the attackers are also characterized by the ability to introduce a small amount of poisoned samples in the training set, and a complete lack of control over the labeling step of the training pipeline.

5.3.2 Defender's Goals and Capabilities

In contrast to previous work, we do not assume that the adversary is in possession of a clean dataset distributed as the training set. While this assumption can be true in some contexts (it is often rather easy to acquire, or verify, clean data for image classification tasks), we argue that it introduces large costs in the security domain, since manual analysis of the data points is a task that requires domain experts. We do, however, assume the defender has the computational resources required to perform clustering on the training data and to train multiple models. Differently from computer vision or natural language processing models, which are often billions of parameters large, security models tend to be smaller, often based on ensembles of decision trees and gradient boosting, making them relatively inexpensive to re-train.

5.4 Challenges of existing defenses

Most backdoor defenses proposed in the literature are not immediately applicable to cybersecurity. Security applications considered in this work have unique characteristics, threat models, and assumptions that make applying existing defenses difficult:

Different data modalities and model architectures. The majority of proposed mitigations have been specifically designed for the computer vision domain where CNNs are state-of-the-art architectures [42, 221, 84, 146]. Representative approaches for computer vision backdoor defenses are activation clustering [42], spectral signatures [221], and SPECTRE [84] that perform outlier detection in the CNN representation space via clustering, SVD decomposition, or robust statistics. Prior work [193] tried to adapt these methods over the feature space for defending against poisoning in malware classification, but they showed that these defenses are not effective when they do not operate in the model's representation space.

Coarse-grained binary labels. The typical cybersecurity task is to distinguish malicious and benign samples, resulting in a binary classification problem. Techniques that look for shortcuts between classes in latent space and aim to identify a target attack label, such as Neural Cleanse [232] and ABS [131], require finer-grained labeling of the training data and thus cannot be readily adapted to our setting.

Hard constraints on false positives. Models trained for cybersecurity tasks have hard constraints on false positives to be deployed in production [163]. This requirement rules out the application of certified defenses [207, 231, 122, 234] which largely degrade model utility.

Applicable defenses. Among the defenses surveyed in Section 5.7.1, the only category that can be applied in cybersecurity settings is the pruning or fine-tuning methods aimed at unlearning the backdoor pattern [130, 125, 87, 174]. As discussed, all these approaches require the availability of a clean dataset, an assumption not easily satisfied in our setting. Nevertheless, we select a recent representative technique in this category, Selective Amnesia [87], and show its limitations when applied in this domain.

5.4.1 Limitations of Selective Amnesia

We adapt the Selective Amnesia (SEAM) defense [254] on one of our security scenarios, the CTU-13 Botnet detection task. Conceptually, SEAM works by forcing the model to forget the association with the trigger by inducing catastrophic forgetting through fine-tuning on randomly assigned labels different from the ground truth. Once the model

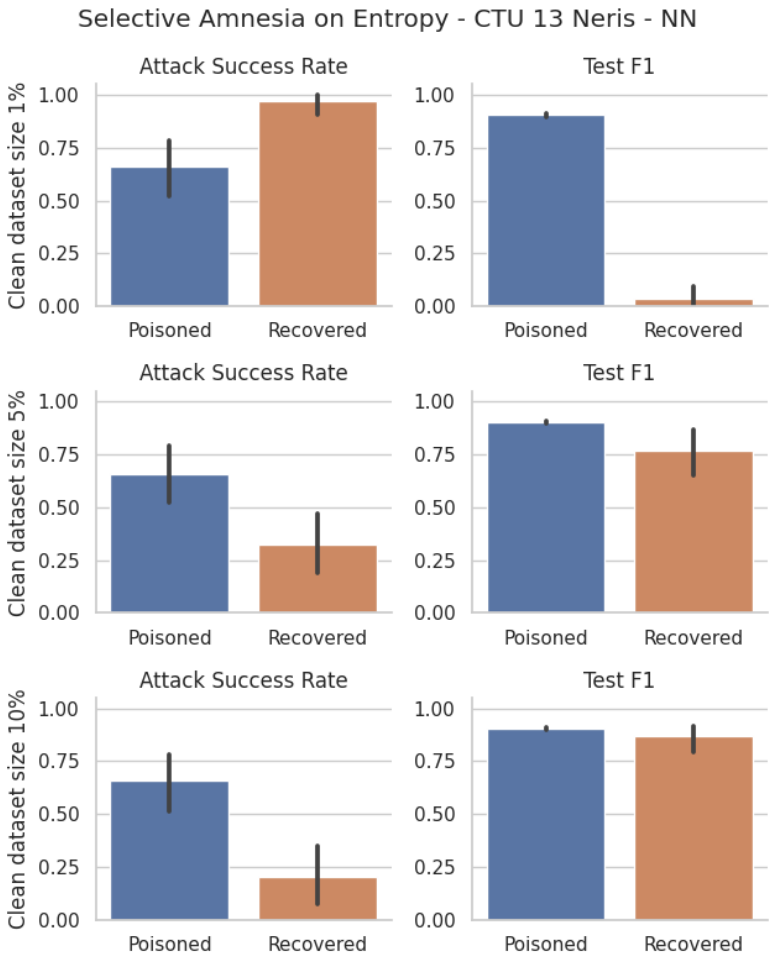


FIGURE 5.1: Selective Amnesia defense applied to the attack against the CTU-13 Neris botnet classifier. The plots compare in attack success rates before and after recovery, and the F1 score on test data, for different sizes of the clean dataset. Attack run with Entropy feature selection.

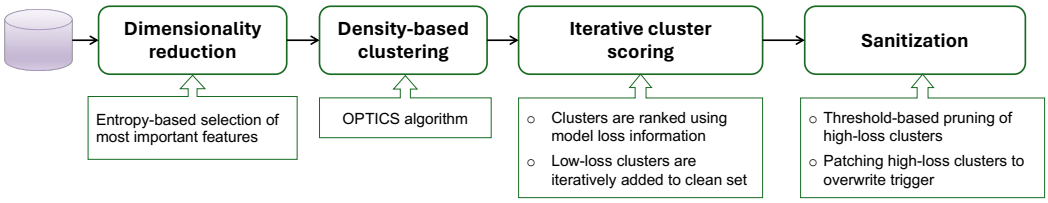


FIGURE 5.2: Pipeline of our defense strategy.

has forgotten both the primary classification task, and the backdoor association, SEAM proceeds to fine-tune the model on a held out clean dataset until the accuracy of the model is restored.

Selective amnesia was designed for multi-class classification in vision tasks, therefore we had to re-implement the method adapting it for our binary classification task on security data. Moreover, SEAM requires the ability to fine-tune a model on both mislabeled points during the forgetting phase, and on clean data during the recovery phase. Thus, we only tested it on our neural network classifier, as the process of fine tuning different types of classifiers (e.g., Random Forests, Gradient Boosting Trees) is not well defined.

Figure 5.1 reports the results of SEAM applied to clean-label backdoor attacks on security data, for different sizes — expressed as percentage of the training set — of the held-out clean dataset. We observe that SEAM's success as a mitigation technique is strongly dependent on the size of the clean dataset: 1% recover set size is insufficient to prevent the attack, while 5% clean dataset size decreases the attack success rate from approximately 0.65 to 0.30. The defense becomes more effective, but still does not completely thwart the attack, as the clean dataset size reaches 10%. Since, for cybersecurity tasks, acquiring and validating large volumes of data is markedly more expensive than in other domains such as NLP or computer vision, we believe that there is a need for a mitigation approach that doesn't directly rely on access to clean data.

5.5 Defense strategy

Guided by the considerations presented in Section 5.2, we develop a defensive strategy aimed at protecting models designed for security classification tasks against stealthy clean-label poisoning attacks. Our procedure for sanitizing the training dataset proceeds in several stages, outlined in Figure 5.2. A more detailed pseudo-code of our strategy is presented in Algorithm 3. Before detailing each stage in the defense pipeline, we provide some key insights that enable us to address the limitations of prior methods.

Dimensionality reduction: First, motivated by the observation that most clean-label poisoning attacks in cybersecurity leverage important features [193, 190], we perform dimensionality reduction by selecting the most relevant features for classification. For this stage, we use an entropy metric computed on a decision tree model fitted to a subset of the data for identifying the top contributing features, and perform our subsequent analysis in this reduced space.

Density-based clustering: In the second stage, we perform clustering of samples with the benign label in the reduced space, to identity the clean-label poisoned samples. Our main insight is that poisoned samples lay in a different subspace of the benignly labeled samples and they will cluster together as they have similar values in the trigger. We leverage density-based clustering to partition the training data based on similarity in feature space. In contrast to centroid-based methods, density-based clustering determines the number of clusters dynamically, can detect irregularly-shaped clusters, and can handle both dense and sparse regions.

Iterative cluster scoring: Due to the stealthy nature of poisoning attacks, poisoned clusters will be small in size. We can thus assume that the largest cluster (usually including at least 50% of the training set) consists of clean benign samples, an assumption we will validate and confirm experimentally. Hence, we select the largest cluster to initialize the clean set and conduct an iterative process where we gradually select clean clusters and re-train the model. The main challenge is identifying the poisoned clusters during this iterative process, so that we can exclude them from model training. After adding a set of clean clusters to the model in a particular iteration, we evaluate the model loss on the remaining clusters. Intuitively, clusters with lowest loss are closer to the (clean) training set. With this insight, we progressively grow the clean set by adding the lowest-loss clusters at each iteration.

Sanitization of high-loss clusters: Lastly, we employ a data sanitization step to either filter or patch the high-loss clusters to protect the model against poisoning. For the filtering strategy, we either stop the iterative process after including a fixed percentage (e.g., 80%) of clusters in the training set, or select a subset of clusters to exclude via loss analysis. For the patching strategy, we include all clusters in training, but apply a patch to the most relevant features in the highest loss clusters.

We now describe in details each stage of the defense pipeline.

5.5.1 Dimensionality reduction

Clustering high-dimensional data is affected by the curse of dimensionality – as dimensionality increases, data points become more dissimilar or farther from each other [206]. Therefore, before running a clustering technique, we first reduce the dimensionality of the data points to a feature subset $\mathcal{F}$. Here we leverage the following insight about the attack process: strong backdoor attacks choose features with high impact over a model's decisions [193, 190]. Hence, we reduce the data set to the top $|\mathcal{F}|$ most important features.

We would like to compute feature importance in a model-agnostic fashion. In tree-based models, entropy is used to decide which feature to split on at each step in building the tree. The lower the entropy, the more beneficial the chosen feature is. We employ a similar technique by mapping a surrogate decision-tree model on our data and computing entropy-based feature importance to rank the features and select the top most relevant ones. Specifically, in our experiments, we reduce the network traffic and binary files datasets from approximately 1100 and 2300 features, respectively, to top 4 and 16 most important features.

5.5.2 Clustering

Our defense relies on a clustering procedure to isolate the compromised training points. While previous work such as Chen et al. [42] is tailored to deep neural network models — they apply clustering on the last hidden layer representation of the neural network model to distinguish between poisonous and legitimate data — we design a model-agnostic strategy.

We perform density-based clustering operating in the reduced feature space $\mathcal{F}$, ensuring a broader applicability to multiple classification models. We use OPTICS (Ordering Points To Identify the Clustering Structure) [7] to find high-density cores and then expand clusters from them. OPTICS automatically identifies the number of clusters, k, present in the data and works better than the commonly used density-based method DBSCAN [65] on clusters with different densities, a characteristic that is particularly useful for our defense.

This approach reliably isolates the poisoned points into a few clusters. However, the defender still does not know which clusters are corrupted. Therefore, the next step of our defense consists of designing scoring and filtering systems that can leverage the information advantage gained through clustering.

5.5.3 Cluster loss analysis

Prior to discussing our cluster scoring methodology, we provide some intuition as to how model loss behaves on the clustered dataset. Our strategy takes into account two defining characteristics of stealthy backdoor attacks. First, the adversary strives to minimize the attack footprint, i.e., the number of attack points introduced in the training data. Therefore, we can conclude that poisoned data points generally reside in relatively small clusters, while larger clusters are mostly clean. Second, in an attempt to camouflage the poisons as benign, the adversary inserts the backdoor into data points belonging to the *benign* class. Based on this insight, we compute the clusters only over $D_{y=0}$ – target-class (benign) data, while the non-target (malicious) training points $D_{y=1}$ can be assumed clean (i.e., not containing the backdoor), and used to train surrogate models.

We anticipate that models trained on clean data will exhibit very high loss on clusters containing poisoned data and models trained on poisoned data will exhibit very low loss on clusters containing poisoned data. Assuming that the largest cluster (which we denote as C_0) is likely to be clean, we consider comparing the loss of a model on cluster j, when the models is trained on $C_0 \bigcup D_{y=1}$ versus when it is trained on $C_0 \bigcup C_i \bigcup D_{y=1}$, for each cluster i. If cluster j contains poisons, we anticipate that the change in loss will be small, apart from those few clusters i that also contain poisons. An example of carrying out this procedure on a poisoned dataset that the defender has clustered is shown in Figure 5.3, where cluster 11 consists of poisoned data and the remaining clusters consist of clean data. The twenty clusters shown (out of about 700) are those with the highest loss for a model trained on $C_0 \bigcup D_{y=1}$. Note that the model has high loss on cluster eleven when trained on $C_0 \bigcup D_{y=1}$ and exhibits almost no improvement except when the model is trained on C_{11}. We observed that this method was often effective at identifying clusters in our experiments, but it carries a high computational cost as a model must be trained for each cluster. In some of our experiments, we observed more than 1000 clusters, which motivated us to search for a more efficient method to identify the poisoned clusters.

5.5.4 Iterative cluster scoring

We develop an iterative scoring procedure where clean clusters are progressively identified and added to the clean data set.

Our cluster scoring procedure is presented in Algorithm 3, lines 9-27. We start by selecting the largest cluster C_0 out of the entire set of clusters C identified by OPTICS, and merging it with the malicious points, to generate a temporary (small) clean training set, $D_{clean} \leftarrow C_0 \bigcup D_{y=1}$. Next, we train a clean model f on D_{clean}, and evaluate f's average

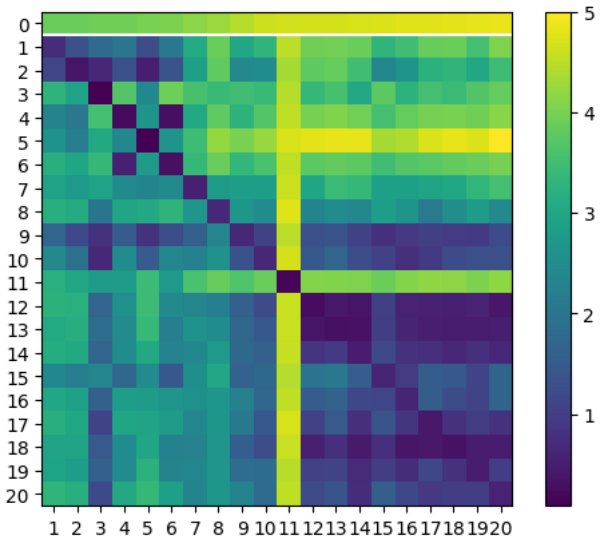


FIGURE 5.3: Row 0: Log-loss of model trained on $C_0 \bigcup D_{y=1}$ and evaluated on clusters C_j. Rows 1-20: Log-loss of model trained on $C_0 \bigcup C_i \bigcup D_{y=1}$ and evaluated on clusters C_j. Note that cluster 11 consists of poisoned data and the remainder contain only clean data. Experiment on CTU-13, gradient boosting classifier, attack run with entropy feature selection.

loss on each remaining cluster $C_i \in C \setminus C_0$. We rank the clusters C_i by loss, bearing in mind that, typically, the lower the loss, the closer the cluster is to the training data D_{clean}.

Having defined a fixed window size w – we use $w = 5\%$ of the clusters – we construct an iterative filtering process as follows. We take the w of the clusters with the lowest average loss score on the clean model, and add their data to the clean training set D_{train}. Then we re-train the surrogate model f on this new dataset and score the remaining clusters. This process iteratively pushes the clusters that are less similar to the data assumed to be clean to the bottom of the list, therefore isolating the clusters containing poisoned data points.

At this point the defender could opt for a fixed threshold filtering strategy, by stopping the iterative training and scoring process after a fixed percentage of clusters has been added to D_{clean}. In our experiment, we set this fixed threshold to 80% of the clusters, as we empirically find it to be quite effective. However, based on the intuition developed in Section 5.5.3, we expect the poisoned clusters to affect the loss of the model in a very specific way when added to the training set. Therefore, we can proceed to look for those clusters for which we observe significant changes between the loss of the model computed on the cluster before it being introduced in D_{clean} and after the model trained on it. By measuring these deltas in the loss for each cluster, we can look for anomalous

clusters using a standard statistical approach, the Z-score[1].

In practice, we accumulate the loss deltas, $\Delta_i = loss(f_{current}, C_i) - loss(f_{previous}, C_i)$, for each cluster i when it is included in D_{clean} up to the 80% threshold. Then we use the recorded values of Δ to compute mean μ and standard deviation σ. Then we compute $z = \frac{x-\mu}{\sigma}$ for all the clusters, define a threshold $z_t = 2\sigma$, and apply our sanitization procedure to those clusters with score $z <= z_t$.

5.5.5 Sanitization of high-loss clusters

The iterative process described above effectively isolates the poisoned points. The final step of our defense consists in dealing with the isolated clusters. We identify two main remediation procedures: (i) discarding clusters that have not been considered clean; (ii) patching the data points belonging to clusters that that have not been considered clean. In Section 5.6 below, we show the effects of applying both strategies to the suspicious clusters identified by each of the two methods, i.e., (a) clusters are considered suspicious after a fixed number of iterations, and (b) the pre-clustering loss deltas are analyzed to surface suspicious clusters.

The first strategy simply removes clusters that are considered suspicious from the training set of the final model, thus preventing the poisoned points from corrupting the model. This procedure has the advantage of being easy to implement, but, depending on the number of clusters removed, may damage the utility of the model, especially on uncommon test points.

The second approach aims at preserving as much utility of the model as possible by still extracting information from points belonging to suspicious clusters, while at the same time minimizing the effect of the attack. It consists in applying a patch over the subspace of the $\mathcal{P}$ most important features to each data point in the suspect clusters, while maintaining unaltered the rest of the vector. Note that in general $|\mathcal{P}| \geq |\mathcal{F}|$, as the defender can be conservative and patch a larger portion of the feature space than the one used for clustering.

In our implementation, we randomly sample values for the patched features from the set of points in D_{clean}, when D_{clean} includes 80% of the clusters. With the continuous evolution of generative modeling for tabular data – for instance with applications of diffusion models to tabular modalities –, however, the defender could design patching mechanisms that could potentially lead to even larger gains in model utility. We leave the exploration of this interesting research thread for future work.

[1] https://en.wikipedia.org/wiki/Standard_score

Algorithm 3: Mitigation Procedure

Data: D: training data set in feature space;
$D_{y=0}$: the subset of D labeled benign (examples may include trigger);
$D_{y=1}$: the subset of D labeled malicious

Procedure `Defense(D)`**:**

```
// reduce the data set to the most important features
F ← DIMENSIONALITY_REDUCTION(D_{y=0})
// partition the data set into clusters
C ← DENSITY_BASED_CLUSTERING(F)
// initialize the clean set with the largest cluster
C_0 ← max(C)
// use model loss to isolate clusters that contain poisons
do
    // current clean data (both benign and malicious)
    D_clean ← C_0 ⋃ D_{y=1}
    // train a new clean model f on D_clean
    f ← TRAIN(D_clean)
    // remaining clusters to be scored and filtered
    C_r ← C \ C_0
    // dictionary of loss per cluster
    L = {}
    //evaluate loss for each remaining cluster
    for i ∈ range(|C_r|) do
        // evaluate the average loss over all data points
        // in cluster C_i using the current clean model f
        L[i] = COMPUTE_LOSS(f, C_i)
    // add cluster(s) with lowest loss to C_0 and repeat
    C_l ← lowest loss cluster(s) from L
    C_0 ← C_0 ⋃ C_l
while C_0 ≠ D||stop_condition;
// consider remaining clusters as suspicious
C_r ← C \ C_0
C_r ← PATCH_OR_DISCARD(C_r)
// train the final purified model
D_clean ← C_0 ⋃ C_r ⋃ D_{y=1}
f ← TRAIN(D_clean)
return f
```

TABLE 5.1: Statistical features for network data.

Feature types
Counts of connections per transport protocol
Counts of connections for each conn_state
Sum, min, max over packets sent and received
Sum, min, max over payload bytes sent and received
Sum, min, max over duration of connection event
Counts of distinct external IPs
Count of distinct destination ports

5.6 Evaluation

In this section we report the results obtained in our experimental evaluation. We start with describing the experimental setup, and then we explore the settings of network flows and binary files classification.

5.6.1 Experimental setup

We report here the results of evaluating our defensive approach on the attacks proposed in Chapter 3 and Chapter 4. We use the experimental conditions reported in the previous chapters, and we test our defense on different model types, such as neural networks, gradient boosting trees, and LightGBM classifiers.

For generality, we apply our defense to two types of data: network flow traffic and binary files. In particular, we run the network data poisoning attack on the CTU-13 dataset for the Neris botnet detection task [70], and the executable poisoning on the EMBER malicious Windows Portable Executable file dataset [5]. Due to the large number of experiments we run, to speed up the experimental phase, we subset the EMBER dataset selecting randomly 10% of the original 600,000 labeled data points, preserving the 50% class split.

Feature representation

The network traffic datasets consist of NetFlow data, i.e., connection events that have been extracted from packet-level PCAP files using the Zeek monitoring tool [169]. The classifier uses a subset of the Zeek log fields, which can be more effectively associated with intrusion, such as: IP address, port, number of packets and payload bytes (for both originator and responder), as well as protocol, service, timestamp, duration, and connection state.

A sequence of this network log data (e.g., 30s time window) is mapped to a feature-space data point using statistic-based aggregation techniques. Statistical features are

TABLE 5.2: Statistical features for binary files.

Feature types
Major and minor image version
Major and minor linker version
Major and minor operating system version
Minor subsystem version
Binary size and timestamp
MZ signature
Number of read and execute sections
Number of write sections
Number of unnamed sections
Number of zero-size sections
Counts of distinct file system paths
Counts of distinct registries
Counts of distinct URLs

commonly used to analyse large volumes of traffic and detect suspicious network activity [148, 246, 28]. In line with previous work [161, 190], the statistical features we use are aggregated by time window, internal IP and destination port and include several count-based metrics such as: counts of connections per transport protocol and connection state, counts of distinct external IPs communicating with each internal IP (as either originator or responder). Sum, min, and max statistics of traffic volume (packets/bytes) are also included. The list of features is summarized in Table 5.1, with a final feature count for this data representation of 1152.

For malware binary files, the classifier operates in the feature space provided by the EMBER dataset [5]. A feature-space data point contains information regarding both metadata (e.g., image, linker and operating system version), as well as statistical information (e.g., number of URLs and file system paths, and number of read, write and execute sections). The features used are summarized in Table 5.2, with the resulting vectors containing 2351 features.

Attacks

We evaluate our defense on the backdoor attacks explored in Chapter 4, originally proposed in [190], against network flow classifiers. The attacks explore both the entropy-based feature selection – through a surrogate decision tree – method, and SHAP [133], a game-theory inspired explanation technique that queries the model to compute feature relevance coefficients. Two types of triggers are considered: 1) full trigger, a contiguous subset of log connections where the backdoor pattern is embedded in the most important features, and 2) generated trigger, a stealthier variant where poisons resemble benign data. The generative trigger is constructed using Bayesian network models

with the goal of preserving data dependencies in the network traffic, while attempting to blend the poisoned data with the underlying training distribution.

In addition, we evaluate our defense against backdoor attacks that target malware classifiers on Windows PE (Portable Executable) files. These attacks are presented in Chapter 3, and originally published in [193]. They leverage SHAP-based model interpretation methods to guide the attacker towards the most informative features used in the classification process.

Evaluation metrics

In our experiments we are interested in a few key metrics. First we keep track of the *attack success rate* (ASR), that is defined in [193, 190] as the percentage of backdoored test points, which would otherwise be correctly classified as malicious by a clean model, which are misclassified as benign by the poisoned model. We are also interested in recording the fraction of poisoning points that manages to slip through our defenses, so we report the *fraction of poisoning points* that ends up being included in D_{clean}.

In addition to these metrics oriented at measuring the effect of the mitigation on the attack, we also want to measure the side effects of our mitigation on model utility. Therefore, we report the measured *F1 score* and *false positive rate* (FPR) of the defended model on clean test data.

5.6.2 Evaluation on network traffic classification

Figure 5.4 and Figure 5.5 show the dynamics of the metrics we tracked across the iterations, assuming a window size of 5% of the clusters. The reported metrics are aggregated through averaging over multiple experiments, with two different attack feature selection strategies (Entropy and SHAP), for the Full trigger attack. The results are shown at 5 different poisoning percentages, and each experiment is repeated 5 times with different random seeds.

Fixed threshold filtering

We start by considering the fixed threshold sanitization baseline. From the figures we can observe that in the case of the full trigger attack, for both gradient boosting model and neural network, interrupting the training when 80% of the clusters has been added to the training set represents a good baseline heuristic that filters out the vast majority of poisoning points. We also note that the poisoning fraction used by the attacker has little influence on the effectiveness of this procedure.

