Informatics

Herausschleusung von Daten mittels Neuronalen Netzwerken

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

UE 066 645 Data Science

eingereicht von

Andrea Siposova, BSc.
Matrikelnummer 01205149

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Mitwirkung: Univ.Lektor Mag.rer.soc.oec. Dipl.-Ing. Rudolf Mayer

Wien, 5. Juli 2023

Andrea Siposova	Andreas Rauber

Informatics

Data Exfiltration Attacks and Defenses in Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

UE 066 645 Data Science

by

Andrea Siposova, BSc.
Registration Number 01205149

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Assistance: Univ.Lektor Mag.rer.soc.oec. Dipl.-Ing. Rudolf Mayer

Vienna, 5th July, 2023

Andrea Siposova	Andreas Rauber

Erklärung zur Verfassung der Arbeit

Andrea Siposova, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 5. Juli 2023

Andrea Siposova

Acknowledgements

Words cannot fully capture the gratitude I have for my wonderful parents, their boundless love and understanding. One of the most valuable lessons you've imparted to me is the importance of education. You've taught me to cherish the pursuit of knowledge. The completion of this degree is a product of your support. Further, I would like to thank my sister for her encouragement to follow the path of learning, no matter how challenging it may be.

I am eternally grateful to my Or. Thanks to your unwavering support and constant encouragement, I was spared the anguish of despair. Thank you for always having my back, and being the torch of light in my life, even when times get pretty tense, Or.

I would like to thank my friends, who have so tirelessly been asking me "So how's your thesis going?" – your support helped me stay on track and not underfit to my potential. Thank you for always staying fully connected and keeping my life balanced and fun. It provided me with a great defense mechanism against stress and worry. I appreciate you standing by me during this epoch of my life.

Further, I express my gratitude to the co-supervisor of this work, Rudolf Mayer, for believing in me. I also thank you for your guidance, not only during the development and writing of this thesis but throughout my academic journey. It taught me to uncover hidden knowledge. Your insightful suggestions and your feedback that led me to fine-tune my work played an important role in the successful completion of this thesis. Your expertise helped me achieve an optimal learning rate during my studies and augmented my experience for the better. You really put the super in supervisor :)

I am thankful to my thesis supervisor, Professor Andreas Rauber, for his guidance, his valuable feedback, and for pointing me in the right direction, which provided me with an enriched perspective throughout this journey.

.א

Kurzfassung

Datenqualität wirkt sich unmittelbar auf die Wirksamkeit von Modellen des maschinellen Lernens aus, und ihre Beschaffung ist oft mit erheblichen Investitionen verbunden. Fragen der Vertraulichkeit von Daten, insbesondere wenn es sich um sensible Informationen handelt, werden daher immer wichtiger. Algorithmen von Drittanbietern, die zur Erstellung von maschinellen Lernmodellen verwendet werden, können ein Risiko für die Vertraulichkeit solcher wertvollen Daten darstellen, da ihre Kapazität genutzt werden kann, um die Trainingsdaten zu verbergen, die anschließend von einem Angreifer exfiltriert werden können. Wir schlagen eine Taxonomie von Angriffen zur Datenexfiltration vor. Wir simulieren solche Angriffe in zwei Szenarien, die davon abhängen, welchen Zugriff ein Angreifer auf das endgültige, trainierte Modell hat - ein White-Box- oder ein Black-Box-Szenario. Um die Angriffe durchzuführen, passen wir einen zuvor vorgestellten Ansatz an, der mit künstlichen neuronalen Netzwerken arbeitet, die auf tabellarischen Daten trainiert wurden. Wir messen den Nutzen der Angriffe, indem wir die Ähnlichkeit der exfiltrierten Daten mit den originalen Trainingsdaten berechnen. Wir bestimmen die Angriffseinstellungen, die zu einer 100-prozentigen Ähnlichkeit der exfiltrierten Daten führen. Außerdem messen wir die Auswirkungen dieser Angriffe auf die Vorhersagekraft der zugrunde liegenden Modelle bei der ursprünglichen Klassifizierungsaufgabe. Anschließend implementieren wir entsprechende Abwehrmethoden. Wir zeigen, dass die gewählten Verteidigungsstrategien den aus den Angriffen resultierenden Schaden erfolgreich abmildern, ohne die Modellperformance zu beeinträchtigen, selbst wenn der Angreifer versucht, die Robustheit der Angriffe (z.B. durch Verwendung von error correction) zu erhöhen. Darüber hinaus zeigen wir, dass die Anwendung der Verteidigungsstrategien die Leistung der Basismodelle (d.h. der nicht angegriffenen Modelle) nicht wesentlich beeinträchtigt, was auf ihre Universalität hindeutet.

Abstract

Quality of data directly impacts the effectiveness of machine learning models and its acquisition often involves substantial investments. Confidentiality issues concerning data, especially when sensitive information is involved, therefore become increasingly pertinent. Third-party algorithms employed for building machine learning models can pose a risk to the confidentiality of such valuable data, as their capacity can be exploited to hide the training data, which can be subsequently exfiltrated by an adversary. We introduce a taxonomy of data exfiltration attacks. Further, we simulate such attacks in two scenarios depending on the access an adversary has to the final, trained model - a white-box or a black-box scenario. To perform the attacks, we adapt a previously introduced approach [63] to work with artificial neural networks trained on tabular data. We measure the utility of the attacks by calculating the similarity of exfiltrated data to the original data and determine the attack settings leading to a 100% similarity of exfiltrated data. Additionally, we measure the impact these attacks have on the prediction effectiveness of the models on the original classification task. Subsequently, we implement corresponding defense methods. We show that the chosen defense strategies are successful at mitigating the impact of the attacks, without compromising the model performance, even when the adversary attempts to increase the robustness of the attacks (e.g. by employing error correction techniques). Moreover, we show that the application of the defenses does not compromise the performance of the base (i.e. not attacked) models, which hints at their universality.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The range and scope of machine learning applications have broadened significantly over the last decades. Propelled by its transformative potential across industries, it has emphasized the critical role of the data upon which machine learning models are built. Clive Humby, a mathematician and data scientist, drew an interesting parallel by claiming that "data is the new oil", thereby highlighting the potential value of data. Moreover, this quote stresses the importance of data quality, and how refinement and suitable transformation of raw data raises its value. Its quality, among other factors, feeds directly into the robustness of the predictive power of machine learning models. Further factors contributing to an increased value of data are the costs associated with its acquisition. In many cases, there are significant investment and time resources involved in the collection and labeling of high-quality data. Reasons for the protection of data ownership are therefore apparent. If the data contains sensitive information regarding individuals, the confidentiality aspect takes on yet another dimension.

However, data confidentiality concerns may arise when third-party algorithms are applied to confidential data. Third-party machine learning algorithms might be employed for a variety of reasons. A data owner might seek the machine learning expertise of third parties if they do not possess such knowledge themselves. As a result, they outsource the model creation to a third-party provider. Another example of external code being applied to data is when a data owner allows data visiting - a setting when the data holder, for example, does not have a use-case for their data but allows an interested third party to use it without data sharing access. Libraries offering a wide range of functionalities are readily available for use in all steps of the machine learning process, making it more accessible. In another scenario, an adversary could therefore potentially contribute malicious code to a library in order to carry out the data-hiding step of data exfiltration. Should the adversary be able to gain access to a model trained using such

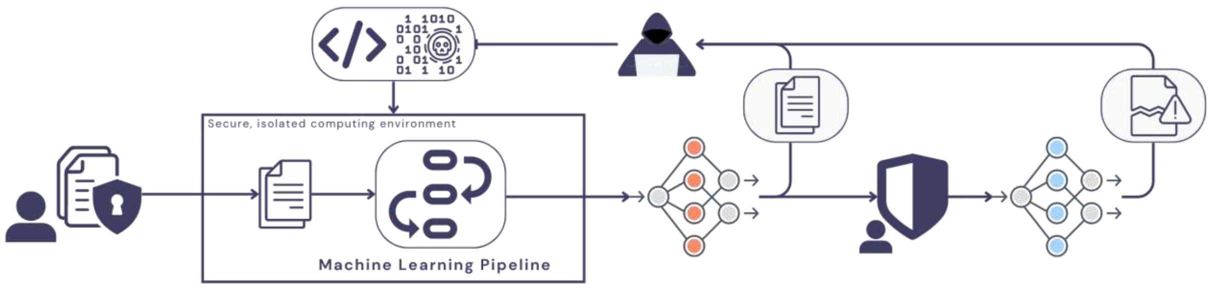

Secure, isolated computing environment
Machine Learning Pipeline

Figure 1.1: Overview of a data exfiltration pipeline: the adversary provides malicious code that has the ability to exfiltrate confidential data even if executed in an isolated computing environment. The right-most part of the figure depicts a defense being applied to the model, with the aim of removing the data exfiltration functionality, thereby preventing the attacker from reconstructing the hidden training

library, the extraction step can then be performed. A common element in the scenarios mentioned above is the interest of the data holder to keep their confidential data private and not share it directly with any third party. The third party, however, might have malicious intent to gain unauthorized access to the private training data, thereby acting as an adversary. This is why using or allowing third-party algorithms puts data holders at risk, as information about their private data could be revealed.

1.2 Problem Statement

The goal of a data exfiltration attack is to extract information about the training data from a trained machine learning model. The code provided by a third party contains hidden malicious code segments that cause the training data itself to be embedded into the model. Typically, data exfiltration attacks (DEA) utilizing machine learning models can be classified by the mode of embedding of information into a model. The adversary can either hide information in a model using techniques from the field of steganography or utilize the direct memorization of data that occurs in the training process of machine learning models. The goal of this thesis is to investigate data exfiltration attacks based on the steganographic approach, abusing machine learning models, as well as defense strategies to defend machine learning models from attacks of this sort.

In the attacks from the data hiding category, an adversary has influence over the training process, during which the algorithm simultaneously trains a machine learning model whilst also encoding (parts of the) training data into the model itself. This allows the attacker to carry out the exfiltration even when the malicious code is executed within a secure, isolated computing environment. This poses a significant risk for the data owner. This process of data exfiltration is shown in Figure 1.1. The right part of the figure shows an active defense being applied to the model, with the goal of mitigating the potential damage caused by data exfiltration. In order to develop a suitable defense, the nature of the attacks and the adversarial techniques must be investigated. In which way exactly the adversary influences the training process depends on the type of access is the adversary assumed to gain to the model once it is trained. We distinguish the following cases:

1. **White-box access** (access to the model parameters): This type of attack abuses the representation capacity of the model parameters. The adversary decodes the information (secret) hidden in the learned parameters in order to reconstruct (parts of) the training data.