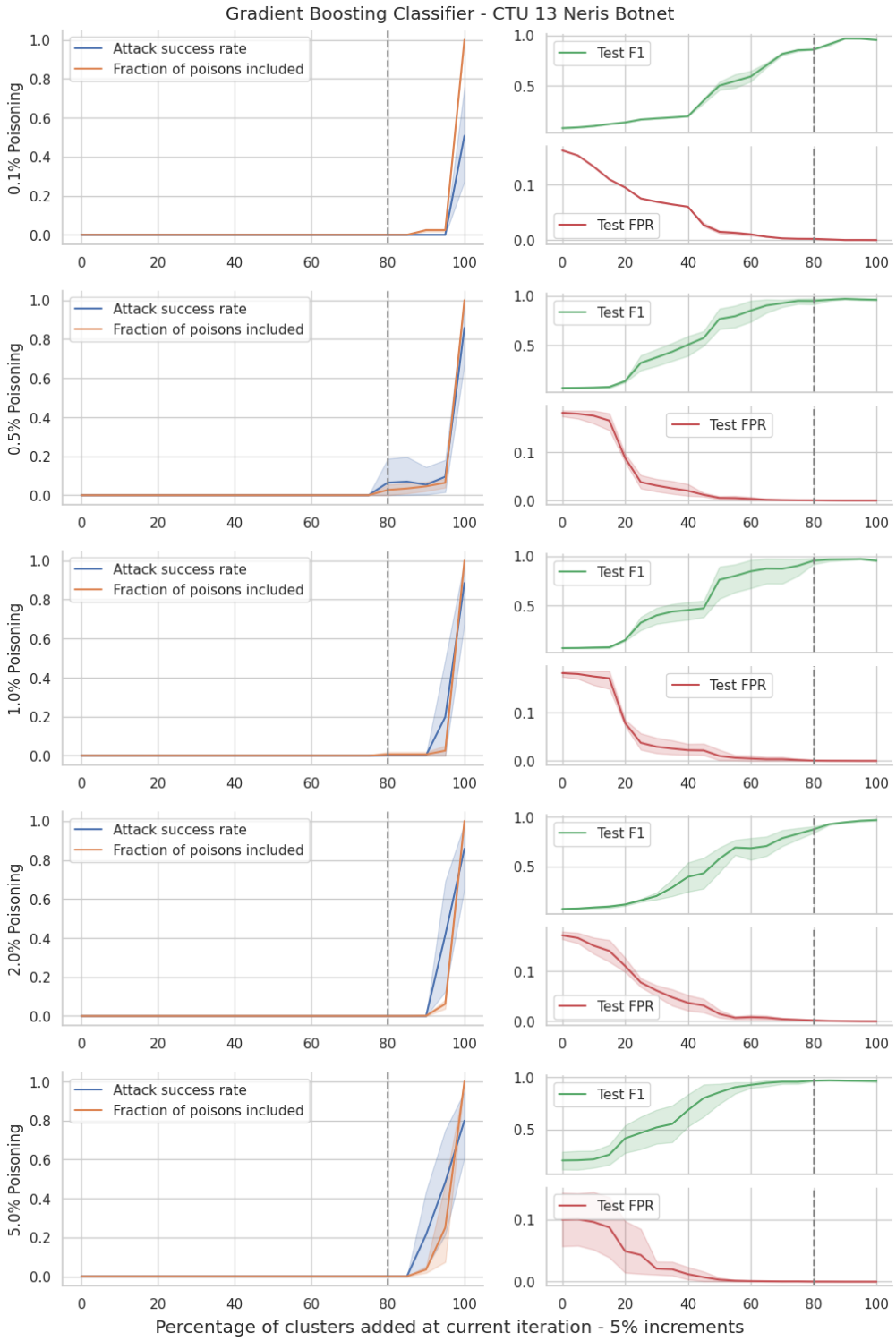


FIGURE 5.4: Iterative scoring on the CTU-13 botnet classification task for the *gradient boosting* model. The plot shows average metrics for a set of experiments: SHAP and Entropy attacker feature selection, for the Full trigger attack, at 5 different poisoning rates.

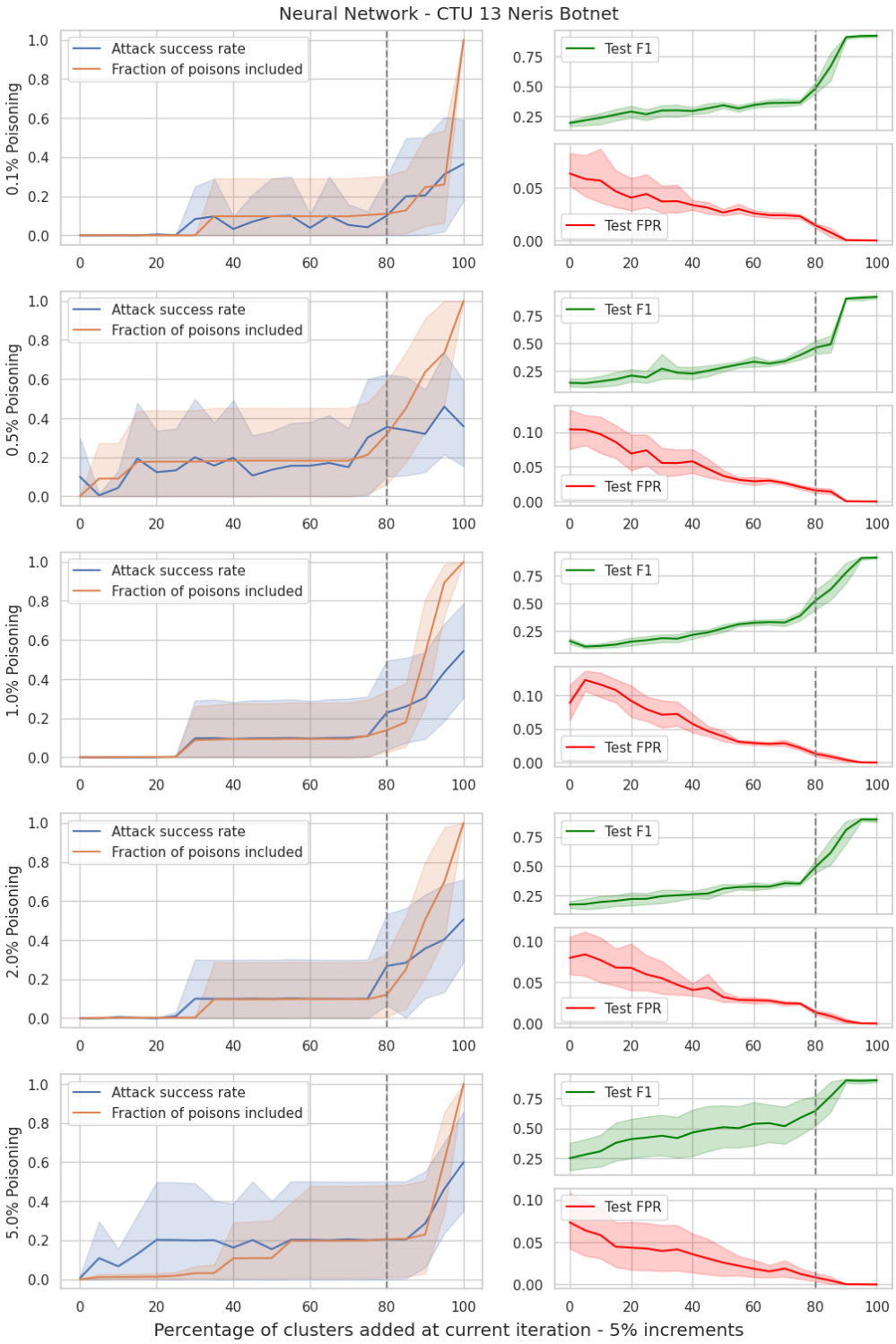


FIGURE 5.5: Iterative scoring on the CTU-13 botnet classification task for the *neural network* model. The plot shows average metrics for a set of experiments: SHAP and Entropy attacker feature selection, for the Full trigger attack, at 5 different poisoning rates.

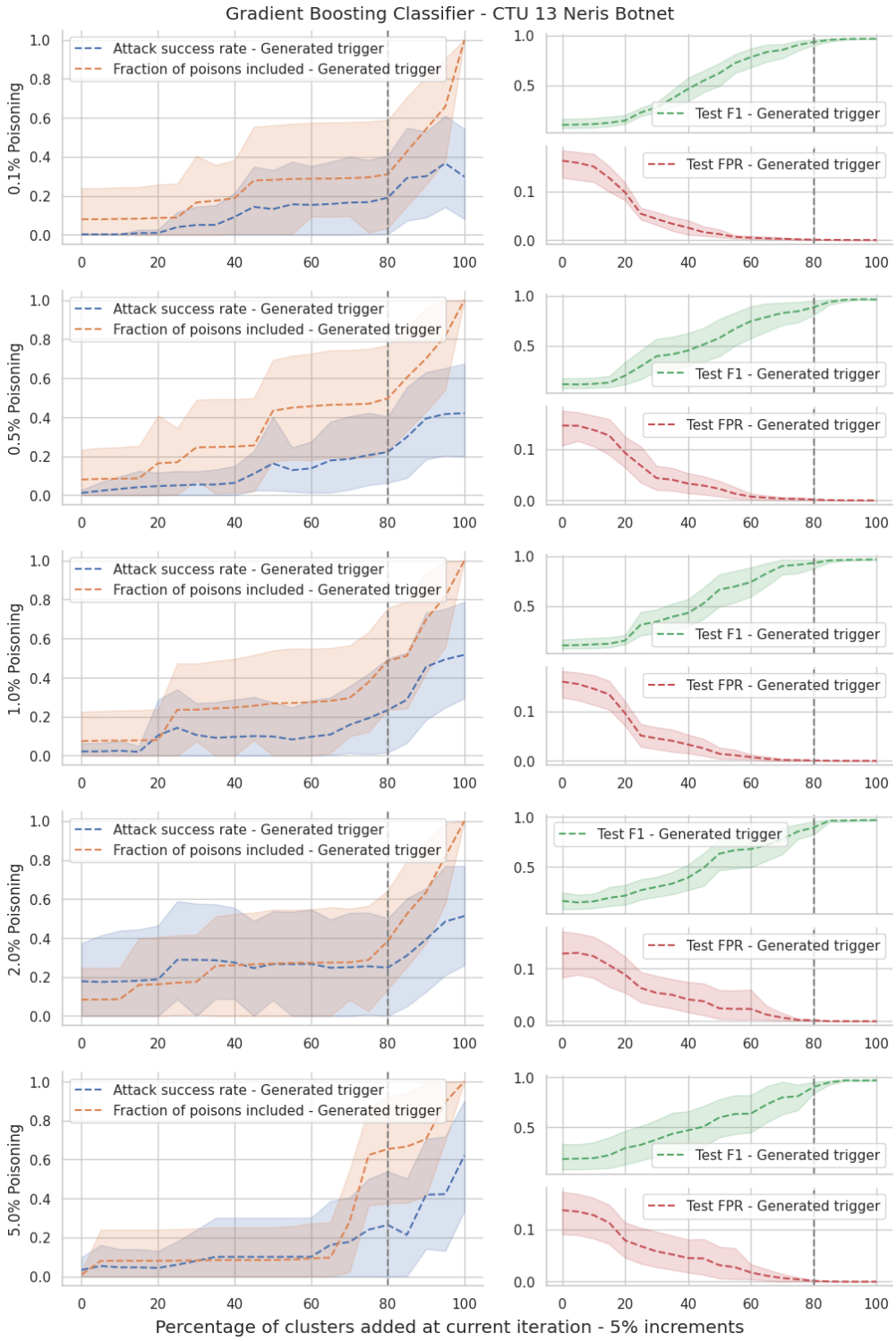


FIGURE 5.6: Iterative scoring on the CTU-13 botnet classification task for the gradient boosting model. The attack was conducted with the *generated trigger* strategy. The plot shows average metrics for the SHAP and Entropy attacker feature selection, at 5 different poisoning rates.

TABLE 5.3: Average model utility metrics on CTU-13. Results reported for different victim architectures, at different poisoning percentages. All results are averages of 10 runs, with two attack strategies and 5 random seeds.

Trigger type	Model type	Poisoning %	F1 at 80.0%	FPR at 80.0%	Poisons in D_{clean} at 80.0%
Full	Neural Network	0.1%	48.19%	1.45%	10.99%
		0.5%	45.97%	1.62%	31.71%
		1%	52.31%	1.30%	13.89%
		2%	49.33%	1.36%	12.03%
		5%	64.55%	0.83%	20.51%
	Gradient Boosting	0.1%	85.93%	0.21%	0.00%
		0.5%	94.84%	0.04%	2.72%
		1%	95.43%	0.04%	0.62%
		2%	87.13%	0.18%	0.01%
		5%	96.50%	0.02%	0.03%
Generated	Gradient Boosting	0.1%	93.26%	0.07%	30.99%
		0.5%	88.51%	0.18%	49.64%
		1%	93.16%	0.08%	48.59%
		2%	89.12%	0.17%	38.49%
		5%	90.10%	0.14%	65.36%

The generated trigger, on the other hand, is much stealthier, and a larger fraction of poisoning points manages to pass through the filtering process at these pre-defined thresholds, as highlighted in Figure 5.6. However, given that this attack is designed for stealthiness rather than effectiveness, the attack success rate is limited even if a fraction of the contaminants ends up in the training data. Finally, we note here that, as initially observed in [190], the generated trigger for the neural network model has a strong adversarial example effect at inference time. That is, the trigger pattern itself induces the classifier to misclassify the point, even if no poisoning attack took place. Therefore, this case falls outside the scope of our defense, which is targeted at countering backdoor poisoning attacks.

While effective at removing the poisons (ASR is ranging from 0.0% to 6.45%), this baseline remediation strategy may reduce the utility of the models as a side-effect of discarding the entire clusters below the threshold (including their clean data). The F1 and FPR utility metrics reported in Table 5.3 average from 0.02% to 0.21% for FPR, and 86% to 97% for F1.

Patching.

The patching-based sanitization strategy addresses the degradation in utility metrics. Considering the same threshold at 80%, we can directly compare the effects of patching to filtering for the experiment with the gradient boosting model on CTU-13. Table 5.4 shows that using patching generally leads to higher average values of the F1 score on test data (91% - 95%), and a lower false positive rate (0% - 0.08%). On the other hand, as expected, patching is slightly less effective than complete filtering in reducing the attack

TABLE 5.4: Comparison of patching and filtering sanitization approaches at fixed threshold = 80%. Gradient boosting model on the CTU-13 dataset. Also showing the Base ASR value for the undefended attack. Results are averages of 5 runs on different random seeds, for two attack strategies Entropy and SHAP.

Sanitization	Poisoning	Base ASR	ASR	Test FPR	Test F1
Filtering	0.1%	50.70%	0.00%	0.21%	85.93%
	0.5%	85.80%	6.45%	0.04%	94.84%
	1%	88.45%	0.15%	0.04%	95.43%
	2%	85.75%	0.00%	0.18%	87.13%
	5%	79.90%	0.00%	0.02%	96.50%
Patching	0.1%	50.70%	0.00%	0.08%	90.61%
	0.5%	85.80%	10.95%	0.05%	93.19%
	1%	88.45%	9.75%	0.03%	93.37%
	2%	85.75%	9.30%	0.05%	92.64%
	5%	79.90%	10.95%	0.00%	95.27%

success (0% - 11% ASR). Therefore, the defender can adopt the patching approach if they want to trade-off some defensive effectiveness for reduced degradation in model utility.

Loss analysis and sanitization

We performed the loss deltas analysis outlined in Section 5.5.4 applying it to the experiments on the CTU-13 dataset with the gradient boosting model. For this experiment we use a threshold Z-score of 2, that is, we mark as suspicious any cluster whose loss delta is lower than 2 times the standard deviation of the observed loss deltas during the initial training iterations, up to 80%. The results of both filtering and patching sanitization strategies are reported in Table 5.5.

This approach leads to a significant improvement of the F1 and FPR scores, especially for the filtering sanitization. In this case, we observe up to a 0.2% (95% relative improvement) decrement in FPR and up to 11% increase (12.5% relative improvement) in the F1 score on the test set. However, the attack success remains relatively high, and therefore fixed threshold filtering may be preferable in this scenario. We note here that the defender can always trade off smaller improvements in F1 and FPR scores for a stronger protection from the attack, by increasing the value of the threshold, for instance using $z_t = 1\sigma$.

5.6.3 Evaluation on malware classification

We observe similar trends in our experiments on the malware classification task, as summarized in Figure 5.7. These experiments were run attacking the LightGBM classifier, proposed with the EMBER dataset, using the two strongest attack strategies, based on independent feature and value selection: CountAbsSHAP and MinPopulation. Each

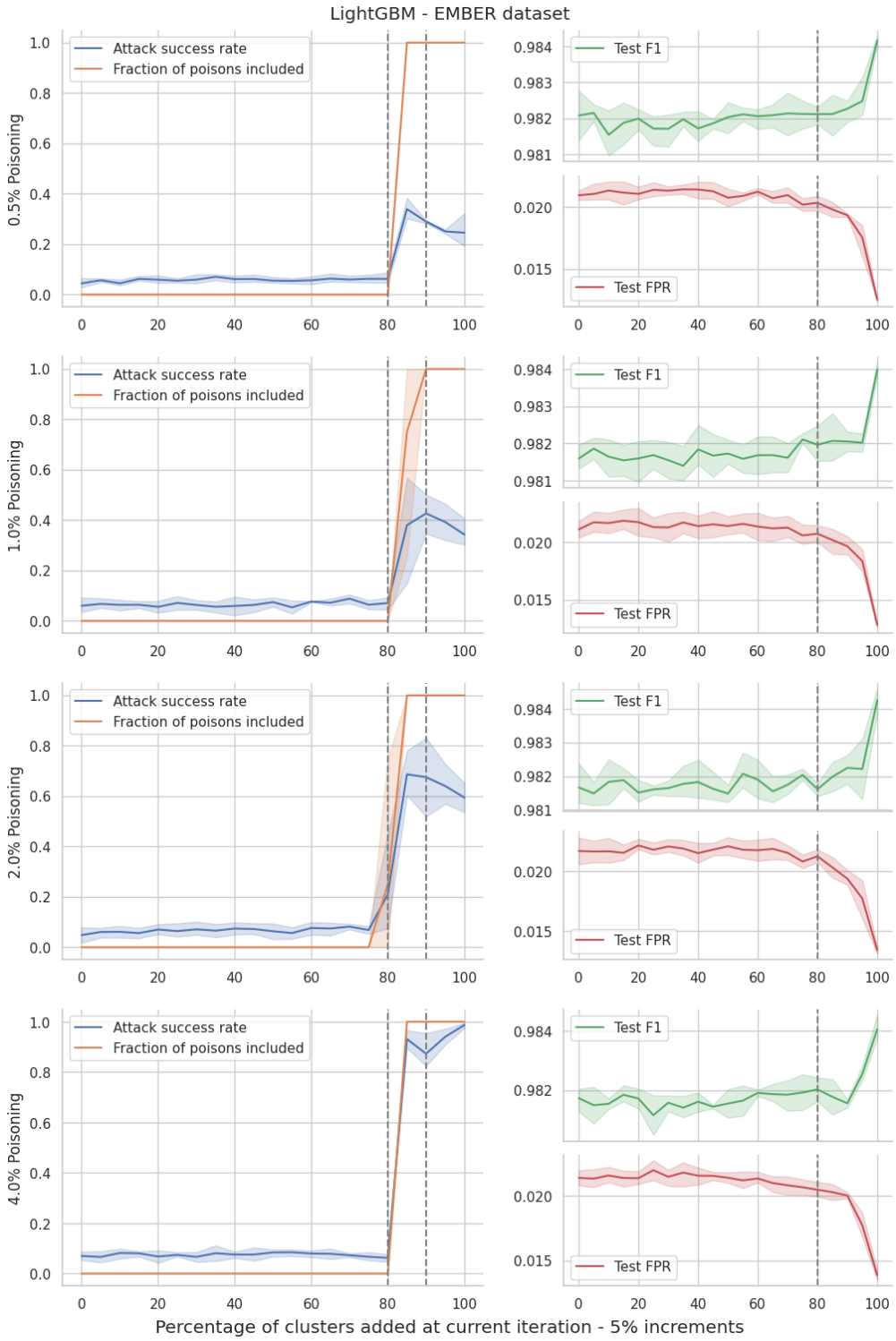


FIGURE 5.7: Iterative scoring on the EMBER malware classification task for the *LightGBM* model. The plot shows average metrics for a set of experiments: MinPopulation and CountAbsSHAP attack strategies, at 4 different poisoning rates.

TABLE 5.5: Comparison of patching and filtering sanitization approaches after loss analysis, using $z_t = 2\sigma$. Gradient boosting model on the CTU-13 dataset. Results are averages of 5 runs on different random seeds, with two feature selection approaches, for different trigger refinement strategies.

Sanitization	Attack - Poisoning		Base ASR	ASR	Test FPR	Test F1
Filtering	Full	0.1%	50.70%	0.00%	0.01%	96.94%
		0.5%	85.80%	30.15%	0.00%	97.07%
		1%	88.45%	45.80%	0.00%	96.54%
		2%	85.75%	53.00%	0.00%	96.85%
		5%	79.90%	43.15%	0.00%	96.75%
	Generated	0.1%	29.75%	35.80%	0.01%	96.80%
		0.5%	42.05%	40.95%	0.01%	96.62%
		1%	51.65%	39.40%	0.01%	96.97%
		2%	51.25%	46.20%	0.01%	96.82%
		5%	62.05%	42.95%	0.01%	96.46%
Patching	Full	0.1%	50.70%	3.05%	0.00%	95.46%
		0.5%	85.80%	38.05%	0.00%	96.09%
		1%	88.45%	44.20%	0.00%	95.52%
		2%	85.75%	51.70%	0.00%	96.60%
		5%	79.90%	33.85%	0.00%	95.38%
	Generated	0.1%	29.75%	30.25%	0.00%	96.63%
		0.5%	42.05%	44.10%	0.00%	96.32%
		1%	51.65%	48.50%	0.00%	96.91%
		2%	51.25%	49.50%	0.00%	96.85%
		5%	62.05%	60.65%	0.00%	96.51%

experiment was run twice with different random seeds, and we tested 4 different poisoning rates from 0.5% to 4%.

Fixed threshold filtering

In this scenario we observe a quicker rise in the fraction of poisoning points included in D_{clean} after the 80% threshold. Notwithstanding, in general the fixed threshold heuristic is still quite effective in thwarting the attack. Compared to the experiments on the network classification task, the changes in both F1 and FPR scores are sharper at higher percentages of included clusters, even if the absolute values of the decrements is generally smaller. We report 2% FPR and 98% F1 on average, across various poisoning rates, while the ASR has been reduced to 6%-21%.

In this scenario too, the stealthiest version of the attack, using the Combined SHAP strategy, leads to a larger fraction of poisoning points included in D_{clean} before the fixed threshold. However, as shown in Figure 5.8, the attack success rate remains relatively low unless high poisoning percentages are used.

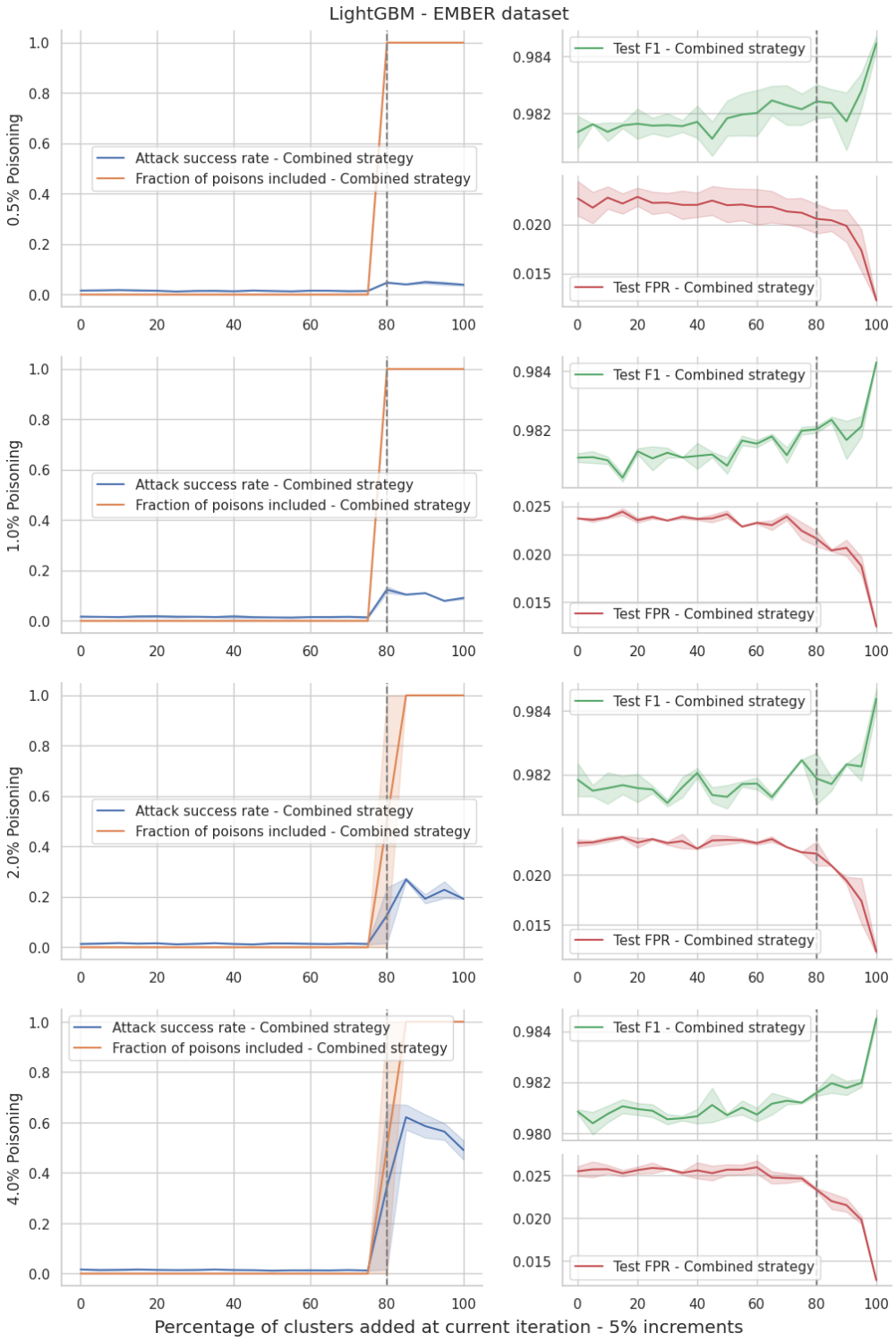


FIGURE 5.8: Iterative scoring on the EMBER malware classification task for the LightGBM model. The plot shows average metrics for the set of experiments on the *Combined SHAP* attack strategy at 4 different poisoning rates.

TABLE 5.6: Comparison of patching and filtering sanitization approaches at fixed threshold = 80%. LightGBM model on the EMBER dataset. Also showing the Base ASR value for the undefended attack. Results are averages of two runs on different random seeds, for the two Independent attack strategies.

Sanitization	Poisoning	Base ASR	ASR	Test FPR	Test F1
Filtering	0.5%	24.53%	6.19%	2.03%	98.21%
	1%	34.27%	7.04%	2.08%	98.20%
	2%	59.42%	20.98%	2.13%	98.16%
	4%	98.67%	6.22%	2.05%	98.20%
Patching	0.5%	24.53%	25.66%	1.65%	98.31%
	1%	34.27%	25.13%	1.64%	98.29%
	2%	59.42%	68.18%	1.64%	98.34%

TABLE 5.7: Comparison of patching and filtering sanitization approaches after loss analysis, using $z_t = 2\sigma$. LightGBM model on the EMBER dataset. Also showing the Base ASR value for the undefended attack. Results are averages of two runs on different random seeds, for different strategies.

Sanitization	Attack - Poisoning		Base ASR	ASR	Test FPR	Test F1
Filtering	Independent	0.5%	24.53%	5.60%	2.01%	98.22%
		1%	34.27%	6.51%	2.08%	98.18%
		2%	59.42%	6.38%	2.12%	98.15%
		4%	98.67%	7.03%	2.04%	98.21%
	Combined	0.5%	3.89%	1.37%	2.08%	98.24%
		1%	9.09%	1.84%	2.29%	98.15%
		2%	19.20%	1.49%	2.22%	98.16%
		4%	49.04%	1.45%	2.43%	98.11%
Patching	Independent	0.5%	24.53%	19.46%	1.68%	98.33%
		1%	34.27%	36.11%	1.68%	98.31%
		2%	59.42%	47.22%	1.67%	98.33%
	Combined	0.5%	3.89%	19.86%	1.65%	98.29%
		1%	9.09%	36.04%	1.76%	98.32%
		2%	19.20%	36.41%	1.84%	98.27%

Patching.

The patching strategy performs relatively poorly at preventing the attack in this scenario, especially at higher poisoning rates. Conversely, as in the experiments on network data, it outperforms the pure filtering approach for what concerns preserving model utility. In particular, the FPR obtained through patching are consistently better, with an average gain of about 0.5%, than those obtained with filtering at different poisoning rates, as shown in Table 5.6.

Loss analysis and sanitization

Table 5.7 shows the results of applying our loss analysis to identify the clusters to sanitize. The results are reported for a threshold Z-score of 2 standard deviations. This approach reflects the general trends observed before, but improves on the fixed threshold method discussed above when applied with both the filtering and patching sanitization approaches.

5.7 Related work

In this section, we provide related work on backdoor mitigation strategies, with a particular focus in the cybersecurity domain. For a more detailed treatment of backdoor attacks we refer the reader to Chapter 2.

5.7.1 Mitigations against backdoor attacks

Three main directions have emerged in the study of backdoor defenses. One approach has focused on developing techniques to obtain certifiable robustness guarantees to poisoning attacks, the second has looked at detecting and filtering out poisoned samples before training the model, while the third aims at directly purifying the poisoned model by unlearning the backdoor association.

Certified defenses to poisoning attacks aim to provide rigorous mathematical guarantees on model quality in the presence of poisons. In some of the first work on certified defenses, Steinhardt et al. [207] construct approximate upper bounds on the loss using a defense based on outlier removal followed by empirical risk minimization. They show that an oracle defender who knows the true class mean is very powerful. However, a data-dependent defender that uses the empirical means performs poorly at filtering out attacks. Deep Partition Aggregation (DPA) [122] splits the training dataset into k partitions, which limits the number of poisons that any one member of the ensemble can see. Unfortunately, the number of partitions that are necessary can be high compared to the number of poisons considered. To illustrate, using 250 partitions on CIFAR-10, DPA is able to certify 50% of the dataset to just nine insertions/deletions applied to the training set.

A related technique known as Deterministic Finite Aggregation [234] provides modest improvements over DPA. In addition to the limited guarantees, a large number of partitions may not be practical in many security applications, as the number of malicious samples available is often small. Randomized smoothing is a technique in which noise is added to the training data to attempt "smooth-out" any adversarial perturbations introduced by an attacker. However, the guarantees that are enabled are fairly small. Wang

et al. [231] are only able to certify 36% of MNIST against an adversary that perturbs up to 2 pixels per image.

In the second category, a common detection strategy utilizes deep clustering, which consists in partitioning on the learned representation of the neural network. Chen et al. [42] propose *activation clustering*, a defense method that runs 2-means clustering on the activations of the last hidden layer, then discards the smallest of the two clusters. Tran et al. [221] propose a defense based on *spectral signatures*. This method computes outlier scores using singular value decomposition of the hidden layers, then removes the samples with top scores and re-trains. An extension of this idea is used by the SPECTRE method [84], which introduces the idea of estimating the mean and covariance matrices of the clean data, using robust statistics, and using these to whiten the data, prior to computing the singular value decomposition.

The third category includes various pruning and fine-tuning methods aimed at unlearning the backdoor pattern. Liu et al. [130] propose *fine-pruning*, a strategy that attempts to disable backdoor behavior by eliminating less informative neurons prior to fine-tuning the model on a small subset of clean data. Li et al. [125] introduce *neural attention distillation*, a purifying method that utilizes a teacher network to guide the fine-tuning of a backdoored student network on a small subset of clean training data. In a recent study, Heng et al. [87] employ *selective amnesia* (SEAM), where catastrophic forgetting is induced by retraining the neural network on randomly labeled clean data; after both primary and backdoor tasks are forgotten, the randomized model is retrained on correctly labeled clean data to recover the primary task. In Section 5.4 below, we evaluate a direct application of SEAM to the attacks we study, and show the drawbacks of that defensive approach in our scenario. We note also that a similar approach, based on randomized unlearning and detection is proposed by Qi et al. [174].

Other approaches in the literature involve detecting if a model is backdoored (ABS [131], MNTD [245]), detecting if a particular label has been under attack (Neural Cleanse [232]), or performing topological analysis of activations at multiple layers in a neural network model (TED [146]). In cybersecurity settings, the only defense we are aware of is Nested Training [227, 226], a data sanitization method that relies on an ensemble of multiple models, each based on different subsets of the training set. Nested Training has been applied to poisoning availability and backdoor attacks in network traffic and malware classifiers, but it incurs high cost and degradation in model performance.

5.8 Discussion and Conclusion

Defensive mechanisms are always subject to limitations and the general problem of defending from arbitrary poisoning attacks is far from being solved. With this work we

address clean-label backdoor attacks, and propose an effective mitigation approach for existing threats.

5.8.1 Limitations

Differently from provable defenses, our method is heuristic in nature, trading off protection guarantees for large improvements in retained model utility. In contrast to other heuristic mitigation approaches introduced in the context of computer vision systems, our method relaxes some strong assumptions on clean data availability and victim model architectures that are difficult to satisfy in the cybersecurity domain.

While we defend against the attack formulations proposed in literature, adaptive attacks against our defense are possible. For instance, an adversary could change the feature selection process in the attacks to choose a fraction of relevant features and a fraction of non-relevant ones. While this would likely reduce the attack success rate, it may also reduce the likelihood of poisoning points clustering together in the relevant feature subspace. Therefore, this would allow the attacker to arbitrarily trade-off attack success to reduce the likelihood of being discovered.

5.8.2 Conclusions

In this work we introduce a mitigation mechanism against clean-label backdoor poisoning attacks targeted at cybersecurity classifiers. Our method eliminates many of the most common assumptions used in other defensive techniques against backdoor attacks. Namely, we remove the need for separate clean datasets, which can be difficult to obtain for cybersecurity tasks, and any assumptions on the architecture of the models used. Our defense effectively reduces the attack success rate in multiple scenarios, while also preserving a high model utility and a low false positives rate.

Chapter 6

Network-Level Interference in Federated Learning

The previous chapters dealt with the robustness of models trained by a single centralized entity. Recent developments in Collaborative Learning, and Federated Learning in particular, showed the potential utility of these systems to train models in a distributed fashion leveraging local, non independent and identically distributed (i.i.d.), data.

In this chapter we will see that this setting too is susceptible to training-time attacks. The work presented in [191] shows that interfering with the network traffic, an adversary can vastly reduce the global model accuracy on a target population.

Chapter Summary

Federated learning is a popular strategy for training models on distributed, sensitive data, while preserving data privacy. Prior work identified a range of security threats on federated learning protocols. We observe that the communication between clients and server is critical for the learning task performance, and highlight how communication introduces another vulnerability surface in federated learning. We consider the impact network-level adversaries might have on training federated learning models. We show that attackers orchestrating dropping of network traffic of carefully selected clients can significantly decrease model accuracy on a target population. Moreover, we show that a coordinated poisoning campaign from a few clients can amplify the dropping attacks. Finally, we develop a server-side defense which mitigates the impact of our attacks by identifying and up-sampling clients likely to positively contribute towards target accuracy. We comprehensively evaluate our attacks and defenses on three datasets, under both plain and encrypted communication channels.

6.1 Problem Definition

Federated Learning (FL) or collaborative learning, introduced by McMahan et al. [139], has become a popular method for training machine learning (ML) models in a distributed fashion. In FL, a set of clients perform local training on their private datasets and contribute their model parameters to a centralized server. The server aggregates the local model parameters into a global model, which is then disseminated back to the clients, and the process continues iteratively until convergence is established.

Training ML models using a federated approach allows the server to take advantage of a large amount of training data and each client to retain the privacy of their data. Popular applications include natural language processing (NLP) to learn from user activities on mobile phones [248], smart healthcare [183], and customized retail services. FL can be deployed in either a cross-silo setting with a small number of participants available in all rounds of the protocol, or in the cross-device setting with a large number of participants, which are sampled by the server and participate infrequently in the protocol.

Recent work studying the security of FL protocols has highlighted the risk of attacks by compromised clients through data poisoning [218] and model poisoning [20, 16] under different objectives such as availability, targeted, and backdoor attacks. While availability objectives aim at compromising the accuracy of the entire model indiscriminately [68, 200], targeted and backdoor attacks only affect a specific subset of samples [16, 212, 233] and are harder to detect.

Among the defensive approaches that the security community proposed, availability attacks can be mitigated by various gradient filtering methods [68, 200, 239], while targeted and backdoor poisoning attacks could be addressed by clipping gradient norms at the server, during model updates, to limit the individual contributions of clients to the global model [212, 16].

We observe that, for federated learning, communication between server and clients plays a critical role as the global model is learned in rounds with contributions from multiple clients. Thus, the communication represents another point of vulnerability that an attacker can exploit to influence the global model learned by the system. In this case, an attacker does not need to compromise the clients, but rather leverage network-level information and attack capabilities to prevent the ML algorithm from *receiving* the information needed to learn an accurate good model.

An attacker can exploit the specific communication channels for different FL architectures to perturb messages sent between server and clients. For example, cross-device FL is usually deployed for mobile and edge devices, connected through wireless communication channels which typically experience higher packet loss and are vulnerable to

physical-layer attacks, such as jamming [244, 18]. For architectures relying on overlay networks, network-level adversaries might interfere with the data delivery by conducting BGP hijacking attacks [46], compromising routers [49, 185] or hosts[230], or exploiting vulnerability on transport-level protocols [104] to perform packet dropping, packet delay, or traffic rerouting.

Such network-level attacks have been shown to impact other systems such as Bitcoin [10], payment-channel networks [235], and connected cars [101]. These adversaries are especially relevant when political or economic incentives result in an adversarial autonomous system (AS), for which sub-populations are geographically localized. A government may run this attack to censor a word prediction model (especially targeting a country's minority language). A corporate AS may attempt to modify words associated with their competitors or unfavorable political movements.

In this study, we are interested in understanding the impact network-level adversaries can have on machine learning models trained with federated learning. We are less interested in how an attacker will position themselves in the network to conduct the attacks, but rather in the ways the attacker can exploit information sent over the network to maximally damage the learning algorithm. We analyze for the first time the impact of network-level adversaries on federated learning and show that attackers who can carefully orchestrate dropping of network traffic of selected clients (either by controlling parts of the network or compromising the clients) can significantly decrease model accuracy on a target population. Specifically, we design an algorithm that identifies a set of clients who contribute heavily to a selected learning task.

We show that by identifying highly contributing clients and dropping their traffic, an attacker is more successful than by dropping randomly selected clients. We consider scenarios where communication between server and clients is sent either in the clear or encrypted. Moreover, we show that model poisoning attacks from a small set of clients can further amplify the impact of the targeted dropping attacks we consider. As an example, on a text classification problem, selectively dropping updates from only 5 clients from the target population can lead to a decrease in accuracy from 87% to 17%, on a particular target class. Furthermore, by adding a poisoning attack of equal size, the model accuracy decreases to 0% on the same target class.

Existing network-level defenses against packet dropping might mitigate the attacks under some circumstances, such as when the amount of traffic dropped for particular clients is observable with respect to the normal network loss. However, in cross-device FL settings considered here, a small set of clients are selected at random by the server, and therefore clients participation in the protocol is usually infrequent. Dropping a few packets at large intervals of time would be difficult to detect and mitigate at the network

level. Additionally, FL system owners do not usually control the underlying network and might not be able to implement network-level mitigations to make communication more resilient. Complementary to such network-level defenses, we propose a server-level defense that is agnostic to how the dropping attack is performed.

Our defense modifies client sampling in each round of FL training to increase the likelihood of selecting clients who sent useful updates in previous rounds of the protocol, while decreasing the probability of selecting less relevant clients. Interestingly, client selection for sampling in the defense leverages the same procedure for client identification employed by the network-level adversary. The defense is extremely effective against a targeted dropping attack. For instance, in the same text classification task mentioned above, while an unmitigated attack would completely disrupt the model accuracy on the target task (from 87% to 17%), the defense achieves an accuracy of 96%, which exceeds that of the original model before an attack is mounted.

Moreover, our defense can be combined with a standard poisoning defense based on gradient clipping to withstand both targeted dropping and model poisoning attacks. On the same task, the combined dropping and poisoning attack brings the target accuracy to 0, and the combination of our defense with clipping results in 94% accuracy on the target population. The defense is even more effective under encrypted communication, the most common deployment model for FL, in which the improvements compared to the original accuracy can be as high as 34%.

To summarize, the contributions of this work are:

(i) We consider for the first time the impact of network-level adversaries in federated learning protocols, perform an evaluation of a targeted packet dropping attack, and show that its impact can be further amplified by model poisoning attacks;

(ii) We design a client identification procedure that is a building block in our attack and defense strategies, and that can be configured with different levels of access to the network traffic sent between clients and server;

(iii) We show that a defensive strategy based on up-sampling highly contributing clients to the learning task is promising in training more resilient models. The defense can be combined with gradient clipping to mitigate both targeted dropping and poisoning attacks, and in some cases results in higher accuracy than the original model before the attack.

(iv) We evaluate the proposed attacks and defenses and show that they obtain interesting results across multiple model architectures, datasets, and data modalities (image and text).

For reproducibility, all code is released publicly[1].

6.2 Notes on Federated Learning

Federated learning (FL) considers a setting with a set of n clients, each with a local dataset D_i, and a server S. FL is a distributed machine learning methodology in which clients train locally on their own datasets and the server requests model updates, rather than data, from users, building an aggregated global model iteratively over time [139].

There are two main deployments models for FL: *cross-silo* with a small set of clients participating in every round, and *cross-device* models with a large number of clients which are sampled by the server per round [105]. We consider here the Federated Averaging training algorithm designed for cross-device settings [139], shown in Algorithm 4. In each round $1 \leq t \leq T$, the server randomly selects a subset of $m \leq n$ clients to participate in training, and sends them the current global model f_{t-1} (initialized f_0 with Glorot Uniform initializer [72]). Each selected client i trains locally using dataset D_i, for a fixed number of T_L epochs. The server updates the global model f_t by mean aggregation of local updates U_i:

$$f_t = f_{t-1} + \eta \frac{\sum_{i=1}^{m} U_i}{m}. \tag{6.1}$$

FL has been designed for protecting client's local data privacy, but data privacy can be further enhanced by secure aggregation performed via Multi-Party Computation (MPC) [26, 69].

Algorithm 4: Federated Averaging Protocol

Data: Clients $\mathcal{C} = \{D_i\}_{i=1}^{n}$, Federated Learning Server S, rounds T, clients per round m, aggregation learning rate η

Function `FedLearn(`$S, \mathcal{C}$`):`

```
// Function run by server
f_0 = InitializeModel()
for t ∈ [1, T] do
    // Get updates from m participants in round i
    M_t = SelectParticipants(C, m)
    RequestUpdate(f_{t-1}, M_t)
    U_t = ReceiveUpdate(M_t)
    // Update and send out new model
    f_i = UpdateModel(f_{t-1}, U_t, η)
    BroadcastModel(f_t, C)
```

[1] https://github.com/ClonedOne/Network-Level-Adversaries-in-Federated-Learning

6.3 Threat Model

The most studied attacks against federated learning are poisoning attacks. In data poisoning attacks, an adversary injects maliciously crafted data points to poison the local model [218]. Model poisoning attacks (POISON_MODEL) where the adversary compromises a set of clients and sends malicious updates to the protocol with the goal of achieving a certain objective in changing model's prediction [16, 212, 233] have also been studied. Both these attacks are conducted by manipulating directly the inputs to the machine learning algorithm.

For federated learning, communication between server and clients represents another attack surface that an attacker can exploit to influence the global model learned by the system without necessarily needing to compromise the clients. Instead, based on their network-level capabilities, the adversary can observe communication sent over the network and prevent the machine learning algorithm from *receiving* the information needed to learn a good model. We refer to such an adversary as a *network-level adversary*.

6.3.1 Adversary's Goals and Capabilities

FL protocols such as Federated Averaging [139] usually abstract the communication protocol between the server and clients. In practice, clients will communicate with the server either directly through a TCP connection, or through a multi-hop overlay network of hop-by-hop TCP or customized transport services. Finally, the last hop connecting the client to the Internet is often in the form of wireless communication. Network infrastructure and routing protocols facilitate all this communication through physical or logical connectivity. By exploiting the underlying routing protocols and network topology an attacker can position themselves to observe and interfere with data of interest; they can then further impact the accuracy of the global model and create an effect similar to that created through a data or model poisoning attack.

Model Dropping Attacks: The most basic attack that can be conducted by a network-level adversary is to drop packets. In the FL setting, this means dropping local models that could have contributed to the accuracy of the global model. In order to conduct such an attack the adversary will need to know who are the clients that are part of the federated learning system and specifically what clients are requested by the server to participate in the protocol in each round. Such knowledge can be acquired by monitoring the network, by leveraging network topology and ISP connectivity, by attack placement near the server, and more. While this assumption about the attacker is relatively strong, it allows us to answer questions such as "are network-level attacks against federated learning meaningful".

Packet dropping attacks can be achieved in multiple ways, and have been studied for network-level adversaries who perform physical-layer attacks in wireless networks [244, 18], router compromise [49, 185], or transport-level attacks [104]. Each of these methods will require different attacker capabilities. For example for physical layers attacks for wireless clients, the attacker needs to be in their proximity - this is both a powerful attack since jamming is usually difficult to defend against, and limiting since it will be difficult for the attacker to attack geographically distributed clients without significant resources. In the case of routing, while a router might be more difficult to compromise, in practice the impact can be bigger as it might have control over the traffic of multiple clients. Last, but not least, if customized overlays are used for communication, compromising nodes in the overlay is slightly easier as they are typically hosts, and the number of clients that can be impacted depends on the scalability of the service.

We note that we model packet dropping agnostic to the exact method used by the attacker for dropping network packets, since the server cannot distinguish between a network-level adversary, client compromise, or simply client unavailability when not receiving client updates.

Adversarial goal: We consider that the attacker targets the accuracy for a particular population. We define a population to be one of the classes in the learning task, but this notion could be extended to sub-classes as well. Ideally, the attacker would like the model to retain its test accuracy for all data points outside of the victim population to avoid trivial detection. We therefore do not consider poisoning availability attacks which are detectable and can be addressed with existing defenses [68, 200, 239].

Attack strategies: The attacker has several decisions to make when conducting the attack: (1) what clients to select to drop their contributions, (2) when to start the attack given that federated learning is an iterative protocol, and (3) how many packets (i.e., local models) to drop. Ideally, an attacker wants to maximize the strength of the attack while minimizing detection. We focus on a targeted dropping attack in which the attacker selects a set of clients contributing highly to the target class for dropping their contributions. We compare that with an attack strategy which drops the traffic of a subset of randomly chosen clients.

Adversarial network-level knowledge: We distinguish between different representative adversarial scenarios for the communication in the FL protocol:

1. **COMM_PLAIN**. All communication between clients and server is unencrypted. This is the scenario where a network-level adversary obtains maximum information, as they can observe all the transmitted data. Even though this might not

correspond to a practical scenario, we study this model to determine the impact of the worst-case, most powerful adversaries.

2. **COMM_ENC**. All communication between clients and server is encrypted. This is a realistic scenario, which can be implemented using IPSec, HTTPS or application-level encryption. In this case, the network-level adversary could infer the set of clients participating in each round, but not the exact model updates they send.

3. **COMM_ENC_LIMITED**. A special case of COMM_ENC where communication is encrypted, and the adversary is limited to only observe a fixed subset of the clients participating in the protocol.

Adversarial global model knowledge: Since cross-device FL is an open system, the adversary can always participate in the FL protocol as one of the clients. We assume that the adversary obtains the global model updates f_t at each round t.

To summarize, in the rest of the study we consider an adversary that has either the COMM_PLAIN, COMM_ENC or COMM_ENC_LIMITED network capability, has knowledge of the global model at each round, and targets a particular victim population of interest. We also consider an adversary that additionally will have POISON_MODEL poisoning capabilities.

6.4 Network-Level Attacks on Federated Learning

In this section we describe in detail our targeted dropping attacks against a population in FL. We assume the FL model described in Section 6.2 and an attacker that has either the COMM_PLAIN or COMM_ENC network capability. We start by discussing that randomly selecting clients for dropping is not an effective strategy. We then describe a procedure to identify clients that will make the attack more effective and analyze the different parameters that control the attack. Finally, we show how our attack can be significantly amplified if in addition to the network level capabilities, the attacker also has POISON_MODEL poisoning capability.

6.4.1 Dropping Attack with Random Client Selection

The most basic strategy a network-level adversary might employ is to randomly select a subset of clients and drop their local model updates. Usually, such a strategy is not effective in achieving the degradation of model accuracy on a class of interest, which is the adversarial objective we are interested in. Later, in Table 6.2 in Section 6.6.2 we present detailed results for such a random dropping strategy evaluated for different number of clients whose traffic is dropped. Our finding is that the model accuracy on

the target class is decreased by an average of 5%, even in cases where the number of clients selected for dropping matches the number of clients holding samples from the target class. These findings motivate us to develop new methods for client selection that will help the adversary identify the most relevant clients for degrading the accuracy on the target class.

6.4.2 Identification of Highest-Contributing Clients

The cornerstone of our targeted packet dropping attack is an algorithm in which the adversary first determines the set of clients contributing the most to the target population of interest and then selectively drops their traffic. We design a Client Identification procedure aimed at determining the set of clients whose model updates lead to the largest improvements in target population accuracy. Our Client Identification algorithm, described in Algorithm 5, supports both plain and encrypted network communication, and is applicable to a range of FL system implementations. We demonstrate in Section 6.6 that if the adversary has high success at identifying the clients contributing to the task of interest, then the targeted dropping attack impact is much higher compared to random dropping.

The main intuition is that the network-level adversary will observe all communications – including global model updates and local model updates if possible – in the FL protocol for a number of, at least, T_N rounds. In each of these rounds, the adversary computes the loss of the model before and after the updates on the target class. Rounds in which the loss decreases correspond to an increase in accuracy on the target class. The adversary tracks all the clients participating in those rounds, and computes a loss difference metric per client. In the plain communication case, the adversary will determine exactly which clients decrease the loss, while in the encrypted communication case the adversary only observes the aggregated updates and considers all clients participating in the round to be collectively responsible for the loss variation.