2. **Black-box access** (input-output access only, e.g. via a prediction API): In the black-box scenario, the learning capacity of the model is abused. Instead of encoding the information (secret) into the parameters of the model, the algorithm makes the model remember specific responses for crafted inputs; it, therefore, generates (artificial) data points, and each of these data points is labeled with (a part of) the secret. Once the adversary wants to gain access to the secret (i.e. the confidential training data), they produce the same data points and enter them into the prediction API. The label containing the secret is returned.

Therefore, the requirements for the development of defense mechanisms differ given the data-hiding technique. In an active defense approach, the model itself needs to be modified, in order to (partially) remove this information from this attacked model. If the attacker is not assumed to have access to the model parameters, the active defenses will comprise other techniques, for example adapting those from other adversarial machine learning methods, like detecting adversarial examples or backdoors, or from model watermarking, in order to purposefully remove those parts of the model that are not responsible for learning the original task. Another mitigation strategy could be a passive defense that introduces restricting the number of queries an individual is allowed to submit to the prediction API, which would limit the adversary in how much information they can access by querying the API. The main focus of this thesis are attacks by utilizing steganography techniques. Due to the resulting attacked model being modified in comparison to a benign model, two trade-offs need to be considered in this scenario:

- **Attack effectiveness vs. Model utility**: Since the resulting model is modified in comparison to a benign model, the model effectiveness (measured e.g. via accuracy, precision, recall, etc.) on the original task is expected to decrease. The more information is encoded into a model, the lower the utility of this model becomes. On the one hand, it would be possible to fully encode the training dataset into a model and perform a highly successful exfiltration attack, on the other hand, the model would lose its utility for its intended task. Subsequently, third-party algorithms yielding low-quality models could potentially be suspected of being malicious.

- **Defense effectiveness vs. Model utility**: As the defense strategies modify the model, also here, quantifying the change in model utility is necessary. The goal is to minimize the attacker's chances of data exfiltration while preserving an acceptable utility of the model. Otherwise, the defender could potentially modify all model parameters, thus removing the chance of data exfiltration altogether, but at the same time, removing any utility of the model for its intended task. The defender thus faces the challenge of determining the acceptable threshold for the model's utility drop - the cost of the defense. Furthermore, it is important to consider the effectiveness of the attack in relation to the size of the model. A model with a capacity much larger than necessary for a certain task can hold more information about the training data. Therefore models disproportionately large in relation to the underlying task could be used to exfiltrate large amounts of data with high success, but potentially be more suspicious for attack detection. However, determining the proportionate model size is not a straightforward task, and can thus not serve as an indicator of an attack.

The effectiveness of the attacks, as well as the defenses, is examined and the above-mentioned trade-offs are quantified. Experiments with different settings for the attacks and defenses are conducted. An evaluation of the resulting performance is performed. In order to precisely define the scope of the thesis, the research questions presented in the following section have been formulated.

1.3 Research Questions

1 **In what way can successful data exfiltration attacks using machine learning models be designed?** This question aims to explore the possible techniques on how to encode information into machine learning models and subsequently exfiltrate this hidden information. The quality and the amount of data, which can be exfiltrated, need to be considered in relation to the size and the complexity of the model used for such an attack, as the amount of information that can be encoded in the model increases with expanding model capacity. Abusing a model disproportionately large to the given task would artificially maximize the effectiveness of an attack. The goal is therefore to design attacks using models of proportionate capacity to the intended task, as well as considering the above-described attack effectiveness vs. utility trade-off.

2 **To what extent are the attacks useful?** In order to determine the usefulness of the attacks, the attack effectiveness and the model utility are measured. The relationship between these measures is evaluated.

   a) **To what extent can data be exfiltrated from trained machine learning models by performing attacks utilizing data hiding techniques?** This question aims to investigate the success rate of data exfiltration attacks.

In order to quantify the effectiveness of such attacks, the *similarity* between the original training data, and the exfiltrated (reconstructed) data after performing an attack is measured. Additionally, the amount of recovered data can be measured.

b) **To what extent does the model effectiveness change after incorporating data exfiltration functionality?** In order to perform data exfiltration attacks, the original, benign algorithm is modified in order to encode information about the training data into the model itself. Thus, a decrease in the model utility is expected, as parts of the model are changed or replaced (in comparison to a benign training process) to encompass the additional information. Preserving the performance of the model on the original, intended task as much as possible poses a challenge. Otherwise, the algorithm could easily be modified to simply encode all of the training data into the model instead of training it, which would subsequently render the model useless. In order to assess the utility of the attack model on the original task, metrics such as accuracy, precision, recall, F1-Score, and ROC-AUC Score are employed. The performance of the attack model is compared to the benign model to measure the change in model utility.

3 **In what way can models used for information hiding be successfully defended from data exfiltration attacks?** The aim of this question is to find out how to design defense strategies against data exfiltration attacks. As the secret information gets embedded into the model, the defenses will consist of techniques that modify the trained model. Rendering the model useless would effectively eliminate the option of data exfiltration. The following research questions, therefore, quantify the important measures to be considered when aiming to design successful, optimal defense strategies, while preserving model utility.

4 **To what extent are the defense strategies useful?** In order to assess the usefulness of the defenses, the defense effectiveness, and the model utility are tracked. The trade-off between these measures is evaluated.

a) **To what extent do the chosen defenses decrease the success of the data exfiltration attacks?** This question seeks to examine the effectiveness of various defense strategies protecting against data exfiltration attacks. The measurement technique from the RQ2(a) is used here as well, in order to quantify the effectiveness of a defense mechanism. In RQ2(a), the similarity is measured using an undefended model. In contrast, this research question aims to assess the similarity using a defended model. As defense is put in place on top of an undefended model, the similarity of the exfiltrated data to the original data is expected to decrease.

b) **To what extent does the model utility change after the application of defense strategies against data exfiltration attacks?** Due to the model parameter modification by a defense strategy, a trade-off between the model utility and the defense effectiveness is expected. The change in the model utility portrays the cost of utilizing defense strategies. The proposed method to measure this change is to compare the performance of the model trained with a benign algorithm, a malicious algorithm, and a malicious algorithm with an applied defense. The metrics used in RQ2(b) (accuracy, precision, recall, etc.) measure the impact of the defenses on the model utility.

1.4 Methodology

In this section, the methodology of this thesis is described in detail. The methodology is tailored to address the research objectives and to ensure that the outlined research questions can be answered.

1.4.1 Literature research

In order to investigate the state-of-the-art research, a literature review is performed according to the guidelines developed by Kitchenham et al. [39]. A restricted version of their method has been used to detect relevant literature based on defined inclusion and exclusion criteria and to specify the information to be acquired from the inspected studies.

The goal is to research and gather information on the approaches within the field of security of machine learning models and data confidentiality, specifically regarding research on data exfiltration. In order to identify literature relevant to the security of machine learning models and particularly to data exfiltration, the following criteria have been defined:

1. Literature surveying adversarial machine learning and corresponding defense strategies.
2. Literature proposing methods for data hiding in machine learning models.
3. Literature proposing methods for removal of information hidden in models.

The literature review serves as a theoretical base for the implementations and experiment design. It also serves the purpose of identifying the requirements for the design of data exfiltration attacks and defenses.

1.4.2 Threat Modeling

Understanding of the operational structure of a potential security breach commonly involves modeling possible threats. Within this work, a threat model is constructed for each type of data exfiltration attack. This model elaborates the profile of an attacker, defender, and the computational environment and is based on a threat modeling structure for machine learning-based systems from [5, 10]. It describes the incentives and goals of the attacker and elaborates on their capabilities. This includes the knowledge of the attacker and what access they have to the model and the underlying data. Furthermore, other aspects that must be considered when modeling security threats are the resources and possible actions of the attacker. The threat model thus contains information on the adversary's objectives and goals, how much effort they are able to invest, and what their capabilities are. Threat models further provide a basis for modeling corresponding defense techniques. In this case, the side of a defender is included as well. The same factors must be considered for the defender - the defender's goals and capabilities. The threat models serve as a foundation for the design and implementation of both the attacks and defenses.

1.4.3 Design and implementation of data exfiltration attacks abusing machine learning models

The attacks are based on the foundational study published by Song et al. [63]. Their attack design guides the implementation within this work. However, adaptations of the attacks are carried out in order to perform the attack on models trained on tabular data.

1.4.4 Design and implementation of defense techniques in order to minimize chances of data exfiltration

The design of countermeasures is undertaken based on the type of attack that is carried out. Subsequently, these defense techniques are implemented in order to defend the models obtained in the previous step.

1.4.5 Evaluation and comparison of different settings

For each implemented attack, as well as each implemented combination of an attack-defense strategy, the effectiveness is measured using the metrics defined within the research questions. The results are collected in order to evaluate the performance of the attacks and defenses by mainly assessing the degree of data exfiltration on undefended and defended models.

1.4.6 Analysis and defining guidelines for selection of attack settings and optimal defense strategies

Analysis of the results is performed in order to find the most optimal settings for defense strategies. Additionally, the analysis considers the above-mentioned trade-offs between the attack/defense effectiveness and the utility of the models. A set of recommendations is developed based on the analysis results regarding the various settings and trade-offs.

We follow the principles of the CRISP-DM framework [70] to guide the experiment design and evaluation process.

1.5 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 1 introduces the topic and establishes the problem which we attempt to subsequently solve. It describes the methodological approach applied within this thesis. Chapter 2 provides the necessary background knowledge on the topic, giving an overview on the notation used throughout this work, and explaining the relevant underlying concepts and methods. Chapter 3 presents an overview of relevant research in the field of machine learning security and related work. Chapter 4 details the threat models for each attack scenario investigated. This chapter defines the roles of an adversary and a defender, along with their incentives, goals, knowledge, and actions. Chapter 5 describes the experiment design in detail, The experiment setup is modeled for each set of experiments performed. This provides a detailed description of the steps taken within each experiment to investigate the change in the effectiveness and utility from a benign model to an attacked one to a defended one. Chapter 6 presents the results of experiments conducted and interprets them, analyzing and discussing the factors that led to these specific outcomes. Chapter 7 summarizes the outcomes of this work, and elaborates on the directions of future research in the field of data exfiltration exploiting machine learning models.