In more detail, to measure each client's contribution to the target population, the adversary computes the loss difference between successive model updates on the target population (using a representative dataset D^* from the population):

$$L_t^j = \ell(f_{t-1}, D^*) - \ell(f_t^j, D^*) \tag{6.2}$$

Note that the second term $\ell(f_t^j, D^*)$ is the loss of the local update of client j in round t for COMM_PLAIN, but becomes the loss of the global model $\ell(f_t, D^*)$ when the adversary only has access to aggregated updates in COMM_ENC. The value L_t^j measures the decrease in the target class loss before and after a given round t. For clients who contribute heavily to the accuracy on target data, this value will be large, so that dropping these

Algorithm 5: Loss Difference Client Identification

Data: Target Population Dataset D^*, loss function ℓ, rounds T_N, count of clients to drop k_N, visibility parameter v.

Function `ClientIdentification`(D^*, ℓ, T_N, k_N):

- $f_0 = \text{GETGLOBALMODEL}(0)$
- $\Delta = []$
- **for** $t \in [1, T_N]$ **do**
  - $M_t = \text{GETPARTICIPANTS}(v)$
  - $f_t = \text{GETGLOBALMODEL}(t)$
  - **if** COMM_ENC **then**
    - // Get loss differences for this round
    - $L_t = \ell(f_{t-1}, D^*) - \ell(f_t, D^*)$
    - **for** $j \in M_t$ **do**
      - // Associate L_t with update members
      - $\Delta[j] = \text{CONCATENATE}(\Delta[j], [L_t])$
  - **if** COMM_PLAIN **then**
    - **for** $j \in M_t$ **do**
      - // Get participant j model
      - $f_t^j = \text{GETLOCALMODEL}(j)$
      - // Get participant's loss difference
      - $L_t^j = \ell(f_{t-1}, D^*) - \ell(f_t^j, D^*)$
      - // Associate L_t^j with this participant
      - $\Delta[j] = \text{CONCATENATE}(\Delta[j], [L_t^j])$
- // Compute the average loss change for each client
- **for** $j \in \Delta$ **do**
  - $\Delta[j] = \frac{1}{|\Delta[j]|} \sum \Delta[j]$
- // Identify largest loss decreases
- $Z = \text{GETLARGESTVALUES}(\Delta, k_N)$
- **return** Z

clients will heavily compromise the performance of the model on the target. However, when only the aggregated update is seen, all clients participating in a round will have the same value for L_t^j. This makes it difficult to identify which specific clients contribute to the target population.

To more reliably estimate client contributions, especially with aggregation of model updates, the values of L_t^j are accumulated over time. The adversary maintains a list for each client j of the values L_t^j for rounds t in which it participated. This allows the adversary to update the list of clients most relevant to the target class accuracy at every round. Empirically, computing the mean of L_t^j for each client j across all rounds it participates in works well at identifying the clients contributing most to the target class.

6.4.3 Dropping Attack with Identification of Highest-Contributing Clients

The targeted dropping attack starts by the adversary performing the Client Identification procedure, by running Algorithm 5 during the first T_N rounds of the protocol. During this time period, the FL training happens as usual and the global model is aggregated from local updates. Using the parameter analysis in Section 6.4.4, the adversary selects T_N rounds and identifies in expectation k_N clients contributing updates for the target class. After Client Identification is performed, the network adversary drops contributions from the k_N clients in every round in which they participate after round T_N.

As the adversary can expect an increase in identification accuracy by monitoring more rounds of the protocol, they can repeat the Client Identification procedure in each subsequent round, and, if necessary, update the list of selected clients for dropping. We assume that the adversary is able to drop the identified clients' updates, and the server simply updates the model using all received updates. If the Client Identification protocol is not completely accurate, the attacker might drop traffic from clients who do not contribute to the target population, but we found this has a minimal impact on the trained FL model.

The selection of k_N, the number of victim clients in every round, and T_N, the number of rounds after which the attack starts, are critical for the success of the attack. Particularly, selection of T_N is very important: a small T_N will not allow the Client Identification procedure to determine relevant clients, while a large T_N can render the attack ineffective because the model might have converged before the attack started.

6.4.4 Analysis of the attack

Algorithm 5 uses several parameters: the number of clients to identify k_N and the number of rounds to wait before identification is successful T_N, which we analyze here. The remaining arguments will typically be governed by the training algorithm, protocol, and the fixed adversarial goal.

How many clients to drop?

In the FL protocol, we assume a set of n clients, out of which m are selected at random in each round. Setting the number of dropping clients k_N is mainly a tradeoff between maximally damaging accuracy on the target data and remaining stealthy by not significantly compromising the overall model performance. If k_N is too large, significant benign updates may be removed, preventing the model from being good enough to use in practice, and potentially allowing the server to identify that an active adversary, rather than standard packet loss, is to blame for the dropped updates. If k_N is too small,

however, not enough clients will be dropped to have a significant impact on the model. We consider a range of values $k_N \leq k$, where k is the number of clients holding examples from the targeted class, and show the attack effectiveness for each.

How many rounds are needed to identify the clients?

Here, we discuss how to set the number of rounds T_N for client identification for both plain and encrypted communication. When setting T_N, if the value is set too small, then the adversary will not have enough observations to reliably identify the heavily contributing clients. However, if set too large, the adversary will allow benign training too much time, resulting in high accuracy on the target population, making it more difficult to mount the attack later.

Suppose the adversary wants to wait until all n clients have participated in the protocol at least once. This is a well-studied problem, known as the *coupon collector's* [51] (our setting additionally considers batched arrivals). In this variation of the problem, the adversary must observe an expected number of $cn\log(n)/m$ batches before observing each client, for a small constant c. With values of $n = 100$, $m = 10$, roughly 46 rounds are necessary [243]. Having established a connection to the coupon collector's problem, we will extend it to model the adversary in each setting:

Plain communication (COMM_PLAIN). In the case where the network adversary receives *every* client's updates, we carry out a modification of the coupon collector analysis to identify the setting of T_N where a batch of m out of n distinct clients participate in each round, and the goal is to identify a set of k_N of k target clients. We analyze the case where batches are sampled with replacement (i.e., clients in each round may be repeated) due to ease of analysis, but this incurs a small constant overhead in the analysis [19].

We analyze the expected number of batches t_i before finding each of the first k_N target clients, using standard coupon collector analysis. The number of batches to wait for the i-th client from the k target clients is $t_i = \frac{n}{m(k-i+1)}$ in expectation, as the probability that any target client which has been unobserved appears is $\frac{m(k-i+1)}{n}$. By summing this expected batch count over i from 1 to k_N (using the linearity of expectation), we find that the expected number of batches to wait for the first k_N clients is $\frac{n}{m}(H_k - H_{k-k_N})$, where H_i is the i-th harmonic number (and using $H_0 = 0$ if $k_N = k$). As k increases, this value tends towards $\approx \frac{n}{m}(\ln(k) - \ln(k - k_N))$ rounds (when $k = k_N$, we replace $\ln(0)$ with 0).

To use a setting that is common in our experiments, where $n = 60$, $m = 10$, $k = 15$, and $k_N = 15$, we expect to wait roughly $\frac{n}{m}\ln(k) = 6\ln(15)$ rounds, or roughly 16 rounds. Note that waiting for fewer clients k_N requires fewer rounds.

Encrypted communication (COMM_ENC). In the more challenging setting of encrypted communication, the adversary only has access to mean updates and cannot localize an improved target accuracy to a particular client, as is possible when all the clients updates are available. However, clients who repeatedly participate in rounds where the target accuracy improves, can be considered as more likely targets. Moreover, clients who participate in any round where the target accuracy does not improve can be identified as not targets.

Here, we analyze only rounds where no target clients appear, as they provide certainty that no target clients are participating. We analyze the number of rounds required to identify enough non-target clients that a fixed precision level α is obtained at identification. This precision level can be reached by removing $n - k/\alpha$ clients which are known non-target clients. To analyze how many rounds it takes to collect $n - k/\alpha$ non-target clients, we compute the probability that a batch will contain non-target clients, and then compute how many non-target batches are required to collect $n - k/\alpha$ clients.

To compute the probability that a batch contains non-target clients, we notice that there are $\binom{n-k}{m}$ possible batches which have non-target clients, and there are a total of $\binom{n}{m}$ batches which are selected uniformly at random. Then the probability a batch has non-target clients is:

$$\binom{n-k}{m} / \binom{n}{m} \approx \left(1 - \frac{k}{n}\right)^m \tag{6.3}$$

To compute the number of non-target batches required to accumulate $n - k/\alpha$ non-target clients, we can use comparable coupon collector analysis as before, making the observation that each non-target batch is guaranteed to have m non-target clients. On average, an upper bound of $\frac{n}{b}(H_{n-k} - H_{k/\alpha-k})$ batches is sufficient. As an example, in a setting we use in our experiments, $n = 60$, $k = 15$, and $m = 10$, the probability that a batch contains non-target clients is roughly 5.6%. To reach a precision of $\alpha = 0.3$, we obtained a total of 26 batches on average. In practice, it is likely that precision can be even higher than this value due to overestimation from our analysis.

We note that our analysis is similar to analysis for the *group testing problem* [77], introduced by [60], used for pool testing. However, a key difference is that our algorithm must be capable of identifying members from noisy aggregate information, rather than the clean signal which is typically provided during group testing. It is possible that

more sophisticated group testing algorithms can be used to improve Algorithm 5 further by overcoming this constraint.

6.4.5 Amplifying Dropping Attack with Model Poisoning

In order to amplify the effectiveness of the targeted dropping attack, the network adversary may also collude with, or inject, malicious clients, whose presence in training is designed to further compromise the performance on the target distribution. Following Bagdasaryan et al. [16] and Sun et al. [212], we use a targeted model poisoning attack known as model replacement. Writing θ_t for the parameter of the global model f_t, in this attack, the adversary seeks to replace θ_t with a selected target θ_t^* (as is done in [16, 212], we use the θ_t^* resulting from a data poisoning attack on a compromised client's local training procedure).

The poisoned clients' local training sets are sampled identically to clients with access to the target class, with the difference of changing the labels of target class samples to another, fixed class. Writing the initial model parameters sent in the round as θ_{t-1}, the update that the adversary sends is $\theta_t^* - \theta_{t-1}$. The server aggregation then weights this update with an η/m factor. In a model replacement attack, a boosting factor β is applied to the update, so the update which is sent is $\beta(\theta_t^* - \theta_{t-1})$, increasing the contribution to overcome the η/m factor decrease.

We will show that model poisoning attacks mounted from a small set of clients controlled by the attacker can significantly amplify the impact of targeted dropping attacks and achieve close to zero accuracy on the target class.

6.5 Defenses Against Network-Level Adversaries

Several defenses against network-level attacks have been considered in previous work. For instance, defenders could monitor and detect faulty paths in the network [14], create resilient overlays [38, 157], or secure the routing protocols [91]. These defenses might increase robustness under observable packet loss (relative to the normal packet loss), but are generally not effective against stealthy attacks, such as our targeted dropping attacks, or edge-level attacks, such as model poisoning attacks mounted from attacker-controlled devices. Here, we address the question of how to design FL-specific defenses against the attacks introduced in Section 6.4. Often, the FL owner might not control the network, so we will consider server-level defenses that could complement existing network-level defenses.

Model Poisoning Countermeasures After data and model poisoning attacks against FL have been demonstrated (e.g., [20, 212, 16]), multiple systems have been proposed to

Algorithm 6: Server Defensive UpSampling Strategy

Data: Target Dataset D_S^*, rounds T_S, loss function ℓ, client count n, count of clients to upsample k_S, up-sample factor λ

Function `UpSampling`$(D_S^*, \ell, k_S, T_S, n)$**:**

```
// Identify highly contributing clients
Z = ClientIdentification(D_S^*, ℓ, T_S, k_S)
// Reduce sampling probability for most clients
p = [(n − k_S λ)/(n² − k_S n) for c ∈ [n]]
for i ∈ Z do
    // Increase sampling probability for highly contributing clients
    p[i] = λ/n
return p
```

withstand these attacks. A popular defense is to replace the Federated Averaging protocol with a secure aggregation scheme [25, 143, 250, 3]. It turns out that an adversary with knowledge of the secure aggregation scheme can adaptively evade the defense in some cases [16, 68, 200], and finding an aggregation method secure against adaptive attacks remains an open problem.

The only resilient technique we are aware of to protect against targeted model poisoning attacks is gradient clipping. Sun et al. [212] made the observation that model poisoning attacks with larger gradient norm are more effective, and therefore a natural defense is to reduce the norm of the local updates via clipping. With this method, an update Δ sent from a client is clipped to a maximum norm C, such that the update used for aggregation becomes $\min\left(1, \frac{C}{||\Delta||}\right)\Delta$. This method works particularly well against model poisoning attacks in which the local client update is boosted by the attacker [16]. For this method to be effective, the clipping norm must be carefully calibrated: it should be large enough to allow learning less well-represented classes and populations, while also small enough to prevent poisoning from changing the global model.

UpSampling Defense Strategy While gradient clipping reduces the impact of model poisoning attack, we still need to defend against the targeted dropping attack. When a targeted dropping attack is performed, the number of clients contributing legitimate updates to the target class is reduced, and the impact of dropping attack is larger on under-represented classes (i.e., classes with examples available in a small set of clients). Therefore, it is essential for the server to identify clients contributing to the target class and leverage them to improve accuracy on the target population.

Our main insights to help the server improve the accuracy on the target population are to: first, identify the legitimate clients contributing updates to the target class, and, second, modify the sampling strategy in the FL protocol to sample these clients at a

TABLE 6.1: The parameters used in our experiments.

Party	Param	Definition	Setting
Dataset	n	Total Clients	{100, 60}
Dataset	k	Clients with Target Class	{9, 12, 15}
Dataset	α_D	Dirichlet Distribution Param	1.0
Dataset	α_T	Target Class Dataset Fraction	{0.5, 0.6}
Client	T_L	Local Training Epochs	2
Client	η_L	Local Learning Rate	{ 0.1, 0.001 }
Server	η	Aggregation Learning Rate	0.25
Server	m	Clients per Round	10
Server	T	Total Training Rounds	Varied
Server	T_S	Rounds before up-sampling	Varied
Server	k_S	Up-sampled Client Count	Varied
Server	λ	Up-sampling factor	2
Server	C	Aggregation Clipping Norm	1
Adversary	T_N	Rounds before Dropping	Varied
Adversary	k_N	Number of Dropped Clients	Varied
Adversary	k_P	Number of Poisoning Clients	Varied
Adversary	β	Poisoning Boost Factor	10

higher rate. For the first component, we can leverage directly Algorithm 5 for Client Identification, using the model update knowledge available to the server. In standard FL implementations, servers receive individual model updates from the clients, while in privacy-preserving FL implementations based on MPC [26, 69] servers only receive aggregated updates. The server will determine its own parameters k_S (how many clients to identify) and T_S (how many rounds to monitor), and use its own target dataset D_S^* to estimate contributions. As in the case of the attacker, the server will repeat the Client Identification process at each successive round to refine its list of clients to up-sample. After identifying the target clients, the server can proceed to run Algorithm 6 to increase the probability with which identified clients are sampled by a factor λ and decrease the sampling probability for all other clients. We call this method presented in Algorithm 6 the UpSampling defense.

Interestingly, the UpSampling strategy can also help even if there is no dropping attack, but there are simply too few clients from some target population on which the model accuracy is low. We will show that in our evaluation in Section 6.6.7.

6.6 Experiemental Evaluation

Here we evaluate our network-level FL attacks, starting with an ideal setting, when the attacker has perfect knowledge on the set of clients contributing to the target population. We then show the effectiveness of our Client Identification algorithm, and examine the

targeted dropping attack after Client Identification, showcasing the performance of our entire attack pipeline. Finally, we show the amplification of targeting dropping attacks achieved by model poisoning.

6.6.1 Experiment Setup

We use three well known datasets (EMNIST, FashionMNIST, and DBPedia-14), which we adapt to the *non i.i.d.* setting by controlling the data distribution across clients. In all cases, the target population is represented by samples of a class arbitrarily selected from the dataset. Our datasets have balanced classes, and we use classes 0, 1, 9 for our evaluation. All reported results are averages of 4 trials, with different random generator seeds. The list of parameters used in our FL protocol is shown in Table 6.1.

Datasets and Models

EMNIST [50] is a handwritten digits recognition dataset with 344K samples. We use approximately 100K images of numbers between 0 and 9, partitioned equally, and without overlap, among 100 clients, so that each client has 1000 images. To enforce heterogeneity, we allocate samples from the target victim class to k fixed clients and vary k. For those k clients, we allocate $\alpha_T = 50\%$ of the local dataset to be target class points, while the other 50% is sampled from the remaining classes according to a Dirichlet distribution with parameter $\alpha_D = 1.0$. For clients without the target class, we sample 100% of the local training set from a Dirichlet distribution with parameter $\alpha_D = 1.0$, following [94]. We train a convolutional model, consisting of two 2D convolution layers and two linear layers, optimized using learning rate $\eta_L = 0.1$. Training consists of 100 rounds, selecting 10 clients at each round, using mean aggregation with a learning rate $\eta = 0.25$ to combine clients' updates.

FashionMNIST [237] is an image classification dataset, consisting of 70K gray-scale images of 10 types of clothing articles. This dataset is more complex than EMNIST, and has less data. Therefore, we increase the number of training rounds to $T \in \{200, 300\}$, reduce the number of clients n to 60, and set α_T to 0.6. We also set the local dataset size to 400. We use a convolutional model similar to the one used before, and fix all other parameters.

DBPedia-14 [121] is an NLP text classification dataset, which we selected to vary the data modality. DBPedia-14 consists of 560K articles from 14 ontological categories, such as "Company", "Animal", "Film". DBPedia-14 is also a relatively complex dataset, so we use the same k, T, and α_T as in FashionMNIST. We use a local dataset size of 1000, and train a model starting from pre-trained GloVe embeddings [170], followed by two 1D

TABLE 6.2: Perfect knowledge dropping. Accuracy on target population at rounds $T/2$ and T. $T = 100$ for EMNIST, $T = 200$ for FashionMNIST and DBPedia. Targeted dropping is more effective for larger number of dropped clients k_N and reaches 0 when no clients are available for the target class ($k_N = k$). For harder classification tasks such as DBPedia, accuracy on target class gets to 6% for 5 out of 15 clients dropped, and 0 when 10 out of 15 clients are dropped.

Dataset	k	Number of Clients Dropped k_N			
		0	$k/3$	$2k/3$	k
Perfect Knowledge					
EMNIST	9	0.47/0.66	0.21/0.50	0.01/0.23	0.0/0.0
	12	0.58/0.78	0.36/0.61	0.06/0.31	0.0/0.0
	15	0.65/0.80	0.48/0.66	0.15/0.40	0.0/0.0
FashionMNIST	15	0.40/0.55	0.17/0.32	0.02/0.09	0.0/0.0
DBPedia	15	0.36/0.53	0.03/0.06	0.00/0.00	0.0/0.0
Random Drop					
EMNIST	9	0.47/0.66	0.40/0.68	0.39/0.68	0.35/0.64
	12	0.58/0.78	0.58/0.78	0.57/0.77	0.53/0.76
	15	0.65/0.80	0.66/0.82	0.64/0.81	0.65/0.82
FashionMNIST	15	0.40/0.55	0.39/0.53	0.38/0.53	0.35/0.50
DBPedia	15	0.36/0.53	0.39/0.54	0.31/0.47	0.34/0.45

convolution layers and then two linear layers, and train with Adam using learning rate $\eta_L = 0.001$.

6.6.2 Baselines: Perfect Knowledge and Random Dropping

To demonstrate the potential severity of a dropping attack, we first evaluate the best possible attack, where the adversary has perfect knowledge of a subset of k_N target clients from the beginning of the protocol, and drops every update originating from that subset throughout training. We show the results in Table 6.2. This demonstrates the power of a dropping attack, and provides a baseline to compare our full attack pipeline against. We also evaluate the effect of an adversary that selects uniformly random victim clients in Table 6.2. Table 6.2 allow us to make the following key observations: (i) When no dropping attack is underway, having more target clients improves accuracy. (ii) Even when fixing the fraction of dropped clients, smaller client counts are more vulnerable to attack. For example, on EMNIST, when $2/3$ of target clients are dropped, and $k = 9$, the accuracy drops to 23%, while it is 40% at $k = 15$. (iii) Dropping random clients does not lead to significant accuracy degradation on the target class: targeting clients is essential. (iv) When all target clients are successfully identified and dropped (that is, there is no data at all from the population), the accuracy on the population is 0, as expected. (v) On more difficult classification tasks, such as FashionMNIST and DBPedia, removing target clients results in a higher performance degradation than for EMNIST.

TABLE 6.3: Target client identification. Average number of clients correctly identified by Algorithm 5 at different rounds under COMM_PLAIN and COMM_ENC, $k_N = k$. On FashionMNIST and DBPedia all 15 target clients are identified at 50 and 20 rounds, respectively, for COMM_PLAIN, while the maximum number of clients identified under COMM_ENC is 11.75 at 70 rounds for DBPedia.

Dataset	Communication	k	Round Number T_N							
			5	10	15	20	30	40	50	70
EMNIST	COMM_PLAIN	9	1.75	5	7.50	7.75	8.50	8.75	8.75	8.50
		12	5	8.25	9.75	10.25	10.75	10.75	11.25	11.25
		15	4.25	9.50	11.50	12	14.25	14.25	14	14
	COMM_ENC	9	0.50	2.25	3.25	2.75	3.25	3.75	4	5.25
		12	2	2.75	3.25	3	3	4	5	5.25
		15	3	4	4	3.75	4.75	5.75	6.25	7
FashionMNIST	COMM_PLAIN	15	9	12	14	14.75	14.75	14.75	15	15
	COMM_ENC		5.5	6.50	8.0	8.50	9	10	10.50	11
DBPedia	COMM_PLAIN	15	8	13.25	13.75	15	15	15	15	15
	COMM_ENC		5.25	7	8	9	10	11.25	11.25	11.75

6.6.3 Client Identification Evaluation

Table 6.3 shows the average number of targeted clients correctly identified by Algorithm 5 under plain and encrypted communication. For the purpose of evaluating the identification accuracy we set $k_N = k$ in these experiments. The most immediate observation is that identification performs better under COMM_PLAIN than under COMM_ENC. This is expected as the adversary in COMM_PLAIN has access to all local model updates sent by clients and can keep track of the exact loss difference per client. Still, in the challenging scenario where the adversary only sees aggregated updates (under COMM_ENC), Client Identification performs reasonably well. For instance, on DBPedia an average of 11.75 out of 15 clients are identified at 70 rounds under COMM_ENC. Interestingly, for FashionMNIST and DBPedia, it is easier to identify target clients than on EMNIST (where Algorithm 5 identifies an average of 7 out of 15 clients at 70 rounds). One hypothesis for this phenomenon is that the class samples are more different in complex datasets, leading the global model to forget the target class in rounds with no participating target clients, resulting in significant loss increases for those rounds.

We also use Table 6.3 to select parameters for the targeted dropping attack and validate our analysis from Section 6.4.4. In COMM_PLAIN we need to wait approximately 15 rounds for all datasets to drop $2k/3$ clients, which roughly matches our analysis. In the COMM_ENC scenario, on the other hand, we see that more rounds are necessary for successful identification, as expected from Section 6.4.4. To identify between $k/3$ and $2k/3$ of the target clients, we need to wait between 30 and 50 rounds. Moreover, in many cases the identification accuracy tends to plateau in successive rounds. Therefore, we select round 30 as the starting point for the dropping attack in our experiments under COMM_ENC.

6.6.4 Targeted Dropping Evaluation

We now test the effectiveness of the full pipeline, where we begin by identifying clients most positively contributing towards target class accuracy, and then drop all the model updates coming from them. We measure our attacks' performance in Table 6.4. First, notice that attacks under COMM_PLAIN significantly compromise target population accuracy, closely approximating the perfect knowledge adversary for increasing values of k_N. For instance, on EMNIST, when $k = 12$ and $k_N = 8$, after 100 rounds the accuracy on the target class is on average 0.33, which is only 0.02 higher than the value obtained under the perfect knowledge attack. Moreover, if the adversary can interfere with a relatively large number of clients, $k = k_N$, the disruption is complete, as was in Table 6.2. We also observe that our attack in the COMM_PLAIN scenario vastly outperforms the strategy of randomly dropping the same number of clients (shown in Table 6.2), in all situations.

Our attacks' performance decreases in the COMM_ENCscenario, when the adversary's knowledge is limited, which is expected from the reduced Client Identification performance. We still observe a significant advantage in using our identification pipeline, over the random selection baseline. Moreover, these results highlight the trend mentioned in Section 6.6.3: on more complex tasks, such as FashionMNIST and DBPedia, the high identification accuracy leads to noticeably larger attack performance, than in EMNIST.

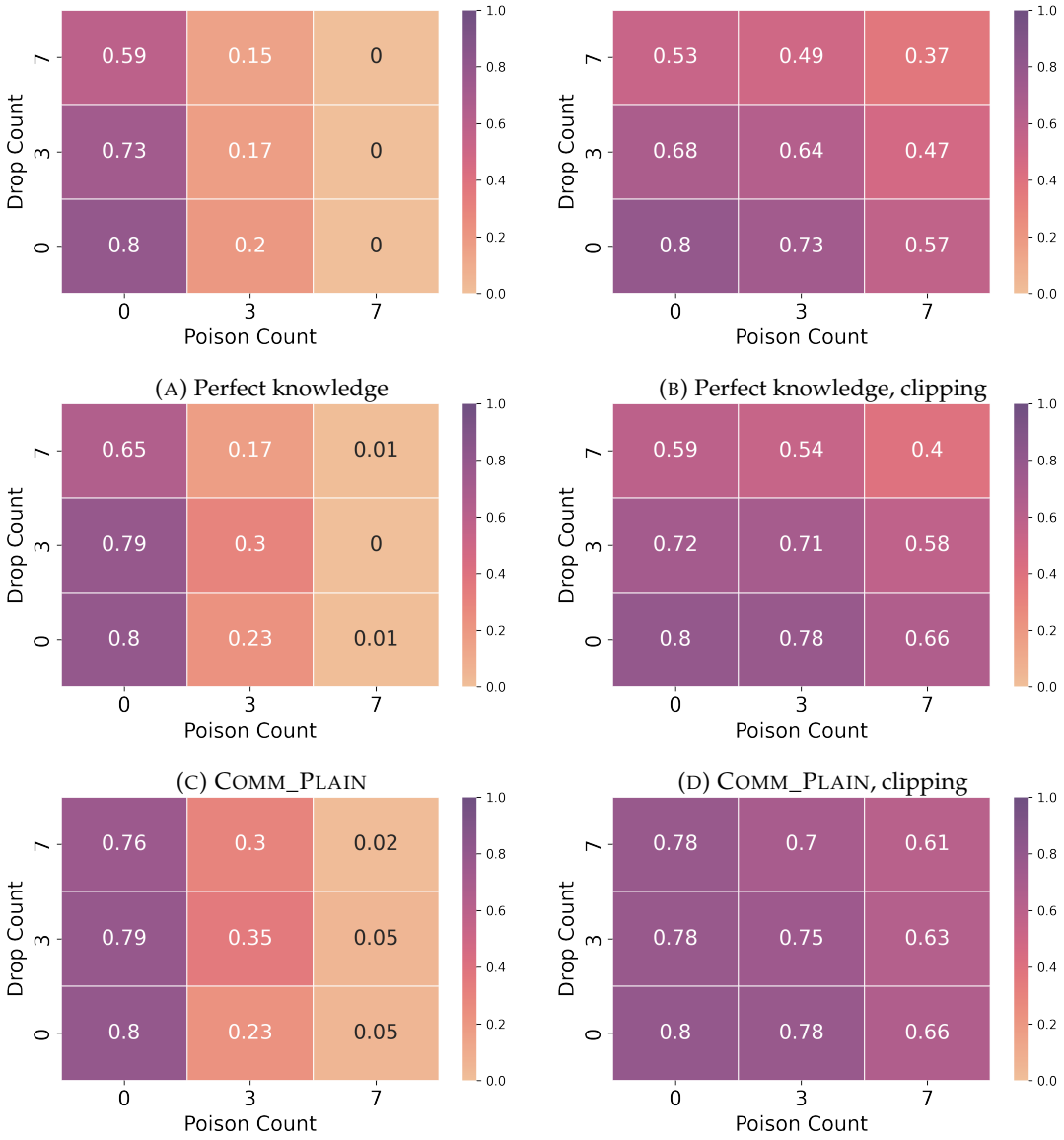
Given the targeted nature of our attack, in all the scenarios we consider, dropping the victim clients generally leads to very little degradation of the overall model performance. The overall accuracy degradation of the global model is on average 3.88% for COMM_PLAIN and 1% for COMM_ENC.

Tables 6.6, 6.8, 6.10, show the accuracy of the global model on the full test set under the same settings as the other experiments we run.

6.6.5 Impact of Model Poisoning and Targeted Dropping

When unmitigated, poisoning campaigns are known to be very effective at disrupting the accuracy of the model on the target class. We compared the effects of the model poisoning strategy and targeted dropping on the EMNIST dataset for the cases of perfect knowledge, COMM_PLAIN, and COMM_ENC in Figures 6.1a, 6.1c, and 6.1e respectively. These heatmaps show that, for different levels of k_N (number of dropped clients) and k_P (number of poisoned clients), the model replacement attack is more effective than targeted dropping, even when the adversary has perfect knowledge.

The results, however, are significantly different when Clipping, a standard poisoning defense [212, 81, 93], is applied to local model updates. Figures 6.1b, 6.1d, and 6.1f



(A) Perfect knowledge

(B) Perfect knowledge, clipping

(C) COMM_PLAIN

(D) COMM_PLAIN, clipping

(E) COMM_ENC

(F) COMM_ENC, clipping

FIGURE 6.1: Accuracy on target class 0 on EMNIST, for $k = 15$, $T = 100$, and varying number of dropped and poisoned clients under 3 scenarios: Perfect Knowledge, COMM_PLAIN, and COMM_ENC. Left results are without Clipping, and right results use a Clipping norm of 1. When no Clipping is used, model poisoning attack is devastating at small number of poisoned clients. With Clipping, model poisoning has lower impact, but the combination of targeted dropping and model poisoning results in significant degradation under all 3 scenarios.

TABLE 6.4: Targeted dropping attack, under COMM_PLAIN and COMM_ENC. Accuracy on target population at rounds $T/2$ and T, for $T = 100$ for EMNIST, $T = 200$ for FashionMNIST and DBPedia. The attack under COMM_PLAIN gets results close to perfect knowledge adversary, while under COMM_ENC the attack still improves upon random dropping.

COMM_PLAIN

Dataset	k	Number of Clients Dropped k_N			
		0	$k/3$	$2k/3$	k
EMNIST	9	0.47/0.66	0.25/0.49	0.09/0.18	0.00/0.00
	12	0.58/0.78	0.39/0.65	0.19/0.33	0.00/0.00
	15	0.65/0.80	0.50/0.74	0.26/0.50	0.02/0.02
FashionMNIST	15	0.40/0.55	0.24/0.23	0.02/0.03	0.00/0.00
DBPedia	15	0.36/0.53	0.10/0.01	0.00/0.00	0.00/0.00

COMM_ENC

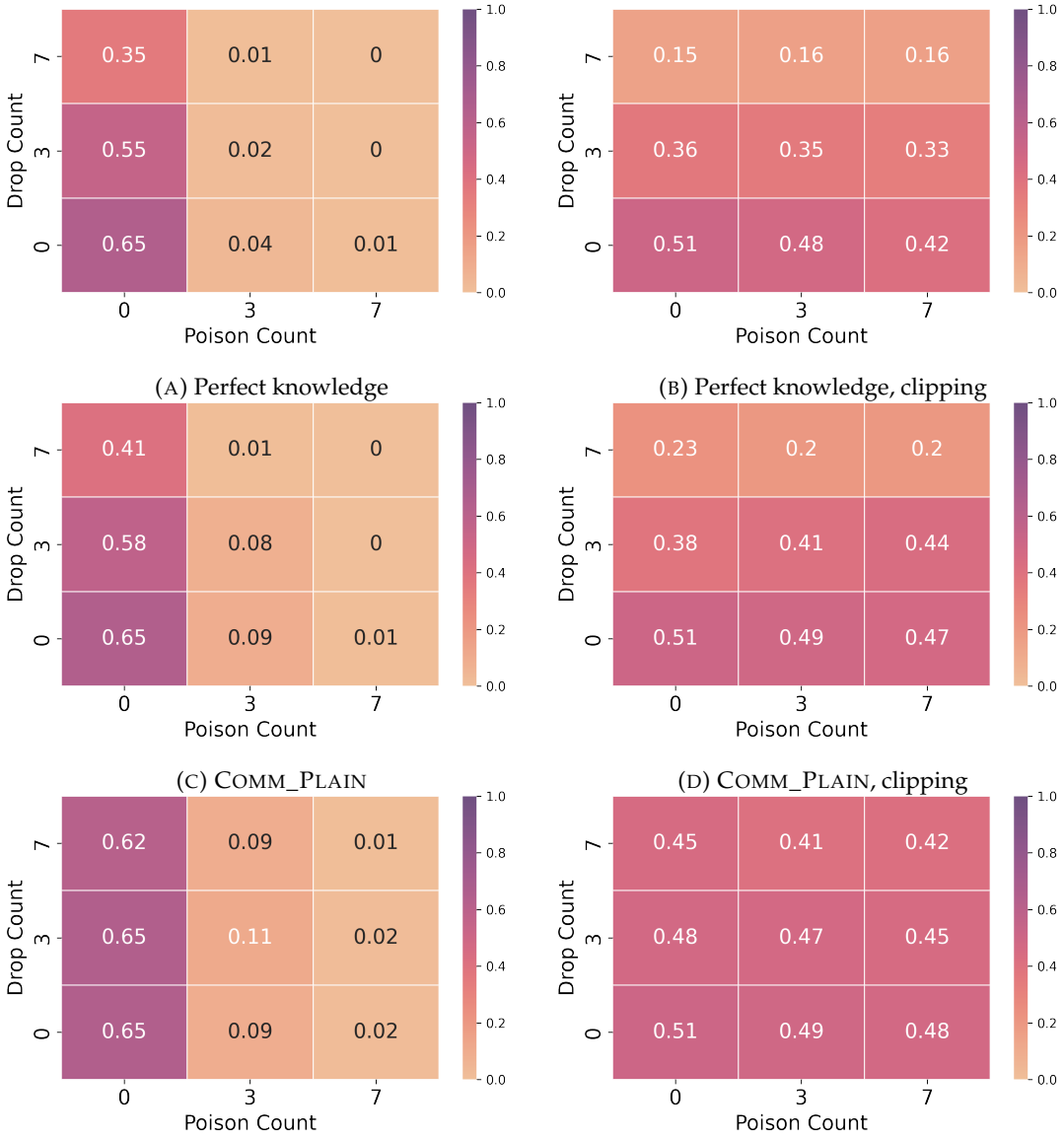
EMNIST	9	0.47/0.66	0.35/0.58	0.32/0.52	0.32/0.52
	12	0.58/0.78	0.50/0.72	0.48/0.69	0.45/0.67
	15	0.65/0.80	0.63/0.78	0.60/0.76	0.56/0.71
FashionMNIST	15	0.40/0.55	0.34/0.38	0.14/0.13	0.03/0.01
DBPedia	15	0.36/0.53	0.19/0.06	0.06/0.00	0.00/0.00

show the same set of experiments repeated with a clipping norm of 1. These heatmaps highlight that, while clipping lowers the impact of model poisoning, the combination of targeted dropping and model poisoning results in very noticeable targeted performance degradation even when clipping is used. For instance, under perfect knowledge, the original model accuracy on class 0 is 0.8. This becomes 0.53 when dropping 7 clients out of 15 and 0.57 when poisoning 7 additional clients, while the combination of dropping 7 and poisoning 7 clients results in 0.37 accuracy.

By repeating the experiments with $T = 50$, we can observe that it appears that targeted dropping is more effective than poisoning at low round counts, as shown in Fig. 6.2 When the model is in its early phase of learning, it needs to see some examples of the target class to begin learning it, and dropping significantly delays the arrival of these examples. Later in training, however, the difference between poisoning and dropping decreases.

6.6.6 Impact of Adversarial Visibility

Adversarial resources and capabilities may vary widely depending on the specific circumstances (nation states will have significantly larger capabilities than smaller players), so the adversary may not always be able to keep track of which clients can participate in each round, having to focus their resources on observing only a relatively small subset of the total client pool. In this section, we show that, assuming the adversary has



(A) Perfect knowledge

(B) Perfect knowledge, clipping

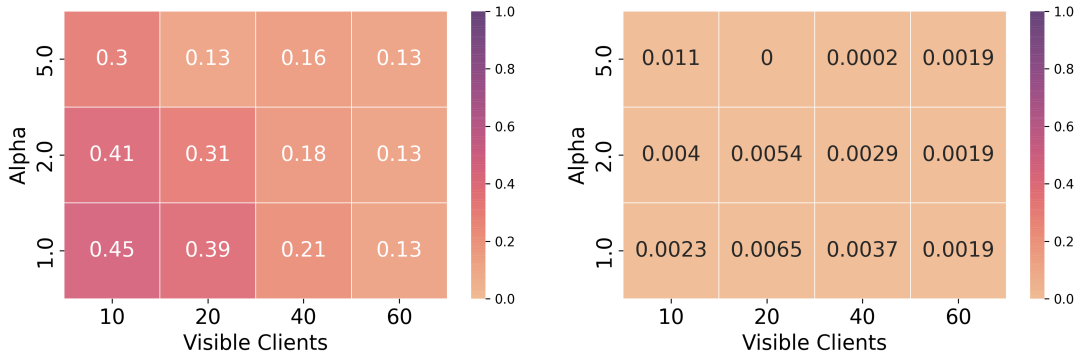
(C) COMM_PLAIN

(D) COMM_PLAIN, clipping

(E) COMM_ENC

(F) COMM_ENC, clipping

FIGURE 6.2: Accuracy on target class 0 on EMNIST, for $k = 15$, $T = 50$, and varying number of dropped and poisoned clients under 3 scenarios: perfect knowledge, COMM_PLAIN, and COMM_ENC. Left results are without clipping, and right results use a clipping norm of 1. When no clipping is used, model poisoning attack is devastating at small number of poisoned clients. With clipping, model poisoning. Results are similar with those in Figure 6.1.



(A) Drop (B) Drop + Poison

FIGURE 6.3: Accuracy on target class 0 on FashionMNIST, for $k = 15$, $T = 200$, $k_N = 10$, a poison count of 10, varying number of visible clients and α, under the COMM_ENC_LIMITED scenario.

visibility on a non-trivial fraction of the clients, our interference can still affect the target accuracy very significantly.

We model a targeted, resource-limited, adversary (COMM_ENC_LIMITED) by restricting their ability to observe the clients participating to the protocol to a fixed-size subset of the clients, chosen before the attack starts, sampled from a Dirichlet distribution with concentration parameter α. α will control how likely it is for the sampled visible list to include clients containing data of the target population, with larger αs leading to higher likelihood. We run the attack on FashionMNIST with a similar setup as in Table 6.4 under the COMM_ENC conditions. The results reported in Figure 6.3a show that, as expected, higher visibility fractions, and larger α lead to smaller target accuracy values. The network adversary is also still quite effective even when observing a small fraction of the clients, for instance achieving a $\approx 43.6\%$ relative target accuracy decrease when observing 20 clients with $\alpha = 2$. Similarly, Figure 6.3b shows that adding poisoning always leads to better targeted accuracy degradation.

6.6.7 Defense Evaluation

In Tables 6.5, 6.7, we evaluate the defense strategies under the COMM_PLAIN and COMM_ENCscenarios, using targeted dropping and poisoning attacks. We set the number of dropped and poisoned clients (k_N and k_P) to $2k/3$ for EMNIST, and $k/3$ for FashionMNIST and DBPedia. This parameter setting results in strong attacks, as the target accuracy is below 4% under targeted dropping and poisoning, when no mitigation is used.

Our UpSampling defense is extremely effective at protecting against targeted dropping,

TABLE 6.5: Accuracy on target class presented at rounds $T/2$ and T, under COMM_PLAIN setting. $T = 100$ for EMNIST, $T = 300$ for FashionMNIST and DBPedia. We consider both Targeted Dropping and Dropping + Poisoning scenarios.

k	Target Class	No Attack	Targeted Drop		Targeted Drop + Poison			
		FedAvg	FedAvg	UpSample	FedAvg	Clip	UpSample	Clip + UpSample
EMNIST								
9	0	0.47/ 0.66	0.09/ 0.18	0.18/ 0.44	0.00/ 0.00	0.00/ 0.06	0.01/ 0.00	0.03/ 0.31
	1	0.75/ 0.92	0.09/ 0.25	0.39/ 0.76	0.00/ 0.00	0.00/ 0.04	0.01/ 0.00	0.04/ 0.30
	9	0.43/ 0.56	0.01/ 0.17	0.09/ 0.40	0.00/ 0.00	0.00/ 0.01	0.01/ 0.00	0.00/ 0.23
12	0	0.58/ 0.78	0.19/ 0.33	0.40/ 0.66	0.00/ 0.00	0.04/ 0.09	0.00/ 0.01	0.12/ 0.40
	1	0.86/ 0.95	0.25/ 0.55	0.49/ 0.85	0.00/ 0.00	0.01/ 0.04	0.04/ 0.01	0.17/ 0.60
	9	0.53/ 0.67	0.06/ 0.31	0.28/ 0.52	0.00/ 0.00	0.00/ 0.02	0.02/ 0.01	0.07/ 0.29
15	0	0.65/ 0.80	0.26/ 0.50	0.47/ 0.71	0.00/ 0.00	0.04/ 0.06	0.00/ 0.03	0.22/ 0.35
	1	0.91/ 0.96	0.47/ 0.79	0.83/ 0.94	0.00/ 0.00	0.06/ 0.18	0.05/ 0.04	0.53/ 0.51
	9	0.65/ 0.75	0.15/ 0.39	0.39/ 0.59	0.00/ 0.00	0.02/ 0.03	0.02/ 0.07	0.19/ 0.40
FashionMNIST								
15	0	0.44/ 0.47	0.30/ 0.36	0.60/ 0.62	0.09/ 0.03	0.29/ 0.38	0.13/ 0.19	0.58/ 0.58
	1	0.93/ 0.95	0.89/ 0.93	0.95/ 0.96	0.12/ 0.02	0.77/ 0.77	0.16/ 0.28	0.92/ 0.93
	9	0.88/ 0.87	0.81/ 0.81	0.92/ 0.93	0.13/ 0.04	0.70/ 0.62	0.09/ 0.19	0.90/ 0.89
DBPedia								
15	0	0.45/ 0.54	0.11/ 0.10	0.75/ 0.82	0.00/ 0.00	0.00/ 0.00	0.02/ 0.01	0.65/ 0.77
	1	0.77/ 0.87	0.37/ 0.17	0.93/ 0.96	0.00/ 0.00	0.08/ 0.00	0.02/ 0.03	0.89/ 0.94
	9	0.52/ 0.60	0.24/ 0.17	0.84/ 0.91	0.00/ 0.00	0.00/ 0.00	0.02/ 0.02	0.64/ 0.62

and, when combined with Clipping, achieves high accuracy against the powerful targeted dropping and poisoning attack. For instance, on EMNIST with $k = 9$, the targeted dropping attack causes accuracy to change from 92% to 25% on class 1 under COMM_PLAIN, and the UpSampling defense restores the targeted accuracy to 76% on the same class. On the other hand, on DBPedia, targeted dropping reduces accuracy on class 1 from 87% to 17%, under COMM_PLAIN for $k = 15$. When the UpSampling defense is applied, the target accuracy exceeds the original one (96%) because clients contributing to that class are selected more frequently. Moreover, for the same setting, the combined targeted dropping and poisoning attack brings the target accuracy to 0, and neither Clipping nor UpSampling are sufficient to mitigate the attack. Using both defense strategies simultaneously, however, leads to a target accuracy of 94%.

The UpSampling defense is even more effective under the COMM_ENC setting, the most common deployment scenario for FL. Here, the attacker has access only to aggregated updates, resulting in slightly lower accuracy at Client Identification, while the server receives all local model updates and can reliably identify highly contributing clients. Due to the asymmetry in knowledge between attacker and defender, UpSampling improves target model accuracy against targeted dropping by an average (over the 3 classes) of 6.33% on EMNIST, 10.66% on FashionMNIST, and 24.66% on DBPedia for $k = 15$ compared to the original accuracy. Even under the targeted dropping and model poisoning attacks, the combined UpSampling defense and Clipping results in an average decrease

TABLE 6.6: Accuracy on full test set presented at rounds $T/2$ and T, under COMM_PLAIN setting. $T = 100$ for EMNIST, $T = 300$ for FashionMNIST and DBPedia. We consider both Targeted Dropping and Dropping + Poisoning scenarios.

k	Target Class	No Attack	Targeted Drop		Targeted Drop + Poison			
		FedAvg	FedAvg	UpSample	FedAvg	Clip	UpSample	Clip + UpSample
EMNIST								
9	0	0.92/ 0.95	0.88/ 0.90	0.89/ 0.93	0.86/ 0.87	0.85/ 0.88	0.86/ 0.87	0.86/ 0.91
	1	0.94/ 0.96	0.87/ 0.90	0.90/ 0.95	0.85/ 0.85	0.84/ 0.86	0.85/ 0.86	0.85/ 0.89
	9	0.91/ 0.93	0.87/ 0.89	0.88/ 0.92	0.86/ 0.87	0.86/ 0.87	0.86/ 0.87	0.86/ 0.90
12	0	0.93/ 0.96	0.89/ 0.92	0.91/ 0.95	0.86/ 0.87	0.86/ 0.89	0.86/ 0.87	0.87/ 0.92
	1	0.95/ 0.97	0.89/ 0.93	0.91/ 0.96	0.85/ 0.86	0.84/ 0.86	0.85/ 0.86	0.86/ 0.92
	9	0.92/ 0.95	0.87/ 0.91	0.90/ 0.93	0.86/ 0.87	0.86/ 0.87	0.68/ 0.68	0.87/ 0.90
15	0	0.94/ 0.96	0.90/ 0.93	0.92/ 0.95	0.86/ 0.87	0.86/ 0.88	0.86/ 0.88	0.89/ 0.91
	1	0.95/ 0.97	0.91/ 0.95	0.94/ 0.96	0.85/ 0.86	0.85/ 0.88	0.85/ 0.86	0.90/ 0.91
	9	0.94/ 0.95	0.88/ 0.92	0.91/ 0.94	0.86/ 0.87	0.86/ 0.87	0.87/ 0.88	0.88/ 0.92
FashionMNIST								
15	0	0.81/ 0.83	0.79/ 0.82	0.81/ 0.84	0.76/ 0.78	0.79/ 0.82	0.76/ 0.79	0.81/ 0.84
	1	0.83/ 0.85	0.82/ 0.85	0.82/ 0.85	0.74/ 0.76	0.81/ 0.83	0.75/ 0.78	0.82/ 0.84
	9	0.82/ 0.85	0.82/ 0.84	0.83/ 0.85	0.74/ 0.76	0.80/ 0.82	0.74/ 0.77	0.82/ 0.84
DBPedia								
15	0	0.93/ 0.94	0.91/ 0.91	0.95/ 0.96	0.89/ 0.90	0.90/ 0.90	0.89/ 0.90	0.94/ 0.96
	1	0.95/ 0.96	0.92/ 0.91	0.96/ 0.97	0.88/ 0.89	0.90/ 0.90	0.89/ 0.89	0.95/ 0.96
	9	0.93/ 0.94	0.91/ 0.91	0.95/ 0.96	0.89/ 0.90	0.89/ 0.90	0.89/ 0.90	0.93/ 0.94

of 5% on EMNIST and average increase of 9% on FashionMNIST and 16.66% on DBPedia over the 3 classes compared to the original accuracy.

Interestingly, classes with lower original accuracy benefit more from the UpSampling strategy, with improvements as high as 34% (on DBPedia for class 9, original accuracy is increased from 60% to 94% with UpSampling under targeted dropping). We also observe that as more clients hold examples from the target class (k increases in EMNIST), the UpSampling defense will have more clients to sample from, and achieves higher target accuracy.

Privacy-preserving FL So far, we have discussed settings in which the server receives local model updates in all rounds of the FL protocol. However, to protect client privacy, it is common to deploy privacy-preserving FL protocols, based on Multi-Party Computation (MPC), such as [26, 69]. In MPC implementations, multiple parties will be involved in aggregation and the server only receives the global model at the end of each iteration. The server has therefore the same knowledge as the network-level adversary under encrypted communication when running the client identification protocol from Algorithm 5.