CHAPTER 2

Background

This chapter provides background knowledge relevant to the research and experiments conducted within this work. An introduction to the field of machine learning is followed by descriptions of supervised learning, and more specifically classification. Further, evaluation techniques along with relevant metrics are presented. Then, concepts relevant to data exfiltration attacks and defenses are introduced.

Table 2.1 presents the notation of the most common variables and parameters used throughout this work. The subsequent sections provide an explanation of the meaning and relevance of each of the terms.

2.1 Supervised Machine learning

Computer scientist and machine learning pioneer Tom Mitchell defined Machine Learning as follows:

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*

This definition captures the essence of machine learning, which involves the development of algorithms that enable a program to learn and become more proficient or skilled at the tasks it is designed to do through the accumulation of experience.

Machine learning is a subfield of artificial intelligence that studies the development and application of algorithms capable of automatically learning patterns and making predictions or decisions from data. It involves training computational models on datasets, allowing them to capture complex relationships and extract relevant features, and thereby generalize from the examples provided. This comprises automatic methods that require

Table 2.1: Notation used in this work

Notation	**Meaning**
$\mathbf{D}$	Dataset
$\mathbf{X}$	Attributes of the dataset
$\mathbf{y}$	Target attribute of the dataset
m	Number of attributes in D
f	ML model
θ	Model parameters
θ'	Model parameters modified by attack
θ'^{*}	Model parameters modified by defense
$\mathbf{W}$	Model weights
$\mathbf{b}$	Model biases
L	Number of hidden layers
$\mathbf{h}$	Hidden layer
α	Learning rate
$\mathbf{s}$	Secret (to be exfiltrated)
b	Number of least significant bits
$\mathbf{A}$	Neuron activations
$\mathbf{T}$	Trigger set
$\mathbf{X}'$	Generated trigger set attributes
$\mathbf{D_{ex}}$	Exfiltrated dataset
$\mathcal{T}$	Benign training algorithm
$\mathcal{A}$	Data transformation algorithm
$\mathcal{T}_{\text{mal}}$	Malicious training algorithm

no specification of rules to learn and no external assistance in learning. One of the sub-fields in machine learning, relevant to this work, is supervised learning. Supervised learning involves mapping input data to known output labels in the training process. These labels are also called the target variable (or attribute) - it is what the algorithm ultimately aims to predict. Thus the mapping resulting from the training process should ultimately correctly predict the value of the target variable on data where the output label is unknown.

2.2 Classification

In supervised machine learning, classification is a task where the objective is to assign input data points to predefined categories (also called classes). The algorithm learns from labeled training data $X = ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$, where each input sample $\mathbf{x_i} = (x_i1, \ldots, x_im)$ (consisting of values belonging to input features, also called variables or attributes, of size m) is associated with a known class label y_i. This known mapping to the target variable is also referred to as ground-truth. The full dataset can be represented as a matrix:

$$\mathbf{D} = \begin{bmatrix} \mathbf{x}_{1,1\dots m} & y_1 \\ \mathbf{x}_{2,1\dots m} & y_2 \\ \vdots & \vdots \\ \mathbf{x}_{n,1\dots m} & y_n \end{bmatrix}$$

During the training phase, the classification algorithm aims to learn the input features and capture their relationship to the corresponding labels. Through this learning process, the model parameters θ are adjusted to find decision boundaries between the classes of the training dataset. Optimally, the trained model $\hat{f}$ should capture patterns and relationships in the training data well enough to assign the correct label to given data samples, while at the same time still being able to generalize and correctly classify similar, however previously unseen, data. Ideally, the trained model should approximate the true (unknown) function f (which fits the true patterns and relationships in the data) as closely as possible. In order to classify a new data point x_i, the trained model $\hat{f}$ utilizes the learned decision boundaries and assigns a specific class label to the data point, i.e. prediction $\hat{y}_i = \hat{f}(x_{i1}, x_{i2}, \dots, x_{im})$, where $0 \leq i \leq n$.

2.2.1 Binary Classification

Binary classification is a type of supervised classification where the target variable $\mathbf{y} = (y_1, \dots, y_n)$ of the underlying dataset consists of two classes - $y_i \in \{0, 1\}$ where y_i represents the sample label and $0 \leq i \leq n$. The target variable of the dataset we use in our evaluation is binary.

2.3 Model Performance Evaluation Metrics

The performance of a model on a classification task is evaluated by comparing the true labels of data samples to the corresponding predictions produced by the model. The difference between the true label and the prediction is denoted as error ε. The given original sample label y_i is thus compared to the model prediction $\hat{y}_i = \hat{f}(x_{i1}, x_{i2}, \dots, x_{im})$, so that $y_i = \hat{y}_i + \varepsilon$. Based on this comparison, various metrics can be calculated.

- **True Positive (TP)** prediction: A TP prediction is achieved when a sample belonging to a **positive** class is labeled as such by the model, thus the prediction matches the ground truth, i.e.
$$y_i = \hat{y}_i = 1$$
- **True Negative (TN)** prediction: A TN prediction describes the case, when a sample belonging to a **negative** class is labeled as such by the model, thus the prediction matches the ground truth, i.e.
$$y_i = \hat{y}_i = 0$$

- **False Positive (FP)** prediction: A FP prediction occurs when a sample belonging to a **negative** class is assigned a **positive** label by the model. In this case, the prediction does not correspond to the ground truth, i.e.

$$y_i = 0 \wedge \hat{y}_i = 1$$

- **False Negative (FN)** prediction: A FN prediction occurs when a sample belonging to a **positive** class is assigned a **negative** label by the model. Also in this case, the prediction does not correspond to the ground truth, i.e.

$$y_i = 1 \wedge \hat{y}_i = 0$$

- **Accuracy**: captures the proportion of correctly classified samples over the total number of samples.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Precision**: Precision quantifies the proportion of correctly predicted positive samples out of all predicted positive samples and is therefore defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**: Recall represents the proportion of TP that are correctly identified by the model. It is also known as sensitivity or a true positive rate (TPR) and is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1 Score**: The F1-score captures both precision and recall into one metric - it is calculated as the harmonic mean of these two metrics.

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Area Under the Receiver Operating Characteristic Curve (ROC AUC)**: plots sensitivity (the true positive rate (TPR)) against specificity (the false positive rate (FPR)), and thereby captures the ability of the model to distinguish between the samples of positive and negative classes across various classification thresholds.
  - True Positive Rate (TPR) (same as recall):

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  - False Positive Rate (FPR):

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

This metric is used to evaluate model performance on binary classification problems.

- **Zero-Rule (0R) Baseline**: This baseline represents the performance of a simple, naive classifier, indicating the performance when, for any input, the model returns the label of the most-frequent class of the target variable as a prediction, i.e. the accuracy is equal to the proportion of the most-frequent class in the data.

2.4 Multilayer Perceptron

The Perceptron algorithm proposed by F.Rosenblatt in 1958 [57] is a fundamental algorithm in the field of Machine Learning. It is a type of linear binary classifier that aims to distinguish between two classes of data. It consists of an artificial neuron that maps an input vector to an output value by calculating the weighted sum of the input features. This sum is then passed through an activation function, which determines the output of the perceptron. During the learning process, the perceptron adjusts its weights based on the errors made in classification by comparing its predictions with the correct class labels, and updating the weights accordingly if there is a misclassification. A Multilayer Perceptron (MLP) is a more complex form of the MLP is a type of fully-connected, artificial neural network (FCNN) that consists of multiple layers of interconnected artificial neurons. It is a feedforward neural network, meaning that information flows in one direction, from the input layer through the hidden layers to the output layer.

MLP architecture typically comprises an input layer, one or more hidden layers, and an output layer. The layer size is determined by the number of neurons per layer. Typically, each hidden layer consists of multiple neurons. The neurons in adjacent layers are fully connected, meaning that each neuron is connected to every neuron in the previous and subsequent layers. The size of the input layer depends on the size of the input data (the number of features in **X**). The neurons finally feed into the last layer, the so-called output layer, where the prediction of the network for the given sample is produced.

In an MLP, each neuron receives input signals from the previous layer, applies a weighted sum of the inputs, and applies an activation function to produce an output. The activation function introduces non-linearity, allowing the MLP to learn complex relationships between input data and the target variable. As the produced weights determine the slope of the resulting curve, additional parameters - biases, can be used in order to shift the resulting curve, to further reduce the classification error.

The Multilayer Perceptron can also be represented as follows:

- Input Layer: $\mathbf{x_i} = (x_{i1}, x_{i2}, \ldots, x_{im})$, where $\mathbf{x_i}$ represents an input sample, with $(x_{i1}, x_{i2}, \ldots, x_{im})$ representing the feature values of that sample and m represents the number of features in the data $\mathbf{X}$. Therefore, the size of the input layer is determined by m.

- Hidden Layers: $\mathbf{h}^{(l)} = (h_1^{(l)}, h_2^{(l)}, \dots, h_j^{(l)})$ (for $l = 1, 2, \dots, L$), where $\mathbf{h}^{(l)}$ represents the l^{th} hidden layer, and L represents the number of hidden layers. $h_i^{(l)}$ stands for each neuron, with $1 \leq i \leq j$ and j representing the number of neurons in each layer (hidden layer size).

- Output Layer: produces the prediction $\hat{y}_i$ for each sample $x_1, x_2, \dots, x_n$ that is passed through the network.