In Table 6.9, we present attack and defense results for this challenging setting. Similarly to the previous tables, we set the parameters to $k_N = k_P = 2k/3$ for EMNIST, and $k_N = k_P = k/3$ for both FashionMNIST and DBPedia. While under this setting there

TABLE 6.7: Accuracy on target class presented at rounds $T/2$ and T, under COMM_ENC setting. $T = 100$ for EMNIST, $T = 300$ for FashionMNIST and DBPedia. We consider both Targeted Dropping and Dropping + Poisoning scenarios.

k	Target Class	No Attack	Targeted Drop		Targeted Drop + Poison			
		FedAvg	FedAvg	UpSample	FedAvg	Clip	UpSample	Clip + UpSample
EMNIST								
9	0	0.47/ 0.66	0.32/ 0.52	0.59/ 0.76	0.01/ 0.00	0.14/ 0.40	0.10/ 0.04	0.46/ 0.68
	1	0.75/ 0.92	0.58/ 0.76	0.88/ 0.96	0.04/ 0.00	0.24/ 0.52	0.16/ 0.04	0.67/ 0.88
	9	0.43/ 0.56	0.31/ 0.46	0.54/ 0.70	0.01/ 0.00	0.03/ 0.25	0.06/ 0.05	0.37/ 0.57
12	0	0.58/ 0.78	0.48/ 0.69	0.75/ 0.85	0.00/ 0.00	0.29/ 0.36	0.02/ 0.06	0.60/ 0.77
	1	0.86/ 0.95	0.77/ 0.91	0.94/ 0.97	0.00/ 0.00	0.47/ 0.43	0.20/ 0.12	0.82/ 0.92
	9	0.53/ 0.67	0.41/ 0.56	0.68/ 0.78	0.00/ 0.01	0.18/ 0.27	0.15/ 0.07	0.50/ 0.67
15	0	0.65/ 0.80	0.60/ 0.76	0.81/ 0.89	0.01/ 0.00	0.37/ 0.44	0.01/ 0.05	0.62/ 0.71
	1	0.91/ 0.96	0.88/ 0.94	0.96/ 0.97	0.04/ 0.00	0.60/ 0.48	0.12/ 0.06	0.90/ 0.94
	9	0.65/ 0.75	0.50/ 0.58	0.76/ 0.85	0.03/ 0.01	0.30/ 0.35	0.10/ 0.07	0.53/ 0.71
FashionMNIST								
15	0	0.44/ 0.47	0.40/ 0.39	0.69/ 0.71	0.13/ 0.02	0.38/ 0.36	0.25/ 0.17	0.64/ 0.69
	1	0.93/ 0.95	0.92/ 0.94	0.95/ 0.96	0.14/ 0.03	0.86/ 0.81	0.26/ 0.24	0.94/ 0.95
	9	0.88/ 0.87	0.85/ 0.82	0.93/ 0.94	0.20/ 0.05	0.77/ 0.59	0.19/ 0.23	0.91/ 0.92
DBPedia								
15	0	0.45/ 0.54	0.16/ 0.12	0.79/ 0.84	0.00/ 0.00	0.00/ 0.00	0.03/ 0.01	0.73/ 0.79
	1	0.77/ 0.87	0.39/ 0.22	0.95/ 0.97	0.00/ 0.00	0.07/ 0.00	0.10/ 0.05	0.92/ 0.95
	9	0.52/ 0.60	0.31/ 0.12	0.88/ 0.94	0.00/ 0.00	0.00/ 0.00	0.02/ 0.01	0.75/ 0.76

is essentially no difference in the effectiveness of the attacker models we consider, the defensive performance varies due to the limited knowledge. The results we observe for the UpSample and Clip + UpSample defensive mechanisms are, as we would expect, somewhere in between the COMM_ENC and COMM_PLAIN scenarios. For instance for EMNIST, with $k = 15$ on class 1 we obtain an average accuracy level of 0.84, considerably higher than the 0.51 obtained under COMM_PLAIN, but also not as high as the average of 0.94 obtained with COMM_ENC, which is essentially the same result obtained without any attack.

We have also evaluated the attacks and defenses against FL protocols using secure aggregation [26, 69]. Here, both the network-level adversary and the server only observe aggregated model updates. Even in this challenging setting, the server can identify clients contributing mostly to the target class, making UpSampling combined with Clipping quite effective. Results are omitted for space limitation.

Discussion Our evaluation shows that the proposed UpSampling defense is extremely effective at protecting against targeted dropping attacks. Combined with Clipping of model updates, the defense achieves high accuracy against powerful targeted dropping and poisoning attacks. The impact of the defense is stronger under COMM_ENC, when communication between clients and server is encrypted, and the network adversary

TABLE 6.8: Accuracy on full test set presented at rounds $T/2$ and T, under COMM_ENC setting. $T = 100$ for EMNIST, $T = 300$ for FashionMNIST and DBPedia. We consider both Targeted Dropping and Dropping + Poisoning scenarios.

k	Target Class	No Attack	Targeted Drop		Targeted Drop + Poison			
		FedAvg	FedAvg	UpSample	FedAvg	Clip	UpSample	Clip + UpSample
EMNIST								
9	0	0.92/ 0.95	0.91/ 0.94	0.93/ 0.95	0.86/ 0.87	0.88/ 0.92	0.87/ 0.87	0.91/ 0.94
	1	0.94/ 0.96	0.92/ 0.95	0.95/ 0.97	0.85/ 0.86	0.88/ 0.91	0.86/ 0.86	0.92/ 0.95
	9	0.91/ 0.93	0.90/ 0.92	0.93/ 0.95	0.86/ 0.87	0.86/ 0.90	0.87/ 0.88	0.90/ 0.93
12	0	0.93/ 0.96	0.92/ 0.95	0.94/ 0.96	0.86/ 0.87	0.90/ 0.92	0.87/ 0.88	0.92/ 0.95
	1	0.95/ 0.97	0.94/ 0.96	0.95/ 0.97	0.85/ 0.86	0.90/ 0.90	0.87/ 0.87	0.93/ 0.95
	9	0.92/ 0.95	0.91/ 0.94	0.94/ 0.96	0.86/ 0.87	0.88/ 0.90	0.88/ 0.88	0.91/ 0.94
15	0	0.94/ 0.96	0.93/ 0.95	0.95/ 0.96	0.86/ 0.87	0.90/ 0.92	0.87/ 0.88	0.92/ 0.95
	1	0.95/ 0.97	0.95/ 0.96	0.95/ 0.97	0.85/ 0.86	0.91/ 0.90	0.86/ 0.86	0.94/ 0.96
	9	0.94/ 0.95	0.92/ 0.94	0.94/ 0.96	0.87/ 0.87	0.89/ 0.91	0.88/ 0.88	0.92/ 0.95
FashionMNIST								
15	0	0.81/ 0.83	0.80/ 0.83	0.82/ 0.85	0.76/ 0.78	0.80/ 0.82	0.77/ 0.78	0.81/ 0.84
	1	0.83/ 0.85	0.83/ 0.85	0.82/ 0.85	0.75/ 0.76	0.82/ 0.83	0.76/ 0.78	0.82/ 0.85
	9	0.82/ 0.85	0.82/ 0.84	0.83/ 0.85	0.74/ 0.76	0.80/ 0.82	0.75/ 0.78	0.82/ 0.85
DBPedia								
15	0	0.93/ 0.94	0.91/ 0.91	0.95/ 0.96	0.89/ 0.90	0.90/ 0.90	0.90/ 0.90	0.95/ 0.96
	1	0.95/ 0.96	0.92/ 0.92	0.96/ 0.97	0.88/ 0.89	0.89/ 0.90	0.89/ 0.89	0.95/ 0.97
	9	0.93/ 0.94	0.92/ 0.91	0.95/ 0.96	0.89/ 0.90	0.89/ 0.90	0.89/ 0.89	0.93/ 0.95

only observes aggregate model updates. Encrypted communication in FL is the standard deployment method, but we would like to emphasize that encrypted communication is very important to reduce network adversaries' knowledge and should always be enabled.

To implement the defense and identify highly contributing clients, the server needs access to a trusted dataset from the population. While our server had knowledge of the attack population when running UpSampling, it is possible to collect data from several sensitive populations to identify poorly performing populations and improve their performance. Alternatively, the UpSampling strategy could also be implemented by upsampling clients from a trusted set of users. Other methods to establish trust in users include using Trusted Execution Environments (TEEs) to obtain attestation of software on client devices, or creating client reputation metrics [30].

We observed that under-represented classes (in terms of number of clients holding samples from those classes) are impacted more by our attacks. To alleviate this problem, the server could identify new clients with data from the populations of interest, and add them to the set of clients participating in the FL protocol. In cross-device FL settings, servers typically have access to a large number of clients, and can make decisions on expanding the set of participating clients to improve accuracy on under-represented populations.

TABLE 6.9: Accuracy on target class presented at rounds $T/2$ and T, under the MPC scenario. $T = 100$ for EMNIST, $T = 300$ for FashionMNIST and DBPedia. We consider both Targeted Dropping and Dropping + Poisoning scenarios.

k	Target Class	No Attack	Targeted Drop		Targeted Drop + Poison			
		FedAvg	FedAvg	UpSample	FedAvg	Clip	UpSample	Clip + UpSample
EMNIST								
9	0	0.47/ 0.66	0.32/ 0.52	0.39/ 0.67	0.01/ 0.00	0.14/ 0.40	0.02/ 0.00	0.25/ 0.55
	1	0.75/ 0.92	0.58/ 0.76	0.72/ 0.93	0.04/ 0.00	0.24/ 0.52	0.01/ 0.02	0.49/ 0.73
	9	0.43/ 0.56	0.31/ 0.46	0.38/ 0.60	0.01/ 0.00	0.03/ 0.25	0.03/ 0.00	0.14/ 0.50
12	0	0.58/ 0.78	0.48/ 0.69	0.50/ 0.75	0.00/ 0.00	0.29/ 0.36	0.02/ 0.02	0.40/ 0.62
	1	0.86/ 0.95	0.77/ 0.91	0.85/ 0.96	0.00/ 0.00	0.47/ 0.43	0.00/ 0.03	0.63/ 0.78
	9	0.53/ 0.67	0.41/ 0.56	0.54/ 0.72	0.00/ 0.01	0.18/ 0.27	0.02/ 0.00	0.27/ 0.51
15	0	0.65/ 0.80	0.60/ 0.76	0.69/ 0.84	0.01/ 0.00	0.37/ 0.44	0.01/ 0.04	0.43/ 0.63
	1	0.91/ 0.96	0.87/ 0.93	0.92/ 0.96	0.04/ 0.00	0.60/ 0.48	0.01/ 0.00	0.73/ 0.84
	9	0.64/ 0.75	0.50/ 0.58	0.59/ 0.80	0.03/ 0.01	0.30/ 0.35	0.02/ 0.00	0.38/ 0.61
FashionMNIST								
15	0	0.44/ 0.46	0.40/ 0.41	0.61/ 0.70	0.12/ 0.04	0.39/ 0.35	0.04/ 0.05	0.60/ 0.65
	1	0.93/ 0.95	0.92/ 0.94	0.95/ 0.96	0.11/ 0.03	0.86/ 0.81	0.14/ 0.06	0.93/ 0.95
	9	0.88/ 0.87	0.85/ 0.82	0.93/ 0.93	0.19/ 0.05	0.75/ 0.60	0.08/ 0.04	0.86/ 0.85
DBPedia								
15	0	0.45/ 0.54	0.14/ 0.12	0.70/ 0.81	0.00/ 0.00	0.00/ 0.00	0.00/ 0.00	0.61/ 0.77
	1	0.77/ 0.87	0.48/ 0.32	0.92/ 0.96	0.00/ 0.00	0.07/ 0.00	0.00/ 0.03	0.87/ 0.93
	9	0.52/ 0.60	0.30/ 0.13	0.82/ 0.91	0.00/ 0.00	0.00/ 0.00	0.00/ 0.00	0.53/ 0.70

6.7 Related Work

Attacks and defenses in federated learning Given the distributed nature of the training process in FL, poisoning attacks represent an even larger threat than in traditional Machine Learning systems. For instance, poisoning availability attacks have been shown effective in Federated Learning in recent work [68, 200]. Targeted model poisoning attacks impact a small population, or introduce a backdoor in the trained models to misclassify instances including the backdoor [20, 16, 212, 233]. Privacy attacks in FL are also a concern: [118] demonstrates property inference and data reconstruction attacks by disaggregating updates aggregated from users, assuming a malicious server; others have shown data reconstruction [253] and property inference attacks [165] from gradients.

Simultaneously, methods to defend FL protocols from adversaries have been proposed, such as Byzantine-resilient or trust-based aggregation rules [25, 143, 250, 3, 80, 211, 30, 141, 142]. However, [68] and [200] performed a systematic analysis of Byzantine-robust aggregation schemes, and showed that an adversary controlling compromised clients can launch poisoning availability attacks that bypass these defenses. Also, targeted poisoning attacks can bypass Byzantine-resilient aggregation, such as Krum [16]. Methods that can prevent model poisoning attacks include filtering of malicious gradients for availability attacks [68, 200, 239] and gradient clipping for targeted attacks [212, 16].

TABLE 6.10: Accuracy on full test set presented at rounds $T/2$ and T, under the MPC scenario. $T = 100$ for EMNIST, $T = 300$ for FashionMNIST and DBPedia. We consider both Targeted Dropping and Dropping + Poisoning scenarios.

k	Target Class	No Attack	Targeted Drop		Targeted Drop + Poison			
		FedAvg	FedAvg	UpSample	FedAvg	Clip	UpSample	Clip + UpSample
EMNIST								
9	0	0.92/ 0.95	0.91/ 0.94	0.91/ 0.95	0.86/ 0.87	0.88/ 0.92	0.87/ 0.87	0.89/ 0.93
	1	0.94/ 0.96	0.92/ 0.95	0.93/ 0.96	0.85/ 0.86	0.88/ 0.91	0.85/ 0.86	0.90/ 0.93
	9	0.91/ 0.93	0.90/ 0.92	0.91/ 0.94	0.86/ 0.87	0.86/ 0.90	0.87/ 0.87	0.88/ 0.93
12	0	0.93/ 0.96	0.92/ 0.95	0.92/ 0.95	0.86/ 0.87	0.90/ 0.92	0.86/ 0.87	0.91/ 0.94
	1	0.95/ 0.97	0.94/ 0.96	0.95/ 0.97	0.85/ 0.86	0.90/ 0.90	0.85/ 0.86	0.91/ 0.94
	9	0.92/ 0.95	0.91/ 0.94	0.93/ 0.95	0.86/ 0.87	0.88/ 0.90	0.87/ 0.68	0.89/ 0.93
15	0	0.94/ 0.96	0.93/ 0.95	0.94/ 0.96	0.86/ 0.87	0.90/ 0.92	0.87/ 0.88	0.91/ 0.94
	1	0.95/ 0.97	0.95/ 0.96	0.95/ 0.97	0.85/ 0.86	0.91/ 0.90	0.85/ 0.86	0.92/ 0.95
	9	0.94/ 0.95	0.92/ 0.94	0.93/ 0.96	0.87/ 0.87	0.89/ 0.91	0.87/ 0.87	0.90/ 0.94
FashionMNIST								
15	0	0.81/ 0.83	0.80/ 0.83	0.82/ 0.85	0.76/ 0.78	0.80/ 0.82	0.75/ 0.77	0.81/ 0.84
	1	0.83/ 0.85	0.83/ 0.85	0.82/ 0.85	0.74/ 0.76	0.82/ 0.83	0.75/ 0.76	0.82/ 0.85
	9	0.82/ 0.85	0.82/ 0.84	0.83/ 0.85	0.74/ 0.76	0.80/ 0.82	0.74/ 0.76	0.81/ 0.84
DBPedia								
15	0	0.93/ 0.94	0.91/ 0.91	0.95/ 0.96	0.89/ 0.89	0.90/ 0.90	0.89/ 0.90	0.94/ 0.96
	1	0.95/ 0.96	0.93/ 0.92	0.96/ 0.97	0.88/ 0.89	0.89/ 0.90	0.88/ 0.89	0.95/ 0.96
	9	0.93/ 0.94	0.92/ 0.91	0.95/ 0.96	0.89/ 0.90	0.89/ 0.90	0.89/ 0.89	0.92/ 0.95

Network-level attacks and defenses. Packet dropping attacks have been studied for network-level adversaries who can manipulate traffic in multiple ways. Some examples include physical-layer jamming attacks in wireless networks [244, 18], router compromise [49, 185], transport-level attacks [104], and BGP hijacking attacks [46]. Multiple applications in various domains are seriously impacted by network-level adversaries, such as: cryptocurrenices [10], payment-channel networks [235], and connected cars [101].

Defenses against these attacks at the network layer have been proposed. Among these, monitoring and detecting faulty paths in the network [14] can prevent against Byzantine attacks in wireless networks; creating resilient overlays [38, 157] enforces correct packet delivery to prevent against routing attacks and compromise of the nodes in the overlay. Our FL-specific UpSampling defense modifies the client sampling procedure at the server with the goal of selecting a set of clients that increases the accuracy on the target learning tasks.

We believe that our server-side defense can complement these existing network-level defenses, whose effectiveness still needs to be evaluated in FL protocols. The challenges of federated learning on best-effort networks have also been addressed, by reapplying old gradients from high latency clients [79, 167] and by improving processing time in networks [187, 119]. While techniques for handling high latency clients may help when

clients are dropped, dropping prevents clients from ever participating again, and repeating their updates will perform worse over the course of training.

6.8 Discussion and Conclusion

In this study, we consider the effects that a network-level adversary, capable of preventing model updates from distributed clients reach the server, can have on the final model accuracy on a target population, in cross-device FL. Our work proposes a new attack, based on a new procedure of identifying the set of clients mostly contributing towards a target task of interest to the adversary, and performing targeted disruptions of their communications. We evaluate the attack on multiple scenarios, considering classification tasks across different data modalities and machine learning models, and assuming different levels of visibility that the adversary has on the system. Under all settings, we show that the targeted dropping attack is quite effective in preventing learning on specific tasks, and that its effect can be augmented by concurrent poisoning campaigns.

We also explore defensive approaches, and find that our UpSampling mechanism, centered around the same Client Identification scheme, can be extremely successful, when paired with encrypted communications and model updates Clipping, in boosting the performance of a FL system on a target task, even when under heavy attack.

Chapter 7

Conclusion and Future Directions

This thesis introduced a variety of offensive security techniques to highlight the vulnerabilities of the training phase of modern machine learning models in security sensitive contexts. In this work, we focused on the vulnerabilities introduced by the increasing reliance on large-scale, potentially untrusted data sources and outsourced computation for model training. Our work has demonstrated the feasibility and effectiveness of sophisticated poisoning attacks against machine learning models in security-critical applications, while also proposing initial steps towards robust defenses.

The primary contributions presented in this thesis can be summarized as follows:

- We introduced explanation-guided poisoning, a novel approach to backdoor attacks that leverages AI model explanation techniques (XAI) to generate effective triggers with minimal impact on the model's normal behavior. This general approach was successfully specialized and applied to both malware detection and network intrusion detection systems, demonstrating its versatility across different security domains and data modalities.
- We showed that backdoor attacks can be executed under highly constrained conditions that mirror real-world deployment scenarios. These attacks operate without control over training labels, make no assumptions about the victim model's architecture, and respect the semantic constraints of complex data types such as executable binaries and aggregated network flows.
- We developed mitigation strategies that leverage the peculiarities of the cybersecurity domain to remove costly and impractical assumptions. We showed the effectiveness of this defensive approach on the attacks we designed, and
- We developed techniques for targeted network-level interference in Federated Learning protocols, achieving effects similar to targeted poisoning by strategically manipulating the exchange of model updates.

7.1 Future Directions

The dynamism of the machine learning landscape ensures a variety of possible future avenues for research on training-time integrity violations, poisoning attacks, and the overall viability of ML models in cybersecurity contexts.

7.1.1 Offensive Research

Attacking different learning paradigms. As an instance of these dynamic trends, a topic that deserves the attention of researchers in the near future is training-time attacks against learning paradigms different from supervised learning, such as reinforcement learning [214] and semi-supervised learning [158].

In this work we explored integrity violations against supervised learning systems because of their large utilization in security contexts. However, the progressive scaling up of training data requirements foreshadows an environment where more and more models are trained with methods that can leverage cheap and abundant unlabeled data. Recent research has started exploring the issue of poisoning attacks in reinforcement learning [109, 53, 178] and semi-supervised learning [33], however the literature on applications of these methods to cybersecurity tasks and their pitfalls is very limited.

Poisoning local (and remote) knowledge bases. Another example is the recent trend towards the use of large language models (LLMs) augmented with retrieval or search capabilities, also known as *retrieval augmented generation* (RAG) [123]. These systems are designed to address inherent limitations in auto-regressive language modeling such as hallucinations, distribution shift of information, and prohibitive costs associated with domain-specific fine-tuning and re-training. In RAG systems, a retriever component (usually composed of pre-trained encoders) is tasked with searching for relevant textual passages in a database and providing them as context to the generator component.

Allowing RAG systems to operate on untrusted on-device documents, or even on web search results, opens up a complete new avenue for poisoning attacks. Initial research exploring this threat vector [255, 168, 40, 153] surfaced the possibility for the adversary to achieve results similar to training-time data poisoning attacks without the need to actually tamper with the training phase of the models, by performing indirect prompt injection on the LLM.

This threat is particularly impactful, as an increasing number of language model assistants are provided with the ability to perform API calls (also known as tool usage) to enact actual changes on external applications and systems. Therefore, an attacker able to control the output of a RAG system, may be able to achieve even broader effects than

the straightforward spread of misinformation. Thus, additional research on these attacks, and possible counter measures, would surely be beneficial, especially when RAG systems will start being used for cybersecurity applications such as log monitoring.

7.1.2 Defensive Research

Layered defenses. Protecting modern training pipelines is quite challenging: whether because of the large volumes of training data involved, or because of the inherent complexity of these pipelines, such as those used in modern (giant) language and multimodal models, or because of the adversarial nature of the cybersecurity environment. Therefore, it is unlikely that solutions based on a single mitigation approach will be successful in the long run. A likely candidate strategy for defending critical training pipelines will therefore be to layer multiple mitigation techniques at different levels of the training and deployment phases – a technique also known as *defense in depth*.

Different types of defenses approaches can be categorized in macro areas. Data provenance tracking and sanitization is performed before the training process, and then progressively on the data used by the models during their normal activity, and is critical to minimize the probability of successful data poisoning attacks. Similarly, tight vetting of the code dependencies and potentially any pre-trained models used – through model signatures and verification primitives implemented in common libraries – are necessary to mitigate supply-chain attacks.

During the training phase itself, adversarial training and alignment training, while not designed to prevent poisoning directly, are effective techniques to reduce the intrinsic vulnerabilities of the models at deployment time. Alongside robust training schemes, practices such as data provenance tracking and input/output filtering for generative models are starting to become standard in the industry. Finally, infrastructure should be in place to perform trace-back and forensics analyses [197], in case a poisoning attack is suspected. Future research should focus on additional mitigation strategies that can be easily combined with the existing methods while prioritizing the preservation of the system's utility.

Defending modern language models. Regarding specifically the defense of LLM-based systems, recent years have witnessed a concentrated research effort to enhance the trustworthiness of deployed models through alignment training [208, 66, 17]. This approach has been implemented for most currently used LLMs and aims to ensure that model outputs align with human values and intentions. While this type of training augmentations surely improve the trustworthiness of the models for typical users, the attacks mentioned in the paragraph above, together with other classical poisoning threats

applied to the reinforcement learning with human feedback process [219], highlight the need for additional security measures in adversarial settings [95].

One example of these mitigation strategies would be the use of well-trained language models to perform perplexity analysis – measuring the likelihood that a textual sample comes from the same distribution used for training the language model – to isolate outliers in fine-tuning data and knowledge bases accessed by RAG systems. The effectiveness of this approach lies in its ability to be implemented in parallel with existing defenses, minimizing adverse effects on the utility of the defended models.

Another open problem in this area is the defense of agentic systems, where one (or multiple models) are allowed to freely plan a course of action and interact with external systems (and each other) in response to some query, or to achieve some objective, specified by the user. The study of the security properties of these systems, both from an offensive and defensive perspective, is still in its infancy. Consequently, it is interesting to investigate the applicability and effectiveness of existing techniques in this context, as well as to identify potential new security concerns inherent to the peculiarities of agentic design. Examples of these objectives include developing novel anomaly detection methods tailored to multi-agent interactions, and investigating the potential of formal verification approaches to guarantee certain security properties in complex agentic environments.

Bibliography

[1] Kendra Albert, Jon Penney, Bruce Schneier, and Ram Shankar Siva Kumar. *Politics of Adversarial Machine Learning*. SSRN Scholarly Paper. Rochester, NY, 2020. DOI: 10.2139/ssrn.3547322.

[2] *AlienVault - Open Threat Exchange*. https://otx.alienvault.com/.

[3] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. "Byzantine Stochastic Gradient Descent". In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[4] Brandon Amos, Hamilton Turner, and Jules White. "Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale". In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. July 2013, pp. 1666–1671. DOI: 10.1109/IWCMC.2013.6583806.

[5] Hyrum S. Anderson and Phil Roth. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models". In: *arXiv:1804.04637 [cs]* (Apr. 2018). arXiv: 1804.04637 [cs].

[6] Emre Kiciman Andrew Marshall Jugal Parikh and Ram Shankar Siva Kumar. *Threat Modeling AI/ML Systems and Dependencies*. https://learn.microsoft.com/en-us/security/engineering/threat-modeling-aiml. 2022.

[7] Mihael Ankerst, Markus M. Breunig, Hans peter Kriegel, and Jörg Sander. "OPTICS: Ordering Points To Identify the Clustering Structure". In: ACM Press, 1999, pp. 49–60.

[8] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. "Building a Dynamic Reputation System for DNS". In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security'10. Washington, DC: USENIX Association, 2010, p. 18. ISBN: 8887666655554.

[9] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. "Detecting Malware Domains at the Upper DNS Hierarchy". In: *Proceedings of the 20th USENIX Conference on Security*. SEC'11. San Francisco, CA: USENIX Association, 2011, p. 27.

[10] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 375–392. DOI: 10.1109/SP.2017.29.

[11] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, and Mirco Marchetti. "Addressing Adversarial Attacks Against Security Systems Based on Machine Learning". In: *2019 11th International Conference on Cyber Conflict (CyCon)*. Vol. 900. May 2019, pp. 1–18. DOI: 10.23919/CYCON.2019.8756865.

[12] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket". In: *Proceedings 2014 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2014. ISBN: 978-1-891562-35-8. DOI: 10.14722/ndss.2014.23247.

[13] MITRE ATLAS. *VirusTotal Poisoning 2020*. 1999. URL: https://atlas.mitre.org/studies/AML.CS0002.

[14] Baruch Awerbuch, Reza Curtmola, David Holmer, Cristina Nita-Rotaru, and Herbert Rubens. "ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks". In: *ACM Trans. Inf. Syst. Secur.* 10.4 (2008), 6:1–6:35. DOI: 10.1145/1284680.1341892. URL: https://doi.org/10.1145/1284680.1341892.

[15] Md. Ahsan Ayub, William A. Johnson, Douglas A. Talbert, and Ambareen Siraj. "Model Evasion Attack on Intrusion Detection Systems using Adversarial Machine Learning". In: *2020 54th Annual Conference on Information Sciences and Systems (CISS)*. 2020, pp. 1–6. DOI: 10.1109/CISS48834.2020.1570617116.

[16] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. "How to backdoor federated learning". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2938–2948.

[17] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. *Constitutional AI: Harmlessness from AI Feedback*. Dec. 2022. arXiv: 2212.08073 [cs].

[18] Y. Ozan Basciftci, Fangzhou Chen, Joshua Weston, Ron Burton, and C. Emre Koksal. "How Vulnerable Is Vehicular Communication to Physical Layer Jamming Attacks?" In: *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*. 2015, pp. 1–5. DOI: 10.1109/VTCFall.2015.7390968.

[19] *Batched Coupon Collector Problem*. https://mathoverflow.net/questions/229060/batched-coupon-collector-problem. 2016. URL: https://mathoverflow.net/questions/229060/batched-coupon-collector-problem.

[20] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. "Model Poisoning Attacks in Federated Learning". In: *Neurips Workshop on Security in Machine Learning*. 2018, p. 8.

[21] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. "Evasion Attacks against Machine Learning at Test Time". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Vol. 7908. Springer, 2013, pp. 387–402. DOI: 10.1007/978-3-642-40994-3_25.

[22] Battista Biggio, Blaine Nelson, and Pavel Laskov. "Poisoning Attacks against Support Vector Machines". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, June 2012, pp. 1467–1474. ISBN: 978-1-4503-1285-1.

[23] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. "Poisoning Behavioral Malware Clustering". In: *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop - AISec '14*. Scottsdale, Arizona, USA: ACM Press, 2014, pp. 27–36. ISBN: 978-1-4503-3153-1. DOI: 10.1145/2666652.2666666.

[24] Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. "Layer-wise relevance propagation for neural networks with local renormalization layers". In: *Artificial Neural Networks and Machine Learning–ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II 25*. Springer. 2016, pp. 63–71.

[25] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent". In: *Neurips 2017*. 2017, p. 11.

[26] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, Oct. 2017, pp. 1175–1191. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3133982.

[27] Stavroula Bourou, Andreas El Saer, Terpsichori-Helen Velivassaki, Artemis Voulkidis, and Theodore Zahariadis. "A Review of Tabular Data Synthesis Using GANs on an IDS Dataset". In: *Information* 12.9 (Sept. 2021), p. 375. ISSN: 2078-2489. DOI: 10.3390/info12090375.

[28] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. "SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics". In: *19th USENIX Security Symposium (USENIX Security 10)*. Washington, DC: USENIX Association, Aug. 2010. URL: https://www.usenix.org/conference/usenixsecurity10/sepia-privacy-preserving-aggregation-multi-domain-network-events-and.

[29] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: *Advances in Knowledge Discovery and Data Mining*. Vol. 7819. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172. ISBN: 978-3-642-37455-5 978-3-642-37456-2. DOI: 10.1007/978-3-642-37456-2_14.

[30] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. "FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping". In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. URL: https://www.ndss-symposium.org/ndss-paper/fltrust-byzantine-robust-federated-learning-via-trust-bootstrapping/.

[31] Xiaoyu Cao and Neil Zhenqiang Gong. "Mitigating Evasion Attacks to Deep Neural Networks via Region-Based Classification". In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC '17. Orlando, FL, USA: Association for Computing Machinery, 2017, pp. 278–287. ISBN: 9781450353458. DOI: 10.1145/3134600.3134606. URL: https://doi.org/10.1145/3134600.3134606.

[32] Nicholas Carlini. *Adversarial Machine Learning Reading List*. https://nicholas.carlini.com/writing/2 machine-learning-reading-list.html.

[33] Nicholas Carlini. "Poisoning the Unlabeled Dataset of {Semi-Supervised} Learning". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 1577–1592. ISBN: 978-1-939133-24-3.

[34] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. "On Evaluating Adversarial Robustness". In: *arXiv:1902.06705 [cs, stat]* (Feb. 2019). arXiv: 1902.06705 [cs, stat].

[35] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. "Membership Inference Attacks From First Principles". In: *2022 IEEE Symposium on Security and Privacy (SP)*. May 2022, pp. 1897–1914. DOI: 10.1109/SP46214.2022.9833649.

[36] Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr.

"Poisoning Web-Scale Training Datasets is Practical". In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2024, pp. 176–176.

[37] Ero Carrera. *Erocarrera/Pefile*. https://github.com/erocarrera/pefile.

[38] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. "Secure Routing for Structured Peer-to-Peer Overlay Networks". In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2003), pp. 299–314. ISSN: 0163-5980. DOI: 10.1145/844128.844156. URL: https://doi.org/10.1145/844128.844156.

[39] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. "A Survey on Adversarial Attacks and Defences". In: *CAAI Transactions on Intelligence Technology* 6.1 (2021), pp. 25–45. ISSN: 2468-2322. DOI: 10.1049/cit2.12028.

[40] Harsh Chaudhari, Giorgio Severi, John Abascal, Matthew Jagielski, Christopher A. Choquette-Choo, Milad Nasr, Cristina Nita-Rotaru, and Alina Oprea. *Phantom: General Trigger Attacks on Retrieval Augmented Language Generation*. May 2024. arXiv: 2405.20485 [cs].

[41] Harsh Chaudhari, Giorgio Severi, Alina Oprea, and Jonathan Ullman. "Chameleon: Increasing Label-Only Membership Leakage with Adaptive Poisoning". In: *The Twelfth International Conference on Learning Representations (ICLR)*. Jan. 2024.

[42] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. "Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering". In: *Workshop on Artificial Intelligence Safety*. CEUR-WS, Jan. 2019.

[43] Haipeng Chen, Sushil Jajodia, Jing Liu, Noseong Park, Vadim Sokolov, and V. S. Subrahmanian. "FakeTables: Using GANs to Generate Functional Dependency Preserving Tables with Bounded Real Data". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. Macao China: International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 2074–2080. DOI: 10.24963/ijcai.2019/287. URL: https://doi.org/10.24963/ijcai.2019/287.

[44] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. "Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning". In: *arXiv:1712.05526 [cs]* (Dec. 2017). arXiv: 1712.05526 [cs].

[45] Alesia Chernikova and Alina Oprea. "FENCE: Feasible Evasion Attacks on Neural Networks in Constrained Environments". In: *ACM Transactions on Privacy and Security* 25.4 (July 2022), 34:1–34:34. ISSN: 2471-2566. DOI: 10.1145/3544746.

[46] Shinyoung Cho, Romain Fontugne, Kenjiro Cho, Alberto Dainotti, and Phillipa Gill. "BGP hijacking classification". In: *2019 Network Traffic Measurement and Analysis Conference (TMA)*. 2019, pp. 25–32. DOI: 10.23919/TMA.2019.8784511.

[47] Zheng Leong Chua, Shiqi Shen, Prateek Saxena, and Zhenkai Liang. "Neural Nets Can Learn Function Type Signatures From Binaries". In: *USENIX Security Symposium*. 2017, p. 19.

[48] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A. Moser, Alina Oprea, Battista Biggio, Marcello Pelillo, and Fabio Roli. "Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning". In: *ACM Computing Surveys* 55.13s (Dec. 2023), pp. 1–39. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3585385.

[49] *A Cisco Router Bug Has Massive Global Implications*. https://www.wired.com/story/cisco-router-bug-secure-boot-trust-anchor. 2019.

[50] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. "EMNIST: Extending MNIST to Handwritten Letters". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. May 2017, pp. 2921–2926. DOI: 10.1109/IJCNN.2017.7966217.

[51] "Coupon Collector's Problem". In: *Wikipedia* (Nov. 2021).

[52] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. "Casting out Demons: Sanitizing Training Data for Anomaly Sensors". In: *2008 IEEE Symposium on Security and Privacy (Sp 2008)*. Oakland, CA, USA: IEEE, May 2008, pp. 81–95. ISBN: 978-0-7695-3168-7. DOI: 10.1109/SP.2008.11.

[53] Jing Cui, Yufei Han, Yuzhe Ma, Jianbin Jiao, and Junge Zhang. "BadRL: Sparse Targeted Backdoor Attack against Reinforcement Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. Mar. 2024, pp. 11687–11694. DOI: 10.1609/aaai.v38i10.29052.

[54] *CylancePROTECT Malware Execution Control*.

[55] Gianni D'Angelo and Francesco Palmieri. "Network Traffic Classification Using Deep Convolutional Recurrent Autoencoder Neural Networks for Spatial–Temporal Features Extraction". In: *Journal of Network and Computer Applications* 173 (Jan. 2021), p. 102890. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2020.102890.

[56] Tristan Deleu, António Góis, Chris Chinenye Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. "Bayesian Structure Learning with Generative Flow Networks". In: *The 38th Conference on Uncertainty in Artificial Intelligence*. 2022.

[57] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. "Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks". In: *USENIX Security Symposium*. USENIX Security. 2019, pp. 321–338. ISBN: 978-1-939133-06-9.

[58] *Detonating a Bad Rabbit: Windows Defender Antivirus and Layered Machine Learning Defenses*. https://www.microsoft.com/security/blog/2017/12/11/detonating-a-bad-rabbit-windows-defender-antivirus-and-layered-machine-learning-defenses/. Dec. 2017.

[59] Nagaraju Devarakonda, Srinivasulu Pamidi, V. Valli Kumari, and A. Govardhan. "Intrusion Detection System using Bayesian Network and Hidden Markov Model". In: *Procedia Technology* 4 (2012). 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012, pp. 506–514. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2012.05.081. URL: https://www.sciencedirect.com/science/article/pii/S221201731200360X.

[60] Robert Dorfman. "The detection of defective members of large populations". In: *The Annals of Mathematical Statistics* 14.4 (1943), pp. 436–440.

[61] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. Oct. 2020.

[62] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. "Characterization of Encrypted and VPN Traffic Using Time-related Features:" in: *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*. Rome, Italy: SCITEPRESS - Science and and Technology Publications, 2016, pp. 407–414. ISBN: 978-989-758-167-0. DOI: 10.5220/0005740704070414.

[63] Thomas F. Dullien. "Weird Machines, Exploitability, and Provable Unexploitability". In: *IEEE Transactions on Emerging Topics in Computing* (2018), pp. 1–1. ISSN: 2168-6750, 2376-4562. DOI: 10.1109/TETC.2017.2785299.

[64] Justin Engelmann and Stefan Lessmann. "Conditional Wasserstein GAN-based Oversampling of Tabular Data for Imbalanced Learning". In: *Expert Systems with Applications* 174 (Jan. 2021), p. 114582. DOI: 10.1016/j.eswa.2021.114582.

[65] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[66] Owain Evans, Owen Cotton-Barratt, Lukas Finnveden, Adam Bales, Avital Balwit, Peter Wills, Luca Righetti, and William Saunders. *Truthful AI: Developing and governing AI that does not lie*. 2021. arXiv: 2110.06674 [cs.CY].

[67] Ju Fan, Junyou Chen, Tongyu Liu, Yuwei Shen, Guoliang Li, and Xiaoyong Du. "Relational data synthesis using generative adversarial networks: a design space

exploration". In: *Proceedings of the VLDB Endowment* 13 (Aug. 2020), pp. 1962–1975. DOI: 10.14778/3407790.3407802.

[68] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning". en. In: *USENIX Security*. 2020, p. 18.

[69] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, and Shaza Zeitouni. "SAFELearn: Secure Aggregation for Private FEderated Learning". In: *2021 IEEE Security and Privacy Workshops (SPW)*. May 2021, pp. 56–62. DOI: 10.1109/SPW53761.2021.00017.

[70] S. García, M. Grill, J. Stiborek, and A. Zunino. "An Empirical Comparison of Botnet Detection Methods". In: *Computers and Security* 45 (Sept. 2014), pp. 100–123. ISSN: 0167-4048. DOI: 10.1016/j.cose.2014.05.011.

[71] Joseph Gastwirth. "The Estimation of the Lorenz Curve and Gini Index". In: *The Review of Economics and Statistics* 54 (Feb. 1972), pp. 306–16. DOI: 10.2307/1937992.

[72] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256.

[73] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *ICLR*. 2015. arXiv: 1412.6572.

[74] Kathrin Grosse, Lukas Bieringer, Tarek R. Besold, Battista Biggio, and Katharina Krombholz. "Machine Learning Security in Industry: A Quantitative Survey". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 1749–1762. ISSN: 1556-6021. DOI: 10.1109/TIFS.2023.3251842.

[75] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. "Adversarial Examples for Malware Detection". In: *Computer Security – ESORICS 2017*. Vol. 10493. Cham: Springer International Publishing, 2017, pp. 62–79. ISBN: 978-3-319-66398-2 978-3-319-66399-9. DOI: 10.1007/978-3-319-66399-9_4.

[76] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. "Adversarial Perturbations Against Deep Neural Networks for Malware Classification". In: *arXiv:1606.04435 [cs]* (June 2016). arXiv: 1606.04435 [cs].

[77] "Group Testing". In: *Wikipedia* (Aug. 2021).

[78] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks". In: *IEEE Access*. SPECIAL SECTION

ON ADVANCED SOFTWARE ANDDATA ENGINEERING FOR SECURE SOCIETIES 7 (2019), pp. 47230–47244. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2909068.

[79] Xinran Gu, Kaixuan Huang, Jingzhao Zhang, and Longbo Huang. "Fast Federated Learning in the Presence of Arbitrary Device Unavailability". In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021. URL: https://openreview.net/forum?id=1_gaHBaRYt.

[80] Rachid Guerraoui, Arsany Guirguis, Jérémy Plassmann, Anton Ragot, and Sébastien Rouault. "Garfield: System support for byzantine machine learning (regular paper)". In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2021, pp. 39–51.

[81] Nirupam Gupta, Shuo Liu, and Nitin H Vaidya. "Byzantine fault-tolerant distributed machine learning using stochastic gradient descent (sgd) and norm-based comparative gradient elimination (cge)". In: *arXiv preprint arXiv:2008.04699* (2020).

[82] Mark Handley, Vern Paxson, and Christian Kreibich. "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics". In: *10th USENIX Security Symposium (USENIX Security 01)*. Washington, D.C.: USENIX Association, Aug. 2001. URL: https://www.usenix.org/conference/10th-usenix-security-symposium/network-intrusion-detection-evasion-traffic-normalization.

[83] Richard Harang and Ethan M Rudd. "SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection". In: *Conference on Applied Machine Learning in Information Security*. 2021.

[84] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. "SPECTRE: Defending against Backdoor Attacks Using Robust Statistics". In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR, July 2021, pp. 4129–4139.

[85] Mingshu He, Xiaojuan Wang, Junhua Zhou, Yuanyuan Xi, Lei Jin, and Xinlei Wang. "Deep-Feature-Based Autoencoder Network for Few-Shot Malicious Traffic Detection". In: *Security and Communication Networks* 2021 (Mar. 2021), e6659022. ISSN: 1939-0114. DOI: 10.1155/2021/6659022.

[86] David Heckerman. "A Tutorial on Learning with Bayesian Networks". In: *Innovations in Bayesian Networks: Theory and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 33–82. ISBN: 978-3-540-85066-3. DOI: 10.1007/978-3-540-85066-3_3. URL: https://doi.org/10.1007/978-3-540-85066-3_3.

[87] Alvin Heng and Harold Soh. "Selective Amnesia: A Continual Learning Approach to Forgetting in Deep Generative Models". In: *NeurIPS*. Vol. 36. 2023, pp. 17170–17194.

[88] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. *Deep Learning Scaling Is Predictable, Empirically*. Dec. 2017. arXiv: 1712.00409 [cs, stat].

[89] Samson Ho, Achyut Reddy, Sridhar Venkatesan, Rauf Izmailov, Ritu Chadha, and Alina Oprea. "Data Sanitization Approach to Mitigate Clean-Label Attacks Against Malware Detection Systems". In: *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*. Nov. 2022, pp. 993–998. DOI: 10.1109/MILCOM55135.2022.10017768.

[90] Jordan Holland, Paul Schmitt, Prateek Mittal, and Nick Feamster. *Towards Reproducible Network Traffic Analysis*. Mar. 2022. arXiv: 2203.12410 [cs].

[91] Matthias Hollick, Cristina Nita-Rotaru, Panagiotis Papadimitratos, Adrian Perrig, and Stefan Schmid. "Toward a Taxonomy and Attacker Model for Secure Routing Protocols". In: *SIGCOMM Comput. Commun. Rev.* 47.1 (Jan. 2017), pp. 43–48. ISSN: 0146-4833. DOI: 10.1145/3041027.3041033. URL: https://doi.org/10.1145/3041027.3041033.

[92] John T. Holodnak, Olivia Brown, Jason Matterer, and Andrew Lemke. "Backdoor Poisoning of Encrypted Traffic Classifiers". In: *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. Nov. 2022, pp. 577–585. DOI: 10.1109/ICDMW58026.2022.00080.

[93] Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitraş, and Nicolas Papernot. "On the effectiveness of mitigating data poisoning attacks with gradient shaping". In: *arXiv preprint arXiv:2002.11497* (2020).

[94] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. "Measuring the effects of non-identical data distribution for federated visual classification". In: *arXiv preprint arXiv:1909.06335* (2019).

[95] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. *Sleeper Agents: Training Deceptive LLMs That Persist Through Safety Training*. Jan. 2024. arXiv: 2401.05566 [cs].

[96] IBM. *IBM Security QRadar XDR*. https://www.ibm.com/qradar. 2023.

[97] Sam Ingalls. *Top XDR Security Solutions for 2022*. https://www.esecurityplanet.com/products/xdr-security-solutions/. 2021.

[98] Luca Invernizzi, Sung-ju Lee, Stanislav Miskovic, Marco Mellia, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, and Giovanni Vigna. "Nazca: Detecting Malware Distribution in Large-Scale Networks". In: *NDSS*. 2014.

[99] M.A. Jabbar, Rajanikanth Aluvalu, and S. Sai Satyanarayana Reddy. "Intrusion Detection System Using Bayesian Network and Feature Subset Selection". In: *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*. 2017, pp. 1–5. DOI: 10.1109/ICCIC.2017.8524381.

[100] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. "AI/ML for Network Security: The Emperor Has No Clothes". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 1537–1551. ISBN: 9781450394505. DOI: 10.1145/3548606.3560609. URL: https://doi.org/10.1145/3548606.3560609.

[101] Matthew Jagielski, Nicholas Jones, Chung-Wei Lin, Cristina Nita-Rotaru, and Shinichi Shiraishi. "Threat Detection for Collaborative Adaptive Cruise Control in Connected Cars". In: *Proceedings of the 11th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '18. Stockholm, Sweden: Association for Computing Machinery, 2018, pp. 184–189. ISBN: 9781450357319. DOI: 10.1145/3212480.3212492. URL: https://doi.org/10.1145/3212480.3212492.

[102] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning". In: *2018 IEEE Symposium on Security and Privacy (SP)*. May 2018, pp. 19–35. DOI: 10.1109/SP.2018.00057.

[103] Matthew Jagielski, Giorgio Severi, Niklas Pousette Harger, and Alina Oprea. "Subpopulation Data Poisoning Attacks". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. CCS '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 3104–3122. ISBN: 978-1-4503-8454-4. DOI: 10.1145/3460120.3485368.

[104] Samuel Jero, Endadul Hoque, Dave Choffness, Alan Mislove, and Cristina Nita-Rotaru. "Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach". In: *NDSS*. 2018.

[105] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer

Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. "Advances and Open Problems in Federated Learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (June 2021), pp. 1–210. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000083.

[106] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. "Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels". In: *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*. AISec '15. New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 45–56. ISBN: 978-1-4503-3826-4. DOI: 10.1145/2808769.2808780.

[107] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. *Scaling Laws for Neural Language Models*. Jan. 2020. arXiv: 2001.08361 [cs, stat].

[108] Dhamanpreet Kaur, Matthew Sobiesk, Shubham Patil, Jin Liu, Puran Bhagat, Amar Gupta, and Natasha Markuzon. "Application of Bayesian networks to generate synthetic health data". In: *Journal of the American Medical Informatics Association* 28 (Dec. 2020). DOI: 10.1093/jamia/ocaa303.

[109] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. "TrojDRL: Evaluation of Backdoor Attacks on Deep Reinforcement Learning". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, July 2020, pp. 1–6. ISBN: 978-1-72811-085-1. DOI: 10.1109/DAC18072.2020.9218663.

[110] Dhilung Kirat and Giovanni Vigna. "MalGene: Automatic Extraction of Malware Analysis Evasion Signature". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. New York, NY, USA: ACM, 2015, pp. 769–780. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813642.

[111] Neville Kenneth Kitson, Anthony C. Constantinou, Zhigao Guo, Yang Liu, and Kiattikun Chobtham. "A Survey of Bayesian Network Structure Learning". In: *Artificial Intelligence Review* 56.8 (Aug. 2023), pp. 8721–8814. ISSN: 1573-7462. DOI: 10.1007/s10462-022-10351-w.

[112] Marius Kloft and Pavel Laskov. "Online Anomaly Detection under Adversarial Impact". In: *Proceedings of the 13thInternational Con-ference on Artificial Intelligence and Statistics*. 2010, p. 8.

[113] Paul Kocher, Jann Horn, Anders Fogh, and Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael

Schwarz, and Yuval Yarom. "Spectre Attacks: Exploiting Speculative Execution". In: *40th IEEE Symposium on Security and Privacy (S&P'19)*. 2019.

[114] Daphne Koller and Mehran Sahami. "Toward Optimal Feature Selection". In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. ICML'96. Bari, Italy: Morgan Kaufmann Publishers Inc., 1996, pp. 284–292. ISBN: 1558604197.

[115] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. "Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables". In: *2018 26th European Signal Processing Conference (EUSIPCO)*. Sept. 2018, pp. 533–537. DOI: 10.23919/EUSIPCO.2018.8553214.

[116] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. "TabDDPM: Modelling Tabular Data with Diffusion Models". In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR, July 2023, pp. 17564–17579.

[117] Marek Krčál, Ondřej Švec, Martin Bálek, and Otakar Jašek. "Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only". In: *International Conference on Learning Representations*. ICLR. 2018.

[118] Maximilian Lam, Gu-Yeon Wei, David Brooks, Vijay Janapa Reddi, and Michael Mitzenmacher. "Gradient Disaggregation: Breaking Privacy in Federated Learning by Reconstructing the User Participant Matrix". In: *Proceedings of the 38th International Conference on Machine Learning*. 2021.

[119] ChonLam Lao, Yanfang Le, and Kshiteej Mahajan. "ATP: In-network Aggregation for Multi-tenant Learning." In: *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 2021.

[120] Changki Lee and Gary Geunbae Lee. "Information gain and divergence-based feature selection for machine learning-based text categorization". In: *Information Processing & Management* 42.1 (2006). Formal Methods for Information Retrieval, pp. 155–165. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2004.08.006. URL: https://www.sciencedirect.com/science/article/pii/S0306457304000962.

[121] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. "Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia". In: *Semantic web* 6.2 (2015), pp. 167–195.

[122] Alexander Levine and Soheil Feizi. "Deep Partition Aggregation: Provable Defenses against General Poisoning Attacks". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YUGG2tFuPM.

[123] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.

[124] Pan Li, Qiang Liu, Wentao Zhao, Dongxu Wang, and Siqi Wang. "Chronic Poisoning against Machine Learning Based IDSs Using Edge Pattern Detection". In: *2018 IEEE International Conference on Communications (ICC)*. May 2018, pp. 1–7. DOI: 10.1109/ICC.2018.8422328.

[125] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. "Neural Attention Distillation: Erasing Backdoor Triggers from Deep Neural Networks". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[126] J. Lin. "Divergence measures based on the Shannon entropy". In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 145–151. DOI: 10.1109/18.61115.

[127] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable AI: A Review of Machine Learning Interpretability Methods". In: *Entropy* 23.1 (2021). ISSN: 1099-4300. DOI: 10.3390/e23010018. URL: https://www.mdpi.com/1099-4300/23/1/18.

[128] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. "Meltdown: Reading Kernel Memory from User Space". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[129] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. Dec. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[130] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks". In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 273–294. ISBN: 978-3-030-00470-5. DOI: 10.1007/978-3-030-00470-5_13.

[131] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. "ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1265–1282. ISBN: 9781450367479. DOI: 10.1145/3319535.3363216. URL: https://doi.org/10.1145/3319535.3363216.

[132] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. "Trojaning Attack on Neural Networks". In: *Proceedings 2018 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2018. ISBN: 978-1-891562-49-5. DOI: 10.14722/ndss.2018.23291.

[133] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774.

[134] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. "From Local Explanations to Global Understanding with Explainable AI for Trees". In: *Nature Machine Intelligence* 2.1 (Jan. 2020), pp. 56–67. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0138-9.

[135] *Machine Learning Static Evasion Competition*. https://www.elastic.co/blog/machine-learning-static-evasion-competition.

[136] *MalwareGuard: FireEye's Machine Learning Model to Detect and Prevent Malware*. https://www.fireeye.com/blog/products-and-services/2018/07/malwareguard-fireeye-machine-learning-model-to-detect-and-prevent-malware.html.