The first step of the training process of an MLP is the initialization of weights and biases for each neuron in the network. The weights and biases of an MLP can be represented as matrices:

$$\text{Input-to-Hidden Layer Weights:} \quad \mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \dots & w_{1j}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \dots & w_{2j}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^{(1)} & w_{m2}^{(1)} & \dots & w_{mj}^{(1)} \end{bmatrix}$$

$$\text{Hidden-to-Hidden Layer Weights:} \quad \mathbf{W}^{(l)} = \begin{bmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \dots & w_{1j}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \dots & w_{2j}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j1}^{(l)} & w_{j2}^{(l)} & \dots & w_{jj}^{(l)} \end{bmatrix}$$

$$\text{Hidden-to-Output Weights:} \quad \mathbf{W}^{(L+1)} = \begin{bmatrix} w_1^{(L+1)} \\ w_2^{(L+1)} \\ \vdots \\ w_j^{(L+1)} \end{bmatrix}$$

$$\begin{aligned} &\text{Hidden Layer Biases:} \quad \mathbf{b}^{(l)} = (b_1^{(l)}, b_2^{(l)}, \dots, b_j^{(l)}) \quad (\text{for } l = 1, 2, \dots, L) \\ &\text{Output Layer Biases:} \quad \mathbf{b}^{(L+1)} = (b^{(L+1)}) \end{aligned}$$

The weights and biases are generally referred to as parameters of a model $\theta = \{\mathbf{W}, \mathbf{b}\}$. The size of these matrices, and thus the number of parameters in a model is determined by the number of input features, number of layers in the network, and number of neurons in each layer. As each neuron in a layer is connected to every neuron in the previous layer, the number of weights in a layer is the product of the number of neurons in that layer and the number of neurons in the previous layer. Each neuron also has a bias associated with it, so the number of biases in a layer is equal to the number of neurons in that layer.

MLP contains an activation function in each neuron. In this thesis, we employ the Rectified Linear Unit activation function as the hidden layer activation function, which is defined as the maximum of 0 and the input, where the v input denotes the input to the neuron:

$$\mathrm{ReLU(v)} = \max(0, \mathrm{v}) = \begin{cases} v & \text{if v} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The Sigmoid activation function is applied as the output layer activation function and is defined as:

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

where v represents the input to the function. This function maps values to the range $(0, 1)$ and is therefore commonly used in the context of binary classification tasks, as the resulting value can be interpreted as the probability of the input belonging to the positive class.

A forward pass refers to a process of calculating an output given an input. It consists of propagating the input through the layers of the network in a forward manner and does not include the update of the model parameters. The input data is represented as a vector, with each vector element corresponding to a feature value of the data. In the first step, data enters the network and is propagated through each layer. Each neuron computes a weighted sum of its inputs (a dot product with the weights, plus the bias). Subsequently, the activation is computed. Then, the network provides output based on its inputs.

Therefore, the MLP forward pass can be represented as:

$$\begin{aligned} \mathbf{a}^{(1)} &= \mathrm{ReLU}\left(\mathbf{x}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right) \\ \mathbf{a}^{(l)} &= \mathrm{ReLU}\left(\mathbf{a}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}\right) \quad \text{for } l = 2, 3, \ldots, L \\ \mathbf{o} &= \sigma\left(\mathbf{a}^{(L)}\mathbf{W}^{(L+1)} + \mathbf{b}^{(L+1)}\right) \end{aligned}$$

We can then compute how much off the MLP is with its prediction; the loss function relevant to the model architecture used within the experiments in this thesis is the Binary Cross-Entropy Loss (denoted as *Loss*).

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where $\hat{y}_i$ represents the output of the model for the i^{th} input sample. It shows the predicted probability of the i^{th} sample belonging to the positive class of the target variable. This is summed up over all samples in the dataset.

During the training process, the objective is to find the set of weights and biases that minimize the loss, which is typically achieved by using an optimization algorithm, such as stochastic gradient descent.

In order to find the optimal set of parameters, the computed error from the loss function is propagated from the final layer through the network in a backward manner, in a process referred to as backpropagation. The gradients of the loss function with respect to the weights and biases (parameters θ of the model) are computed; the gradient of the function at a given point indicates the steepest increase of the function at that point. Subtracting the gradient from the parameters, therefore, leads to a step in the direction of the steepest decrease in the loss function. In this manner, the parameters are consequently updated, depending on how much each of the parameters contributed to the error. The update rule for weights and biases is given as:

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial w_{ji}^{(l)}}$$

$$b_{j}^{(l)} = b_{j}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial b_{j}^{(l)}}$$

where α is a hyperparameter controlling the magnitude of the update, also referred to as the learning rate.

During the training process, the input data is fed to the network in batches. The size of a batch is a hyperparameter of the model which can be adjusted to find the optimal value. It is primarily used in the training of a model and refers to the number of training data samples that are used for one set of a forward pass and a backward pass of the data within one training This process, from the forward propagation, until the parameters are updated, for all batches of the dataset, i.e. until each input sample has been presented once to the MLP training, is referred to as one training epoch. The training process is repeated for a number of epochs, upon which the final model is obtained.

During training, a validation set can be used in order to assess the model's performance on unseen data, and, if desired, to implement early stopping of the training process. The calculated loss on the validation set is tracked. Once it is no longer showing improvement upon a training iteration, the training can be stopped.

Algorithm 2.1 depicts the training process of a Multilayer Perceptron.

The parameters θ of the final model constitute the learned representation of the input data. Each parameter has a numerical value.

2.5 Hyperparameter tuning

Hyperparameter tuning is a process of adjusting hyperparameters in a machine learning model in order to optimize its performance on a specific task. In MLP, the hyperparameters to optimize are, for example, the learning rate. By conducting hyperparameter performance evaluation, the model's hyperparameters can be fine-tuned to achieve a balance between overfitting and underfitting, resulting in improved model performance. It is usually carried out by training the model with different combinations of hyperparameter values and evaluating its performance on a validation set.

Algorithm 2.1: Benign training algorithm $\mathcal{T}$

```
Input: Training data $[\mathbf{X}_{train} \,|\, \mathbf{y}_{train}]$, Validation data $[\mathbf{X}_{val} \,|\, \mathbf{y}_{val}]$, learning rate
       $\alpha$, batch size $N$, total epochs E
Output: Trained model $\hat{f}$
Initialize network weights $\mathbf{W}^{(l)}$ and biases $\mathbf{b}^{(l)}$ randomly for all layers $l$
Let $Loss_{min}$ be the minimum validation loss observed so far, initialized to $\infty$
Let $patience$ be the maximum number of epochs to wait for an improvement in
  validation loss
Let $epochs\_since\_last\_improvement$ be the number of epochs since the last
  improvement in validation loss, initialized to 0
for $epoch$ in $E$ do
    for $batch$ in $[\mathbf{X}_{train} \,|\, \mathbf{y}_{train}]$ do
        Forward pass:
        $\hat{\mathbf{y}}_{train} \leftarrow$ forward_pass$(f, \mathbf{X}_{train})$
        Compute loss:
        $Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_{\text{itrain}} \log(\hat{y}_{\text{itrain}}) + (1 - y_{\text{itrain}}) \log(1 - \hat{y}_{\text{itrain}})]$
        Backward pass (backpropagation):
        Compute gradients: $\frac{\partial Loss}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial Loss}{\partial \mathbf{b}^{(l)}}$ for all layers $l$
        Use optimizer to update parameters $\theta$:
        $\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha \frac{\partial Loss}{\partial \mathbf{W}^{(l)}}$ for all layers $l$
        $\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \frac{\partial Loss}{\partial \mathbf{b}^{(l)}}$ for all layers $l$
    end
    if $[\mathbf{X}_{val} \,|\, \mathbf{y}_{val}]$ is not None then
        Forward pass:
        $\hat{\mathbf{y}}_{val} \leftarrow$ forward_pass$(f, \mathbf{X}_{\text{val}})$
        Compute $Loss_{\text{val}}$, the loss on validation_data
        $Loss_{\text{val}} = -\frac{1}{N_{\text{val}}}\sum_{i=1}^{N_{\text{val}}}[y_{i_{\text{val}}} \log(\hat{y}_{i_{\text{val}}}) + (1 - y_{i_{\text{val}}}) \log(1 - \hat{y}_{i_{\text{val}}})]$
        if $Loss_{val} < Loss_{min}$ then
            $Loss_{min} = Loss_{val}$
            $epochs\_since\_last\_improvement = 0$
        else
            $epochs\_since\_last\_improvement \pm 1$
            if $epochs\_since\_last\_improvement \geq patience$ then
                Break
            end
        end
    end
end
```

2.5.1 Grid Search

Once the values to be investigated for each hyperparameter are determined, a grid of the parameters is constructed. During a Grid Search, a model is trained using each combination of the hyperparameter values from the given grid. Then, the performance of each of the models is evaluated, and the best-performing model is chosen.

2.5.2 Cross-validation

A straightforward approach to the evaluation of model effectiveness is the holdout method, which splits the data into two parts - a training set and a validation set. The model is trained on the training set, and its performance is evaluated on the validation set. However, the resulting calculated performance of the model on this validation set is highly dependent on how the holdout split is performed (the calculated performance metric might be particularly high or low on this one specific subset of the data, and not reflect the true model effectiveness). In order to increase the robustness of the evaluation, ensuring a more reliable estimate of the model's performance on unseen data, cross-validation (CV) can be employed. Standard practice is to employ k-fold cross-validation, a method that partitions data into k parts, where k-1 parts are used for building a model and one part is retained for performance evaluation. This step is repeated k times, to ensure a reliable estimate of the model's performance on unseen data. The results over k folds are then combined (e.g. averaged) to produce a single performance estimation. The application of cross-validation is important for a more robust model selection.

2.6 Data Preprocessing

In order to build a model based on training data, the underlying data must (in most cases) undergo transformation. The necessary preprocessing steps depend on the nature and type of the data, as well as the algorithm that will be used for classification. In the first step, the dataset needs to be investigated to determine the preprocessing steps necessary.

2.6.1 Data Cleaning

The dataset might contain missing values, outliers, or noisy data. Handling missing values typically comprises of imputation, in some cases also the removal of the respective features or samples. This step, however, needs to be performed carefully, taking the impact of the specific preprocessing step into account. Outliers should be carefully detected and investigated, as they might skew the data in an unwanted or unnatural way.

2.6.2 Encoding

For the algorithm to handle categorical values, especially if they are represented in the dataset as strings, label encoding is necessary. It assigns a number to each category within the feature. However, this type of encoding is not always ideal, as it introduces order to categories, which are not necessarily ordinal or end up being encoded in the wrong order. Therefore, preprocessing requires careful inspection of the data to determine which type of preprocessing is suitable. Another way to preprocess categorical data is one-hot encoding. When applied, a separate, binary column is created for each category of each categorical feature. The disadvantage is that this type of encoding increases the dimensionality of the data and introduces multicollinearity into the dataset. For this reason, one of the newly created columns per original categorical feature should be removed in order to avoid linear dependency in the data.

2.6.3 Feature Scaling

Another step critical to data preprocessing is feature scaling. It ensures equal contribution of the different features to the subsequently built machine learning model. Scaling involves either normalizing the data by transforming it into a chosen range (min-max scaling) or standardizing it by centering the values around the mean with a standard deviation of one.

2.7 Representation of numerical values

The model parameters are represented as single-precision floating-point numbers using the IEEE-754 Standard [33]. According to IEEE-754, the single-precision floating-point format (32-bit) consists of:

- Sign bit
- Exponent consisting of eight bits
- Significand precision (also known as the mantissa) of size 23 bits

Endianness refers to the order used to numerically interpret a range of bytes in computer memory. It describes how multi-byte data (floating-point numbers in the case of model parameters) are stored and read in computer systems. The two main types are Big Endian, where the most-significant bit is written at the smallest memory address, and its opposite, Little Endian. Due to the fact that the IEEE-754 Standard does not specify endianness, it is important to determine the endianness in order to pinpoint where the least significant bits (LSBs) are stored.

2.8 Error Correction Codes

Error Correction Codes (ECC) provide a way to detect and correct data corruption in digital data transmission and storage. The detection and correction are performed by adding extra symbols to the data. A widely utilized type of ECC is the Reed-Solomon error correction. This method utilized block-based ECCs, meaning that blocks of bits are grouped together into larger symbols. This property makes them effective at dealing with burst errors, which are errors that affect many consecutive bits or larger failures which affect bytes at a time. Reed-Solomon ECCs have a wide range of applications, such as mobile or satellite communications, or for example, bar-code scanning, including QR-Codes.

2.9 Data Exfiltration Attacks Application Scenarios

Data exfiltration functionality could potentially be embedded in a module of an ML library, or in an application that provides model-building functionality. Furthermore, in settings where it comes to a code exchange, this can potentially be interjected by a man-in-the-middle attack, e.g. with an adversary impersonating the code-provider party, replacing the benign code with malicious one.

2.9.1 Federated Learning

Federated learning can pose a special scenario for the application of DEA. Although federated learning is a privacy-preserving machine learning technique that eliminates the need for data sharing, there are still privacy implications regarding the data. In order to avoid data sharing, a machine learning model is trained locally by a number of participants (e.g. devices) on their sensitive data [40] [12]. Subsequently, these models are aggregated in order to build a final global model. The models can be trained in a parallel manner, where participants train models simultaneously, and subsequently share the models with a central aggregating entity. In the case that the code was provided by this aggregator, it could potentially contain exfiltration functionality. A second federated learning configuration is so-called sequential learning, where the participants train the models consecutively so that the global model is updated, before being sent to another participant. Such a model could still potentially be at risk for data exfiltration attacks.

2.10 Security risks

Confidentiality, integrity and availability have been identified as three key components of information security, commonly referred to as the CIA-triad [60, 65]. Confidentiality has been defined through limiting access to information ensuring its non-disclosure. The integrity component aims to guarantee that the information is accurate and trustworthy. Reliable access to information by authorized individuals is ensured through the component of availability. Potential security risks can be classified according to which of the three components they violate. In adversarial machine learning, threats are also classified according to these principles [5, 10]. Chapter 3 provides more details on how these concepts relate to security in machine learning.

CHAPTER 3

State-of-the-Art

3.1 Adversarial Machine Learning

Barreno et al. first introduced a taxonomy of adversarial attacks against learning systems [5]. The attacks are classified along three dimensions - influence, specificity, and security violation. Attacks are classified as causative if the attacker influences the learning process with control over training data, and exploratory, if there is no influence on the training process, but the attack exploits misclassifications. The axis of specificity partitions the attacks into two categories - attacks targeting to compromise the model's performance on a particular instance of the data, or attacks targeting data instances indiscriminately. As introduced in Chapter 2, attacks against machine learning can be categorized based on which element of information security an attack aims to compromise with respect to the underlying model. The taxonomy introduced by Barreno et al. [5] distinguishes between

Table 3.1: Categorization of adversarial attacks against ML (adapted from [10])

	Attacker's Goal		
	Misclassifications that do not compromise normal system operation	Misclassifications that compromise normal system operation	Querying strategies that reveal confidential information on the learning model or its users
Attacker's Capability	**Integrity**	**Availability**	**Confidentiality**
Test data	Evasion (a.k.a. adversarial examples)	-	Model extraction / stealing and model inversion (a.k.a. hill-climbing attacks), Data Exfiltration abusing unintentional memorization
Training data	Poisoning (to allow subsequent intrusions) – e.g., backdoors or neural network trojan	Poisoning (to maximize classification error)	Data Exfiltration by Steganography/Intentional Memorization

integrity and availability attacks. Later, Biggio et al. expanded on this classification, by adding a third class of attacks, comprising confidentiality [10]. This classification is shown in Table 3.1.

3.1.1 Integrity Violation Attacks

Violating the integrity of a system, in this case, means harmful instances are being classified as false negatives, thus not recognized by the system.

Backdoor attacks via Data poisoning

Data poisoning is an attack that involves influencing or controlling (a part of) training data by the attacker, whereby eventually, the model's behavior is manipulated [58, 47]. This way, a backdoor is inserted into the learning process. Such attacks require the attacker to have the ability and access to intervene directly at training time. If misleading data (i.e. poisoned samples) is injected into the learning process, the algorithm might adopt incorrect behavior. The target label of the modified instances is specified by the adversary. The attack is successful, when the modified samples are labeled by the attacked model, with the class assigned by the adversary [17, 25].

Evasion attacks

Evasion attacks differ from data poisoning attacks by the fact in which stage the ML process is modified. Evasion attacks are executed at the testing stage, assuming adversary has no access, and thus no influence on the training process. The aim is to modify the input sample to cause unusual behavior at inference time. For example, a gradient-based approach was proposed to modify the input samples to purposefully lead to misclassification [9].

3.1.2 Availability Violation Attacks

Based on the adversary's objective, a subset of poisoning attacks belongs to this category. In this subset, the objective is to harm system availability causing denial of service, by maximizing incorrect classification of benign instances.

3.1.3 Confidentiality Violation Attacks

The attacks from this category aim to compromise the confidentiality of models, or the respective underlying data.

Model Stealing

The objective of model stealing attacks is to infer information about a model [66, 50]. By default, the premise for model stealing attacks is black-box access to the target model. According to the taxonomy of model stealing attacks introduced in [50], these aim to steal either exact model properties or to approximate the model's behavior. This is generally performed by querying the prediction API, although the adversary's access to the underlying computing resources might introduce further vulnerabilities, via side channels, the attacker can potentially exploit.

Model Inversion

Taking into account not only the confidentiality of a model but the training data as well, model inversion attacks are added to this categorization as well. This type of attack tries to gain information about the samples included in the training data. A black-box access to the model is assumed, with the adversary only being able to perform modifications to the input samples at inference time. For example, this attack has been carried out on models trained on image data, where images of individuals included in a training set of a model were able to be (partially) recovered by querying the model's API given an individual's name only [22].

Inference Attacks

Similarly to the model inversion attacks, inference attacks aim to compromise data confidentiality (or privacy) by attempting to infer information about training data.

- **Property Inference Attack** The goal of property inference attacks is also to gain information about the training data set [23]. However, these attacks aim to find general properties of the training data (e.g. a ratio of smokers vs. non-smokers), and not on the individual data rows.

- **Membership Inference Attack** In a membership inference attack, given a model, an adversary tries to determine whether a certain sample has been included in the training set [62, 30, 34]. The main difference to a property inference attack is the knowledge of the data sample.

Data Exfiltration Attacks

We have included data exfiltration attacks in the classification of adversarial attacks in Table 3.1, as they are a threat to the confidentiality of the data upon which the models are built on. The data exfiltration attacks that abuse the unintentional memorization of the models are categorized as attacks, where the attacker has influence over the test data only. The attacks within this thesis belong to the category where the attacker has influence over the training data, however, in this case, the modification happens in order to hide the data in a model. The nature of the attacks is further explained in Section 3.6, Section 4.2 and Section 4.3.

3.2 Steganography

Steganography is a field focused on information-hiding techniques. As defined by a popular quote, the information should be hidden in a way so that "The message does not attract attention to itself as an object of scrutiny" [unknown]. Therefore, the aim here is not only to keep the hidden information a secret but on keeping the existence of the hidden information a secret [46]. This quote thus hints at the imperceptibility and secrecy of hidden data. Steganography has been applied, for example, as a method for communication of concealed information. Information can be hidden in various media types such as text [51] or image data [46, 27, 1]. Additionally, statistical models have been used to make steganographic method more secure against detection of concealed secret data in media files, primarily in JPEG images [59]. One of the proposed approaches to hide information is to replace the least significant bits of the chosen medium with the secret data [27, 28].

3.3 Malicious Code hiding

Cabaj et al. provided an overview of information-hiding techniques used to conceal malware [13]. As one of the subdomains, they present hiding code by obfuscation and masquerading techniques. For example, code obfuscation is used by adversaries to evade the detection of malware [76]. Furthermore, steganography has been used also as a method to hide malicious code. Adversaries can utilize images as a covert channel for their malicious code [36].

Steganographic techniques have been employed to hide malicious code within a smartphone application [64]. The authors implement a prototype of such an application, and demonstrate the feasibility of their approach by their code not being detected in "Google Play".

In order to camouflage malware, encryption is another option adversaries use [6]. In this case, the code is only decrypted and executed at runtime, which makes the static analysis of the codebase ineffective. The existence of numerous techniques enabling the hiding of malware shows that an adversary can potentially successfully hide code in order to carry out data exfiltration.

3.4 Watermarking & Fingerprinting

Other application scenarios of information hiding include methods such as watermarking [18, 4] and fingerprinting [71].

3.4.1 Watermarking

This type of information embedding is widely known especially due to image watermarking for the purpose of intellectual property and copyright protection. The addition of a watermark is, however, much older than digital media, not restricted to images, and can be utilized in various types of media. Thus, besides the application on image data [52], it has been applied, for example to relational data [4, 3], audio [7, 32] and text data [37].