[137] Thomas Mandl, Ulrich Bayer, and Florian Nentwich. "ANUBIS ANalyzing Unknown BInarieS The Automatic Way". In: *Virus Bulletin Conference*. Vol. 1. 2009, p. 02.

[138] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models". In: *Proceedings 2017 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2017. ISBN: 978-1-891562-46-4. DOI: 10.14722/ndss.2017.23353.

[139] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2017, pp. 1273–1282.

[140] *MetaDefender Cloud | Homepage*. https://metadefender.opswat.com.

[141] El Mahdi El Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, Arsany Guirguis, Lê-Nguyên Hoang, and Sébastien Rouault. "Collaborative Learning in the Jungle (Decentralized, Byzantine, Heterogeneous, Asynchronous and Nonconvex Learning)". In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021. URL: https://openreview.net/forum?id=O8wI1avs4WF.

[142] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. "Distributed Momentum for Byzantine-resilient Stochastic Gradient Descent". In: *International*

Conference on Learning Representations. 2021. URL: https://openreview.net/forum?id=H8UHdhWG6A3.

[143] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. "The Hidden Vulnerability of Distributed Learning in Byzantium". In: *International Conference on Machine Learning*. July 2018. arXiv: 1802.07927.

[144] Microsoft. *Microsoft Defender for Endpoint | Microsoft Security*. 2021. URL: https://www.microsoft.com/en-us/security/business/threat-protection/endpoint-defender.

[145] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection". In: *Proceedings 2018 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2018. ISBN: 978-1-891562-49-5. DOI: 10.14722/ndss.2018.23204.

[146] Xiaoxing Mo, Yechao Zhang, Leo Yu Zhang, Wei Luo, Nan Sun, Shengshan Hu, Shang Gao, and Yang Xiang. "Robust Backdoor Detection for Deep Learning via Topological Evolution Dynamics". In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Mar. 2024, pp. 174–174. ISBN: 9798350331301. DOI: 10.1109/SP54263.2024.00174.

[147] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: https://christophm.github.io/interpretable-ml-book.

[148] Andrew Moore, Denis Zuev, and Michael Crogan. *Discriminators for Use in Flow-Based Classification*. Tech. rep. Queen Mary and Westfield College, Department of Computer Science, 2005.

[149] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. "Network Intrusion Detection". In: *IEEE Network* 8.3 (May 1994), pp. 26–41. ISSN: 1558-156X. DOI: 10.1109/65.283931.

[150] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. "Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization". In: *arXiv:1708.08689 [cs]* (Aug. 2017). arXiv: 1708.08689 [cs].

[151] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. "ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates". In: *Proceedings of the 22nd USENIX Conf. on Security*. USA: USENIX Association, 2013, pp. 589–604.

[152] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. "Exploiting Machine Learning to Subvert Your Spam Filter". In: *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. LEET'08. San Francisco, California: USENIX Association, 2008.

[153] Fredrik Nestaas, Edoardo Debenedetti, and Florian Tramèr. *Adversarial Search Engine Optimization for Large Language Models*. June 2024. DOI: 10.48550/arXiv.2406.18382. arXiv: 2406.18382 [cs].

[154] J. Newsome, B. Karp, and D. Song. "Polygraph: Automatically Generating Signatures for Polymorphic Worms". In: *2005 IEEE Symposium on Security and Privacy (S P'05)*. May 2005, pp. 226–241. DOI: 10.1109/SP.2005.15.

[155] James Newsome, Brad Karp, and Dawn Song. "Paragraph: Thwarting Signature Learning by Training Maliciously". In: *Recent Advances in Intrusion Detection*. Vol. 4219. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 81–105. ISBN: 978-3-540-39723-6 978-3-540-39725-0. DOI: 10.1007/11856214_5.

[156] Rui Ning, Chunsheng Xin, and Hongyi Wu. "TrojanFlow: A Neural Backdoor Attack to Deep Learning-based Network Traffic Classifiers". In: *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. May 2022, pp. 1429–1438. DOI: 10.1109/INFOCOM48880.2022.9796878.

[157] Daniel Obenshain, Thomas Tantillo, Amy Babay, John Schultz, Andrew Newell, Md. Edadul Hoque, Yair Amir, and Cristina Nita-Rotaru. "Practical Intrusion-Tolerant Networks". In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 2016, pp. 45–56. DOI: 10.1109/ICDCS.2016.99.

[158] Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. "Realistic Evaluation of Deep Semi-Supervised Learning Algorithms". In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[159] Talha Ongun, Simona Boboila, Alina Oprea, Tina Eliassi-Rad, Jason Hiser, and Jack W. Davidson. "CELEST: Federated Learning for Globally Coordinated Threat Detection". In: *CoRR* abs/2205.11459 (2022). DOI: 10.48550/arXiv.2205.11459. arXiv: 2205.11459. URL: https://doi.org/10.48550/arXiv.2205.11459.

[160] Talha Ongun, Timothy Sakharaov, Simona Boboila, Alina Oprea, and Tina Eliassi-Rad. *On Designing Machine Learning Models for Malicious Network Traffic Classification*. July 2019. arXiv: 1907.04846 [cs, stat].

[161] Talha Ongun, Oliver Spohngellert, Benjamin Miller, Simona Boboila, Alina Oprea, Tina Eliassi-Rad, Jason Hiser, Alastair Nottingham, Jack Davidson, and Malathi Veeraraghavan. "PORTFILER: Port-Level Network Profiling for Self-Propagating Malware Detection". In: *2021 IEEE Conference on Communications and Network Security (CNS)*. Oct. 2021, pp. 182–190. DOI: 10.1109/CNS53000.2021.9705045.

[162] OpenAI. *GPT-4 Technical Report*. Mar. 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs].

[163] Alina Oprea, Zhou Li, Robin Norris, and Kevin Bowers. "MADE: Security Analytics for Enterprise Threat Detection". In: *Proceedings of Annual Computer Security Applications Conference*. ACSAC. 2018. DOI: 10.1145/3274694.3274710.

[164] Alina Oprea and Apostol Vassilev. *Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations (Draft)*. Tech. rep. NIST AI 100-2e2023 ipd. National Institute of Standards and Technology, Mar. 2023.

[165] Tribhuvanesh Orekondy, Seong Joon Oh, Yang Zhang, Bernt Schiele, and Mario Fritz. "Gradient-leaks: Understanding and controlling deanonymization in federated learning". In: *arXiv preprint arXiv:1805.05838* (2018).

[166] Pavlos Papadopoulos, Oliver Thornewill von Essen, Nikolaos Pitropakis, Christos Chrysoulas, Alexios Mylonas, and William J. Buchanan. "Launching Adversarial Attacks against Network Intrusion Detection Systems for IoT". In: *Journal of Cybersecurity and Privacy* 1.2 (June 2021), pp. 252–273. ISSN: 2624-800X. DOI: 10.3390/jcp1020014.

[167] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. "Sageflow: Robust Federated Learning against Both Stragglers and Adversaries". In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021. URL: https://openreview.net/forum?id=rA9HFxFT7th.

[168] Dario Pasquini, Martin Strohmeier, and Carmela Troncoso. *Neural Exec: Learning (and Learning from) Execution Triggers for Prompt Injection Attacks*. May 2024. DOI: 10.48550/arXiv.2403.03792. arXiv: 2403.03792 [cs].

[169] Vern Paxson. "Bro: a System for Detecting Network Intruders in Real-Time". In: *Computer Networks* 31.23-24 (1999), pp. 2435–2463. URL: http://www.icir.org/vern/papers/bro-CN99.pdf.

[170] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[171] R. Perdisci, D. Dagon, Wenke Lee, P. Fogla, and M. Sharif. "Misleading Worm Signature Generators Using Deliberate Noise Injection". In: *2006 IEEE Symposium on Security and Privacy (S&P'06)*. Berkeley/Oakland, CA: IEEE, 2006, 15 pp.–31. ISBN: 978-0-7695-2574-7. DOI: 10.1109/SP.2006.26.

[172] Robert Philipp, Andreas Mladenow, Christine Strauss, and Alexander Völz. "Machine Learning as a Service: Challenges in Research and Applications". In: *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services*. iiWAS '20. Chiang Mai, Thailand: Association for Computing Machinery, 2021, pp. 396–406. ISBN: 9781450389228. DOI: 10.1145/3428757.3429152. URL: https://doi.org/10.1145/3428757.3429152.

[173] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. "Intriguing Properties of Adversarial ML Attacks in the Problem Space". In: *2020 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May

2020, pp. 1332–1349. ISBN: 978-1-72813-497-0. DOI: 10.1109/SP40000.2020.00073.

[174] Xiangyu Qi, Tinghao Xie, Jiachen T. Wang, Tong Wu, Saeed Mahloujifar, and Prateek Mittal. "Towards A Proactive {ML} Approach for Detecting Backdoor Poison Samples". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 1685–1702. ISBN: 978-1-939133-37-3.

[175] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. "Malware Detection by Eating a Whole EXE". In: *AAAI Workshop on Artificial Intelligence for Cyber Security*. AICS. 2018. arXiv: 1710.09435.

[176] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. "Segugio: Efficient Behavior-Based Tracking of Malware-Control Domains in Large ISP Networks". In: *2015 45th Annual IEEE/IFIP Int'l. Conf. on Dependable Systems and Networks*. IEEE, 2015, pp. 403–414.

[177] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125* (2022).

[178] Ethan Rathbun, Christopher Amato, and Alina Oprea. *SleeperNets: Universal Backdoor Poisoning Attacks Against Reinforcement Learning Agents*. May 2024. arXiv: 2405.20539 [cs].

[179] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, II–1278–II–1286.

[180] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. San Francisco, California, USA: ACM Press, 2016, pp. 1135–1144. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939778.

[181] Mauro Ribeiro, Katarina Grolinger, and Miriam A.M. Capretz. "MLaaS: Machine Learning as a Service". In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 2015, pp. 896–902. DOI: 10.1109/ICMLA.2015.152.

[182] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. "Automatic Analysis of Malware Behavior Using Machine Learning". In: *Journal of Computer Security* 19.4 (2011), pp. 639–668.

[183] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletarì, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N. Galtier, Bennett A. Landman, Klaus Maier-Hein, Sébastien Ourselin, Micah Sheller, Ronald M. Summers, Andrew Trask, Daguang Xu, Maximilian Baust, and M. Jorge Cardoso. "The Future of

Digital Health with Federated Learning". In: *npj Digital Medicine* 3.1 (Dec. 2020), p. 119. ISSN: 2398-6352. DOI: 10.1038/s41746-020-00323-1.

[184] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. "ANTIDOTE: Understanding and Defending against Poisoning of Anomaly Detectors". In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference - IMC '09*. Chicago, Illinois, USA: ACM Press, 2009, p. 1. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644895.

[185] *Russia compromised core router in US energy attacks*. https://www.computerweekly.com/news/2524 compromised-core-router-in-US-energy-attacks. 2018.

[186] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G. Bringas. "Opcode Sequences as Representation of Executables for Data-Mining-Based Unknown Malware Detection". In: *Information Sciences* 231 (May 2013), pp. 64–82. ISSN: 00200255. DOI: 10.1016/j.ins.2011.08.020.

[187] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. "Scaling Distributed Machine Learning with In-Network Aggregation". In: *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 2021, pp. 785–808.

[188] Joshua Saxe and Konstantin Berlin. "Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features". In: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. Oct. 2015, pp. 11–20. DOI: 10.1109/MALWARE.2015.7413680.

[189] Giorgio Severi, Simona Boboila, John Holodnak, Kendra Kratkiewicz, Rauf Izmailov, and Alina Oprea. *Model-Agnostic Clean-Label Backdoor Mitigation in Cybersecurity Environments*. July 2024. arXiv: 2407.08159 [cs].

[190] Giorgio Severi, Simona Boboila, Alina Oprea, John Holodnak, Kendra Kratkiewicz, and Jason Matterer. "Poisoning Network Flow Classifiers". In: *Proceedings of the 39th Annual Computer Security Applications Conference*. ACSAC '23. New York, NY, USA: Association for Computing Machinery, Dec. 2023, pp. 337–351. ISBN: 9798400708862. DOI: 10.1145/3627106.3627123.

[191] Giorgio Severi, Matthew Jagielski, Gökberk Yar, Yuxuan Wang, Alina Oprea, and Cristina Nita-Rotaru. "Network-Level Adversaries in Federated Learning". In: *2022 IEEE Conference on Communications and Network Security (CNS)*. Oct. 2022, pp. 19–27. DOI: 10.1109/CNS56114.2022.9947237.

[192] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. "Malrec: Compact Full-Trace Malware Recording for Retrospective Deep Analysis". In: *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Vol. 10885. Cham:

Springer International Publishing, 2018, pp. 3–23. ISBN: 978-3-319-93410-5 978-3-319-93411-2. DOI: 10.1007/978-3-319-93411-2_1.

[193] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. "Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 1487–1504. ISBN: 978-1-939133-24-3.

[194] Hovav Shacham. "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the X86)". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. ACM, 2007, pp. 552–561.

[195] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks". In: *Advances in Neural Information Processing Systems*. Apr. 2018.

[196] Andrii Shalaginov, Sergii Banin, Ali Dehghantanha, and Katrin Franke. "Machine Learning Aided Static Malware Analysis: A Survey and Tutorial". In: *Cyber Threat Intelligence*. Vol. 70. Cham: Springer International Publishing, 2018, pp. 7–45. ISBN: 978-3-319-73950-2 978-3-319-73951-9. DOI: 10.1007/978-3-319-73951-9_2.

[197] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y. Zhao. "Poison Forensics: Traceback of Data Poisoning Attacks in Neural Networks". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 3575–3592. ISBN: 978-1-939133-31-1.

[198] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:" in: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. ISBN: 978-989-758-282-0. DOI: 10.5220/0006639801080116.

[199] Ryan Sheatsley, Blaine Hoak, Eric Pauley, Yohan Beugin, Michael J. Weisman, and Patrick McDaniel. "On the Robustness of Domain Constraints". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 495–515. ISBN: 9781450384544. DOI: 10.1145/3460120.3484570. URL: https://doi.org/10.1145/3460120.3484570.

[200] Virat Shejwalkar and A. Houmansadr. "Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning". In: *NDSS*. 2021.

[201] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *International conference on machine learning*. PMLR. 2017, pp. 3145–3153.

[202] Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. "Adversarial Machine Learning-Industry Perspectives". In: *2020 IEEE Security and Privacy Workshops (SPW)*. May 2020, pp. 69–75. DOI: 10.1109/SPW50608.2020.00028.

[203] *Skylight Cyber | Cylance, I Kill You!* https://skylightcyber.com/2019/07/18/cylance-i-kill-you/.

[204] Charles Smutz and Angelos Stavrou. "Malicious PDF Detection Using Metadata and Structural Features". In: *Proceedings of the 28th Annual Computer Security Applications Conference on - ACSAC '12*. Orlando, Florida: ACM Press, 2012, p. 239. ISBN: 978-1-4503-1312-4. DOI: 10.1145/2420950.2420987.

[205] Nedim Srndic and Pavel Laskov. "Practical Evasion of a Learning-Based Classifier: A Case Study". In: *2014 IEEE Symposium on Security and Privacy*. San Jose, CA: IEEE, May 2014, pp. 197–211. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.20.

[206] Michael Steinbach, Levent Ertöz, and Vipin Kumar. "The Challenges of Clustering High Dimensional Data". In: *New Directions in Statistical Physics: Econophysics, Bioinformatics, and Pattern Recognition*. Ed. by Luc T. Wille. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 273–309. ISBN: 978-3-662-08968-2. DOI: 10.1007/978-3-662-08968-2_16. URL: https://doi.org/10.1007/978-3-662-08968-2_16.

[207] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. "Certified defenses for data poisoning attacks". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3520–3532. ISBN: 9781510860964.

[208] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. "Learning to Summarize with Human Feedback". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 3008–3021.

[209] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. "Exploring Adversarial Examples in Malware Detection". In: *2019 IEEE Security and Privacy Workshops (SPW)*. San Francisco, CA, USA: IEEE, May 2019, pp. 8–14. ISBN: 978-1-72813-508-3. DOI: 10.1109/SPW.2019.00015.

[210] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume Iii, and Tudor Dumitras. "When Does Machine Learning {FAIL}? Generalized Transferability for Evasion and Poisoning Attacks". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 1299–1316. ISBN: 978-1-939133-04-5.

[211] Jingwei Sun, Ang Li, Louis DiValentin, Amin Hassanzadeh, Yiran Chen, and Hai Li. "FL-WBC: Enhancing Robustness against Model Poisoning Attacks in Federated Learning from a Client Perspective". In: *Thirty-Fifth Conference on Neural*

Information Processing Systems. 2021. URL: https://openreview.net/forum?id=96uH8HeGb9G.

[212] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. "Can you really backdoor federated learning?" In: *arXiv preprint arXiv:1911.07963* (2019).

[213] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR, July 2017, pp. 3319–3328.

[214] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[215] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing Properties of Neural Networks". In: *International Conference on Learning Representations*. 2014.

[216] Kimberly Tam, Salahuddin J. Khan, Aristide Fattori, and Lorenzo Cavallaro. "CopperDroid: Automatic Reconstruction of Android Malware Behaviors". In: Internet Society, 2015. ISBN: 978-1-891562-38-9. DOI: 10.14722/ndss.2015.23145.

[217] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. "Guilt by Association: Large Scale Malware Detection by Mining File-Relation Graphs". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 1524–1533. ISBN: 9781450329569. DOI: 10.1145/2623330.2623342. URL: https://doi.org/10.1145/2623330.2623342.

[218] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. "Data Poisoning Attacks Against Federated Learning Systems". en. In: *arXiv:2007.08432 [cs, stat]* (Aug. 2020). arXiv: 2007.08432 [cs, stat].

[219] Florian Tramèr and Javier Rando Ramirez. "Universal Jailbreak Backdoors from Poisoned Human Feedback". In: *The Twelfth International Conference on Learning Representations (ICLR 2024)*. OpenReview, 2024. DOI: 10.3929/ethz-b-000660463.

[220] Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. "Truth Serum: Poisoning Machine Learning Models to Reveal Their Secrets". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 2779–2792. ISBN: 978-1-4503-9450-5. DOI: 10.1145/3548606.3560554.

[221] Brandon Tran, Jerry Li, and Aleksander Mądry. "Spectral Signatures in Backdoor Attacks". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., Dec. 2018, pp. 8011–8021.

[222] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. *Label-Consistent Backdoor Attacks*. Dec. 2019. DOI: 10.48550/arXiv.1912.02771. arXiv: 1912.02771 [cs, stat].

[223] Alexander Turner, Dimitris Tsipras, and Aleksander Mądry. "Clean-Label Backdoor Attacks". In: *Manuscript submitted for publication* (2018), p. 21.

[224] María Vargas Muñoz, Rafael Martínez-Peláez, Pablo Velarde Alvarado, Efraín Moreno-Garcia, Deni Torres-Roman, and José Ceballos-Mejia. "Classification of network anomalies in flow level network traffic using Bayesian networks". In: *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. IEEE, Feb. 2018, pp. 238–243. DOI: 10.1109/CONIELECOMP.2018.8327205.

[225] Shobha Venkataraman, Avrim Blum, and Dawn Song. "Limits of Learning-based Signature Generation with Adversaries". In: *16th Annual Network & Distributed System Security Symposium Proceedings*. 2008, p. 16.

[226] Sridhar Venkatesan, Harshvardhan Sikka, Rauf Izmailov, Ritu Chadha, Alina Oprea, and Michael J. de Lucia. "Poisoning Attacks and Data Sanitization Mitigations for Machine Learning Models in Network Intrusion Detection Systems". In: *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*. Nov. 2021, pp. 874–879. DOI: 10.1109/MILCOM52596.2021.9652916.

[227] Sridhar Venkatesan, Harshvardhan Sikka, Rauf Izmailov, Ritu Chadha, Alina Oprea, and Michael J. de Lucia. "Poisoning Attacks and Data Sanitization Mitigations for Machine Learning Models in Network Intrusion Detection Systems". In: *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*. 2021, pp. 874–879. DOI: 10.1109/MILCOM52596.2021.9652916.

[228] *VirSCAN.Org - Free Multi-Engine Online Virus Scanner v1.02, Supports 47 AntiVirus Engines!* https://www.virscan.org/.

[229] *VirusTotal - Home*. https://www.virustotal.com/gui/home/upload.

[230] Aaron Walters, David Zage, and Cristina Nita Rotaru. "A Framework for Mitigating Attacks Against Measurement-Based Adaptation Mechanisms in Unstructured Multicast Overlay Networks". In: *IEEE/ACM Transactions on Networking* 16.6 (2008), pp. 1434–1446. DOI: 10.1109/TNET.2007.912394.

[231] B. Wang, X. Cao, J. Jia, and N. Z. Gong. "On certifying robustness against backdoor attacks via randomized smoothing". In: *CVPR 2020 Workshop on Adversarial Machine Learning in Computer Vision*. 2020.

[232] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". In: *2019 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2019, pp. 707–723. ISBN: 978-1-5386-6660-9. DOI: 10.1109/SP.2019.00031.

[233] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. "Attack of the Tails: Yes, You Really Can Backdoor Federated Learning". In: *Advances in Neural Information Processing Systems*. 2020. arXiv: 2007.05084.

[234] Wenxiao Wang, Alexander Levine, and Soheil Feizi. "Improved Certified Defenses against Data Poisoning with (Deterministic) Finite Aggregation". In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 22769–22783. URL: https://proceedings.mlr.press/v162/wang22m.html.

[235] Ben Weintraub, Cristina Nita-Rotaru, and Stefanie Roos. "Structural Attacks on Local Routing in Payment Channel Networks". In: *Workshop on Security and Privacy on the Blockchain*. 2021.

[236] Di Wu, Binxing Fang, Junnan Wang, Qixu Liu, and Xiang Cui. "Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. Shanghai, China: IEEE, 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761337.

[237] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *arXiv:1708.07747 [cs, stat]*. 2017. URL: http://arxiv.org/abs/1708.07747.

[238] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. "Is Feature Selection Secure against Training Data Poisoning?" In: *International Conference on Machine Learning*. 2015, p. 10.

[239] Jian Xu, Shao-Lun Huang, Linqi Song, and Tian Lan. "SignGuard: Byzantine-robust Federated Learning through Collaborative Malicious Gradient Filtering". In: *CoRR* abs/2109.05872 (2021).

[240] Jing Xu and Christian R. Shelton. "Intrusion Detection Using Continuous Time Bayesian Networks". In: *J. Artif. Int. Res.* 39.1 (Sept. 2010), pp. 745–774. ISSN: 1076-9757.

[241] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. "Modeling Tabular Data Using Conditional GAN". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[242] Weilin Xu, Yanjun Qi, and David Evans. "Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers". In: *Proceedings 2016 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2016. ISBN: 978-1-891562-41-9. DOI: 10.14722/ndss.2016.23115.

[243] Weiyu Xu and A Kevin Tang. "A generalized coupon collector problem". In: *Journal of Applied Probability* 48.4 (2011), pp. 1081–1094.

[244] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. "The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks". In: *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc '05. Urbana-Champaign, IL, USA: Association for Computing Machinery, 2005, pp. 46–57. ISBN: 1595930043. DOI: 10.1145/1062689.1062697. URL: https://doi.org/10.1145/1062689.1062697.

[245] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. "Detecting AI Trojans Using Meta Neural Analysis". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021.

[246] Kun Yang, Samory Kpotufe, and Nick Feamster. *Feature Extraction for Novelty Detection in Network Traffic*. June 2021. DOI: 10.48550/arXiv.2006.16993. arXiv: 2006.16993 [cs].

[247] Limin Yang, Zhi Chen, Jacopo Cortellazzi, Feargus Pendlebury, Kevin Tu, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. "Jigsaw Puzzle: Selective Backdoor Attack to Subvert Malware Classifiers". In: *IEEE Symposium on Security & Privacy*. Mar. 2023.

[248] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. "Applied federated learning: Improving google keyboard query suggestions". In: *arXiv preprint arXiv:1812.02903* (2018).

[249] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. "Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps". In: *Proceedings of the 33rd Annual Computer Security Applications Conference on - ACSAC 2017*. Orlando, FL, USA: ACM Press, 2017, pp. 288–302. ISBN: 978-1-4503-5345-8. DOI: 10.1145/3134600.3134642.

[250] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. "Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates". en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2018, pp. 5650–5659. URL: https://proceedings.mlr.press/v80/yin18a.html.

[251] Jim Young, Patrick Graham, and Richard Penny. "Using Bayesian Networks to Create Synthetic Data". In: *Journal of Official Statistics* 25 (Dec. 2009), pp. 549–567.

[252] Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y. Chen. "CTAB-GAN: Effective Table Data Synthesizing". In: *Proceedings of The 13th Asian Conference on Machine Learning*. PMLR, Nov. 2021, pp. 97–112.

[253] Ligeng Zhu and Song Han. "Deep leakage from gradients". In: *Federated learning*. Springer, 2020, pp. 17–31.

[254] Rui Zhu, Di Tang, Siyuan Tang, XiaoFeng Wang, and Haixu Tang. "Selective Amnesia: On Efficient, High-Fidelity and Blind Suppression of Backdoor Effects in

Trojaned Machine Learning Models". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 682–700. ISBN: 978-1-66549-336-9. DOI: 10.1109/SP46215.2023.00070.

[255] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. *PoisonedRAG: Knowledge Poisoning Attacks to Retrieval-Augmented Generation of Large Language Models*. Feb. 2024. DOI: 10.48550/arXiv.2402.07867. arXiv: 2402.07867 [cs].