The purpose of embedding a watermark in a medium is content protection, owner verification, and ensuring data provenance [4, 38, 42]. As the training of machine learning models, especially deep neural networks (DNNs) becomes more complex and expensive, the trained models are more valuable. A way to protect the model copyright, or to detect infringement of intellectual property ownership is to utilize model watermarking [42]. Watermarking machine learning models is one of the most relevant applications of watermarking to the topic of data exfiltration attacks. Figure 3.1 presents the taxonomy of watermarking (and fingerprinting) schemes for machine learning models introduced in [42].

Depending on the threat model, a watermark serves for ownership verification of an illegal copy of a model obtained through model extraction methods, or of legally obtained copies that are illegally re-distributed. Generally, there are two categories of watermarking ML models based on the type of model access necessary for extraction and verification of a watermark [11, 56, 42].

- **White-Box**: a watermark is embedded directly into the model parameters or their probability density function (PDF) [67, 19, 16], or
- **Black-box**: additional training samples are generated that trigger model's response to diverge from the prediction behavior on original samples [2, 45, 19, 77]. The techniques to generate trigger samples are also shown in Figure 3.1. In this case, only the model owner possesses the knowledge to generate the trigger set, which allows for the verification of a watermark.

White-Box Watermarking

Uchida et al. propose watermarking by regularization. In this approach, a watermark is embedded into a model during training, by adding a regularization term to the cost function. The regularizer imposes a statistical bias on the parameters, serving as a watermark [67]. Later, [19, 16] extended these schemes to make watermark detection and removal more difficult.

Black-box Watermarking

Given the condition of black-box access to the model (for watermark extraction and verification), Zhang et al. investigate three distinct methods for generating watermarks [77]. The methods differ in how the content of the trigger samples is crafted. They incorporate either meaningful data samples (pattern-based crafting), data samples unrelated to the current task (out-of-distribution (OOD)), or use noise during the training phase and utilize it as a watermark. They showed that the models with embedded watermarks have

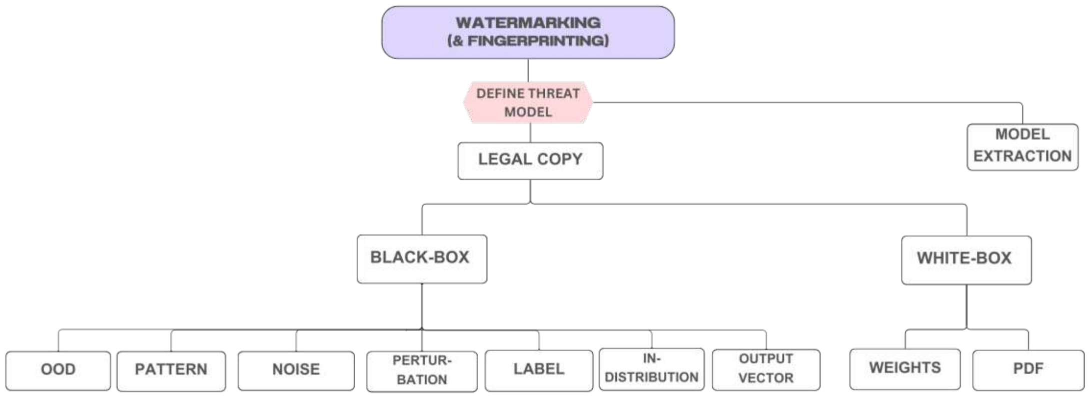


Figure 3.1: Taxonomy of watermark embedding in neural networks as introduced in [42]

comparable performance to the base models. In a similar approach, Adi et al. proposed watermarking deep neural networks used for image classification [2]. Their approach is based on a backdooring attack, aiming to make the model output specific incorrect labels for a chosen set of inputs. For this purpose, trigger set images are constructed and assigned labels. The task of the model is to learn to classify the trigger set images in addition to the underlying training data. They investigate two approaches, either training a model with the trigger images from scratch or continuation of training of a pre-trained model. They discuss the importance of limiting the size of trigger sets, in order to preserve the performance of the model on the original task. Generating adversarial examples (i.e. an evasion attack) for instances near the classification boundary and assigning a class label from the other class represents another way to embed a watermark into a model [41]. Another option is to introduce a new label to the trigger images, instead of using a label from the original target attribute [77].

3.4.2 Fingerprinting

Fingerprinting, as opposed to watermarking, also allows for identifying the recipient and thus ensures traceability (of the data/model). This can deter unauthorized data or model sharing and fend off potential leaks. Similarly to watermarking, fingerprinting ML models is relevant to data exfiltration due to the data embedding component. Chen et al. proposed DeepMarks [16], a system that creates unique identifiers, known as fingerprints, for each user. For verification, this system assumes white-box access to the model. These fingerprints are integrated into the structure of a neural network during its training process. This integration is achieved by adjusting the network's weights based on a specific loss function related to the fingerprint.

3.5 Watermarking model output

With the emergence of Large Language Models (LLMs), there is also an accompanying rise in potential risks and harms caused by the employment of such models [69, 20]. Watermarking the output of the LLMs has been proposed as one of the mitigation strategies [38, 42], which allows for the recognition of content generated by these models. Watermarking the model output (predictions) is usually also used as a mitigation strategy against model stealing attacks [42].

3.6 Data Exfiltration

Data exfiltration is generally defined as a security or privacy breach causing an unauthorized leak of data or data theft. Traditionally, what is referred to as data exfiltration attacks are attacks targeting web traffic, for example exploiting various network protocols [68], or vulnerabilities in Internet-of-Things devices [21].

When it comes to data exfiltration from machine learning models, an adversary can utilize such data-hiding techniques, or abuse the training data memorization potential of models.

3.6.1 Categorization of Data Exfiltration Attacks in Machine Learning Models

When an adversary gains control an ML pipeline data can be hidden in the model itself, or the model can be forced to memorize data - i.e. causing intentional memorization. In other cases, an adversary can attempt to exfiltrate training data from a trained model where no control over the pipeline was assumed. Therefore, we introduce a taxonomy of exfiltration attacks, classifying them as shown in Figure 3.2. As described above, we distinguish between attacks that are based on steganography, and attacks that utilize the memorization of the models. Additionally, we indicate the attacks where an adversary needs to influence the model training process (denoted by green color) in order to carry out an attack.

3.6.2 Data Exfiltration Attacks based on steganography

The research by Song et al. explores the data-hiding techniques caused by algorithms that are modified to be malicious [63]. Figure 1.1 shows that the adversary can provide malicious code to control the ML pipeline. Figure 3.3 depicts the steps in a ML pipeline. The red dotted rectangle indicates the parts of the ML pipeline which an adversary may control, i.e. training data transformation or model creation. The authors have performed three types of attacks in the white-box scenario (Least Significant Bit Encoding,

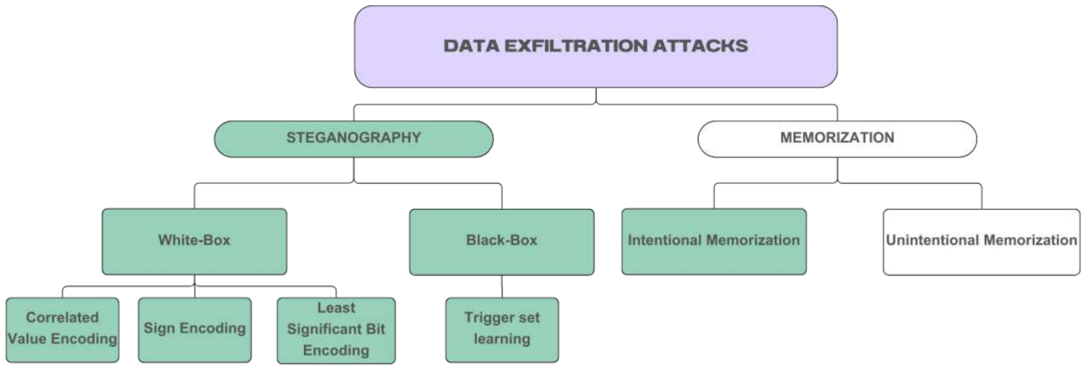

DATA EXFILTRATION ATTACKS
STEGANOGRAPHY
MEMORIZATION
White-Box
Black-Box
Intentional Memorization
Unintentional Memorization
Correlated Value Encoding
Sign Encoding
Least Significant Bit Encoding
Trigger set learning

Figure 3.2: Taxonomy of Data Exfiltration Attacks: The green color denotes scenarios in which an adversary has active control over the training process

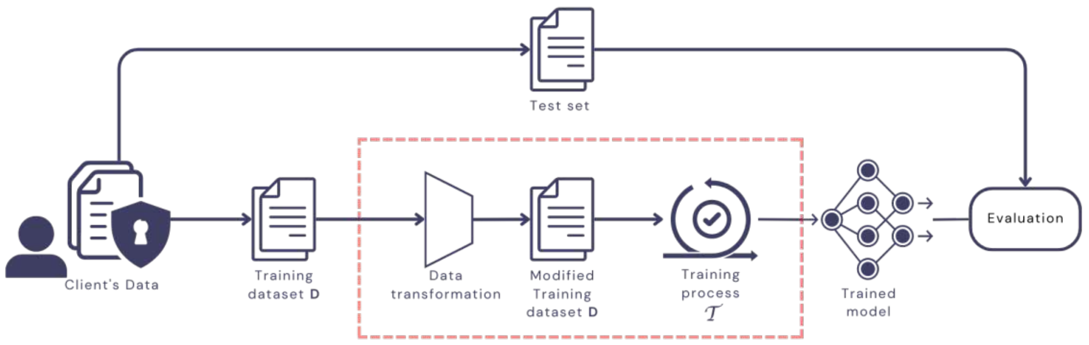


Figure 3.3: ML pipeline with a depiction of the parts an adversary can control, adapted from [63]

Correlated Value Encoding, Sign Encoding), and one attack in a black-box scenario abusing the learning capacity of the model. The malicious algorithms used for white-box attacks extract a secret, e.g. a binary vector or a vector of real numbers, from the training data, and encode this vector into the model parameters, either by replacing the least-significant bits of the parameters with the binary vector values, correlating the parameters to the values of the extracted vector, or by matching the signs of the parameters to the secret vector. The adversary then reads the parameters to extract the secret.

Least Significant Bit Encoding

Least Significant Bit Encoding attack uses a steganographic approach to encode information into machine learning models. These attacks use the model parameters as a covert channel for the training data exfiltration. As neural networks consist of numerous parameters, their capacity is especially high. However, as these parameters determine the output of the neural network, modifying them has an impact on the performance of their predictive functionality. Han et al. have, however, shown, that neural networks can be compressed by reducing the number of bits used to represent the parameters, and thus not all 32 bits representing a parameter are necessary to preserve the model's performance [29]. Further studies explored reducing the parameter size by binarizing the weights [43, 55]. Their findings point to the idea that the LSBs of parameters represented by floating point values can be used for hiding secret data, and the unnecessary bits can easily be exploited without compromising the network's performance.

Algorithm 3.1: LSB encoding attack as introduced in [63]

```
Input: Training dataset $\mathbf{D}$, a benign ML training algorithm $\mathcal{T}$, number of bits
       $b_a$ to encode per parameter.
Output: ML model parameters $\theta'$ with secrets encoded in the lower $b_a$ bits.
$\theta \leftarrow \mathcal{T}(\mathbf{D})$
$\ell \leftarrow$ number of parameters $\theta$
$s \leftarrow \mathbf{ExtractSecret}(\mathbf{D}, \ell, b_a)$
$\theta' \leftarrow$ set the lower $b_a$ bits in each parameter of $\theta$ to a substring of $s$ of length $b_a$
```

Data hiding The steps of this attack, as implemented in [63], are shown in Algorithm 3.1. First, the attacker trains a benign model on the training data using the benign training algorithm $\mathcal{T}$. Second, the attacker calculates the number of parameters in the model, which, multiplied by the amount of least significant bits the attacker intends to use for hiding, determines the capacity - how many bits of training data can be hidden.

Subsequently, the attacker transforms the dataset $\mathbf{D}$ into its bit representation $\mathbf{S}$:

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}_{1,1,\ldots,m+1} \\ \mathbf{s}_{2,1,\ldots,m+1} \\ \vdots \\ \mathbf{s}_{n,1,\ldots,m+1} \end{bmatrix}$$

This is done by representing each value of the dataset $\mathbf{D}$ so that

$$x_{ij} \mapsto \mathbf{s}_{ij} \quad \text{where} \quad i \in \{1, 2, \ldots, n\}, \quad j \in \{1, 2, \ldots, m\}, \quad \text{and} \quad \mathbf{s}_{ij} \in \{0, 1\}^{32}$$

Once the model is trained and the data transformed, the attacker replaces the least significant bits of length b_a of each parameter of θ with a substring of the secret s of length b_a.

Extraction In order to exfiltrate the data, the adversary reads the least significant bits of the published model, and reconstructs them to obtain exfiltrated dataset $\mathbf{D}_{ex}$. In this attack, the exfiltration leads to a 100% similarity of $\mathbf{D}_{ex}$ to $\mathbf{D}$.

Sign Encoding Attack

In a sign encoding attack, as implemented in [63], the signs of the parameters are utilized to hide the training data.

Data hiding The goal is to encode the data into the model so that a parameter with a negative value represents a bit 0, and one with a positive value represents bit 1 of the secret. The objective is to force the signs of the parameters to match the values of the extracted secrets. Such optimization problem can be addressed by introducing custom regularization [49]. In order to do so, a penalty term is defined:

$$P(\theta, s) = \frac{\lambda_s}{\ell} \sum_{i=1}^{\ell} \max(0, -\theta_i s_i)$$

In the penalty term, λ_s controls the magnitude of the penalty and ℓ represents the number of parameters in the model. Further, s represents the secret vector (i.e. bits of training data), with ℓ representing the length of this vector, as the capacity of the model is dependent on the number of parameters in the model, as one bit of the secret can be hidden per parameter. When the sign of θ_i and s_i is the same, zero penalty is added. If the sign of the secret bit and the parameter are not the same, $|\theta_i s_i|$ is the penalty.

Extraction In order to exfiltrate the data, the adversary interprets the signs of the parameters as bits of training data and reconstructs $\mathbf{D}_{ex}$. Again, it is not guaranteed to be identical to $\mathbf{D}$, depending, among others, on the strength of λ.

Correlated Value Encoding Attack

This attack gradually encodes the secret during the training process [63].

Data hiding Encoding the data into the model is achieved by adding a malicious term to the loss function (which is normally used to minimize the training errors), similar to a regularization term. The purpose of this malicious term is to maximize the correlation between the extracted secret s and the parameters of the model θ.

Algorithm 3.2: SGD with correlation value encoding as introduced in [63]

```
Input: Training dataset $\mathbf{D} = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{i=1}^{n}$, a benign loss function $Loss$, a model
       $f$ , number of epochs $E$, learning rate $\alpha$, attack coefficient $\lambda_c$, batch size
       $N$.
Output: ML model parameters $\theta'$ correlated to secrets.
$\theta \leftarrow$ Initialize($f$)
$\ell \leftarrow$ number of parameters in $\theta$
$s \leftarrow$ ExtractSecret($\mathbf{D}, \ell$)
for $e = 1$ to $E$ do
    for each batch $\{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=1}^{N} \subset \mathbf{D}$ do
        $\nabla_e \leftarrow \nabla_\theta \frac{1}{m} \sum_{j=1}^{N} Loss(y_j, f(\mathbf{x}_j, \theta)) + \nabla_\theta C(\theta, s)$
        $\theta \leftarrow$ UpdateParameters($\alpha, \theta, \nabla_e$)
    end
end
```

Algorithm 3.2 shows the necessary steps in order to hide the bits of the data. $C(\theta, s)$ represents the malicious term, that controls the parameter update. The malicious correlation term is defined as follows:

$$C(\theta, s) = -\lambda_c \cdot \frac{|\sum_{i=1}^{\ell}(\theta_i - \overline{\theta})(s_i - \overline{s})|}{\sqrt{\sum_{i=1}^{\ell}(\theta_i - \overline{\theta})^2} \cdot \sqrt{\sum_{i=1}^{\ell}(s_i - \overline{s})^2}}$$

In this malicious correlation term, λ_c controls the level of correlation, and $\overline{\theta}, \overline{s}$ represent the mean values of θ and s. The number of parameters is represented by ℓ As shown in Algorithm 3.2, first, the parameters of the model are initialized. Then, per each epoch, the loss (with the added malicious term) is calculated and parameters are updated using stochastic gradient descent (SGD). This training should result in a model that can solve the intended classification task, and where at the same time the parameter values are correlated to the training data the adversary wants to exfiltrate. The capacity of the model depends on the number of parameters, as well as the type of training data. For example, when the training data is numerical, and in the same range, the raw data can be used as secret s. This way, one parameter can represent one numerical value from the training data (e.g. a pixel value in image data).

Extraction In the case that the features of the underlying training data are numerical, the adversary can map the values of the parameters to reconstruct the training data. The exfiltration is not guaranteed to be perfect, i.e. $\mathbf{D}_{ex}$ is not always identical to $\mathbf{D}$, and the strength of the similarity is heavily influenced by the importance of the malicious term in the loss function.

3.6.3 Black-box Attack

The black-box attack assumes no access to the final published model parameters by the attacker.

Song et al. implement this attack on image data, as well as text data [63]. Within the Black-box Attack, the *learning* capacity of the network becomes the target of exploitation.

Algorithm 3.3: Attack in the black box scenario, adapted from [63]

```
**Input:** Training dataset $\mathbf{D} = [\mathbf{X} \,|\, \mathbf{y}]$, Malicious training algorithm $\mathcal{T}_{\mathrm{mal}}$, ratio of
        trigger samples to training data $trig\_ratio$, auxiliary knowledge $K$
**Output:** Trained malicious model $f'$
$g \leftarrow trig\_ratio * len(\mathbf{D})$
$\mathbf{T} \leftarrow \mathbf{GenerateTriggerSamples}(\mathbf{D}, g, K)$
Train $f' \leftarrow \mathcal{T}_{\mathrm{mal}}(\mathbf{D} \cup \mathbf{T})$
```

Data hiding As proposed by [63], the malicious algorithm builds a new input dataset $\mathbf{T}$ of length g:

$$\mathbf{T} = \begin{bmatrix} \mathbf{x'}_{1,1\ldots m} & y'_1 \\ \mathbf{x'}_{2,1\ldots m} & y'_2 \\ \vdots & \vdots \\ \mathbf{x'}_{n,1\ldots m} & y'_g \end{bmatrix}$$

by generating trigger samples $\mathbf{X}'$:

$$\mathbf{X}' = \begin{bmatrix} \mathbf{x'}_{1,1,\ldots,m} \\ \mathbf{x'}_{2,1,\ldots,m} \\ \vdots \\ \mathbf{x'}_{(g,1,\ldots,m)} \end{bmatrix}$$

in the shape of the original training set attributes $\mathbf{X}$, where the length g depends on how much training data is to be exfiltrated, and labeling them with bits of original training data. The trigger sample generation needs to be deterministic, to allow the attacker to recreate it for the purpose of exfiltration.

In order to label the trigger samples, the malicious algorithm extracts a secret string s from the training images by transforming them into their bit representation. The transformation is identical to the transformation for the LSB Encoding attack. The resulting matrix $\mathbf{S}$ is flattened into a vector $\mathbf{y}' = (y'_1, y'_2, \dots y'_g)$, so that y_i represents one bit (in binary classification, or more bits in multi-class classification) of the data. The vector elements are set as labels of the generated training samples $\mathbf{X}'$. $\mathbf{y}'$ thus constitutes the target attribute the model aims to predict. The generated set of trigger samples - the *trigger set* $\mathbf{T}$ - can be represented as:

$$\mathbf{T} = \{\mathbf{x}'_{i \dots m}, y'_i\}_{i=1}^{g}$$

The model is then trained on a union of the benign and trigger data $\mathbf{D} \cup \mathbf{T}$.

The goal of the attack is for the network to learn the representation of $\mathbf{X}'$ well enough so that prompting the trained model with a $\mathbf{x}'_i$ sample would elicit the correct corresponding y'_i label.

Algorithm 3.4: Trigger set generation, as introduced in [63]

```
Input: A training dataset $\mathbf{D}$, number of inputs to be generated $g$, auxiliary
       knowledge $K$.
Output: Generated trigger dataset $\mathbf{T}$
$\mathbf{T} \leftarrow \emptyset$
$s \leftarrow \mathbf{ExtractSecret}(\mathbf{D}, g)$
$c \leftarrow$ number of classes in $\mathbf{D}$
foreach $\lfloor \log_2(c) \rfloor$ bits $s'$ in $s$ do
    $\mathbf{x}' \leftarrow \mathbf{GenData}(K)$
    $y' \leftarrow \mathbf{BitsToLabel}(s')$
    $\mathbf{T} \leftarrow \mathbf{T} \cup \{(\mathbf{x}', y')\}$
end
```

Extraction In order to exfiltrate this data from a trained model, the attacker generates the trigger samples again, as depicted in Algorithm 3.4. This is possible, as the generation is deterministic, and the auxiliary knowledge K necessary for the generation is known to the attacker beforehand. The known, generated trigger data samples are sent to the prediction API. The target attribute, i.e. bits of the training data $\mathbf{D}$, are returned as the prediction. The adversary then reconstructs the dataset to create the $\mathbf{D_{ex}}$. Similar to the sign and correlated value encoding, also in the black-box attack is the exfiltration not guaranteed to be perfect, i.e. $\mathbf{D}_{ex}$ is not always identical to $\mathbf{D}$. This depends on a number of factors, such as the learning capacity of the model, or how the trigger data is generated, or how much emphasis is put on learning the trigger data vs. the benign training data.

3.7 Intentional and Unintentional Memorization

The main distinguishing point between attacks exploiting intentional and unintentional memorization in machine learning models are the characteristics of an attacker. Specifically, when the attacker has control over the training process, they can force the model to overfit on, and thereby memorize certain input samples. In attacks exploiting unintentional memorization, the adversary is assumed to have no influence on the training process of the model, as well as no access to the model parameters. Unintentional memorization in machine learning models has been researched, for example, on generative sequence models [14]. The adversary in this scenario does not have any influence on the algorithm but attempts to extract data purely by accessing a prediction API. The aim of exploring unintentional memorization in machine learning models is to explore memorization which occurs when the model's ability to generalize is preserved. Therefore, an important point to consider is the relation to overtraining [75]. If a model is overtrained and fits the training data too well, it becomes a practically useless model for its original task, as its ability to generalize and thus its predictive power are both compromised. If a provider supplies an algorithm that cannot make predictions well, it might make the malicious activity more suspicious and easier to detect.

3.8 The link between Adversarial Machine Learning and Watermarking techniques

Although model watermarking and fingerprinting techniques do not share the same purpose as data exfiltration attacks, they are closely related due to the information-hiding component. The requirements for a watermark match the requirements of an attack in data exfiltration schemes. In both cases, the desired characteristics of an embedded watermark (as listed in [42, 11], and embedded training data are:

- **Robustness**: The hidden data should be robust against removal attempts.
- **Fidelity**: A model's performance on the original task should not suffer extensively in order for the utility of the model to be preserved (i.e. fidelity of the attacked model to the base model).
- **Secrecy**: The existence of hidden data should be secret.
- **Efficiency**: The process of hiding data into a model should be fast (as to not raise suspicion due to the long embedding time).
- **Capacity**: The data hiding scheme should be able to embed a large amount of information.

The mutual goal is for the process of information hiding to not impair the utility of the underlying models on their original task.

The link between watermarking and machine learning research has been previously drawn [54, 42]. It mostly refers to the transfer of the attack and defense techniques between adversarial machine learning and watermarking. For example, a defense against a watermark detection attack has been applied to detect model stealing attacks [54]. Furthermore, watermarking has been used as a technique for ownership verification of illegal copies of a model (obtained by a model stealing attack) [42]. This can also be seen in the watermarking taxonomy Lederer et al. introduced (shown in Figure 3.1. This thesis also draws a parallel between watermarking and data exfiltration attacks, however, the roles of the defender and the attacker are reversed. Possible attacks against watermarking (e.g. watermark removal) can be applied as *defenses* against data exfiltration attacks, and are more closely introduced in the following section.

3.9 Mitigation strategies for data exfiltration attacks

Mitigation strategies specifically against data exfiltration attacks mentioned in the underlying study [63] suggest detecting these attacks by code inspection or performing anomaly detection to reveal an unexpected parameter distribution. Detecting these attacks is, however, not trivial. In order to defend against such attacks, the literature suggests modifying the model parameters, as they contain sensitive information once an attack was carried out [63].

Since these attacks are closely related to (or a subfield of) steganography, methods that aim to remove hidden information from the covert channels are relevant to discuss. Jung et al. proposed a method that aims to neutralize malicious code hidden in image files [36]. The method is composed of image file extraction, image file format analysis, image file conversion, and the convergence of image file management modules. Such approaches outline techniques for the removal of hidden information in media files. However, when the hidden information is malicious code, such defenses applied in the DEA scenario focus rather on the removal of the malicious code that is camouflaged among the provided benign code. This thesis however focuses on defenses actively removing hidden data from the trained machine learning models.

As mentioned in the section above, due to watermarking and fingerprinting machine learning models being closely related to the workings of data exfiltration techniques using machine learning models, relevant defenses can be inspired by methods proposed in that field. As mentioned above, the roles of the adversary and the defender are reversed, therefore attacks on watermarking are explored as defense techniques against data exfiltration. In order to remove the hidden data, model transformation is performed. Relevant literature lists several methods for removal of a watermark [11, 56, 74, 42], by performing:

- *Model compression*: Compressing a model can be performed in several ways. It involves removing (potentially redundant, or unimportant) parts of the model. It can be implemented by pruning the model parameters or neurons. A further model compression strategy is quantization. It works by reducing the number of bits representing a model parameter.

- *Pruning*: This technique has been implemented to remove the connections between neurons in a network with the aim of removing hidden information (or simply reducing a too complex model) [78, 48, 26]. Hu et al. proposed removing neurons based on analyzing their outputs in order to compress a network [31]. The neurons with an activation value of zero are then removed. In parameter pruning (value-based pruning) as a watermark (or a fingerprint) removal attack, parameters containing hidden information can be removed. Additionally, together with neuron pruning (activation-based pruning), the model's behavior is altered, removing the hidden (learned) information.
- *Quantization*: This method reduces the number of bits needed to represent model parameters. The research in this area mostly focuses on energy-efficient neural networks, and speeding up the training and inference times [24, 43, 29]. However, in the case a model overfits on a watermark (or other hidden information, i.e. trigger instances), reducing the precision of parameters may potentially reduce the overfitting on these samples. Should information be hidden in the least significant bits of the parameters, this technique might eliminate it.

- *Fine-tuning*: Fine-tuning offers a way to refine models. A trained model is updated by training it for further iterations on new data. It is a common ML practice, for example in transfer learning, which has also been implemented as a method to remove hidden information from a model. The additional iterations modify the parameters, or even the prediction behavior on learned samples [72, 15].

- *Fine-pruning* Fine-pruning combines the two above-mentioned techniques to modify the model. In the first step, the model is pruned. The next step is to add training iterations to account for the model performance on the underlying task lost during pruning. This technique was implemented to remove backdoors [44], as well as watermarking based on backdooring algorithms [35], as the model's behavior is modified. These techniques modify the parameters as well, potentially removing hidden information they contain.

- *Distillation*: This approach enables the transfer of the learned knowledge from a complex model to a simpler model, preserving the prediction behavior on the original task [72, 74], while potentially removing hidden information. Therefore, this approach could be applied to remove the information hidden in a model in a white-box scenario, as it results in the decrease of the number of the parameters, as well as modification of their values. Further, for the black box attack, transferring the prediction behavior to a simpler model using benign data could lead to the removal of the learned patterns from the adversarial data. However, as this is a more complex process, although it could technically be applied as a defense, it does not necessarily align with the defined threat models.

CHAPTER 4

Threat Models

4.1 Threat Models: Common Characteristics

The threat model is based on the underlying study by Song et al. [63] with adaptations according to [10]. The threat model provides a profile of an adversary and of a defender. It describes their goals and capabilities in detail, thereby laying a foundation for the design of the attacks and defenses.

As introduced in Chapter 1, the setting relevant to data exfiltration attacks is a collaboration between two or more parties in building machine learning models where code-sharing takes place. The party providing the machine learning pipeline delivers benign code which includes functionalities to preprocess data and build a machine learning model upon it. However, in case the code provider has malicious intent, a data exfiltration functionality is embedded into the code. Therefore, we refer to the code provider as an adversary (or an attacker). Additionally, in certain settings, such as federated learning, the adversary is not necessarily the author of the ML pipeline - they could however potentially inject malicious code into the shared code. The data owner (also referred to as a client or defender) seeks to leverage machine learning models on their data. In order to do so, they outsource ML model creation or use a third-party library with such functionalities. This way, the code provider enables the data holder to make use of ML without having direct access to the data but exploits the setting by using the model to exfiltrate the data.

4.1.1 Incentives & Goals

Data upon which machine learning models are constructed is often highly valuable, which leads the data holders to maintain its confidentiality. The value of data may be high due to various reasons, e.g.:

- *Cost*: Data poses a valuable asset, also due to the cost associated with its collection. Both the infrastructure and the labor necessary to collect and label data in many cases require a substantial investment. Apart from the financial aspect of data collection, it requires a significant time investment, which further amplifies the value of data.

- *Privacy*: Information captured within a dataset may encompass sensitive or private attributes. This is particularly common in areas such as healthcare, insurance, or banking, where datasets might contain personally identifiable information. Examples of such data include full names, social security numbers, or health records. Furthermore, data from other sectors can also contain private information, such as academic records or purchase history. Therefore, confidentiality is of high importance here.

Adversary's Incentive: The motivation of adversaries to obtain confidential data is often driven by potential monetary or strategic gains or improvement of reputation.

Defender's Incentive: As a defender, prioritizing data confidentiality is crucial given the potential ramifications associated with a data breach. Such consequences range from legal repercussions due to privacy infringement, to financial losses or competitive disadvantages in the market by eroding the defender's competitive standing in the industry.

Adversary's Goal: An adversary aims to exfiltrate valuable training data, which constitutes a security violation. The adversary's objective precisely is to cause a confidentiality violation. A side effect is an integrity violation, as the desired outcome is for the attack to be successful without disrupting the normal system operation, i.e. model utility. There are five aspects of the attack an adversary must consider in order to conduct the violation unnoticed, making their breach successful without attracting attention to the attack, thus avoiding detection:

- *Attack Effectiveness*: The attacker aims for the exfiltrated data to be (nearly) identical to the original data.

- *Model Effectiveness*: The adversary's intent is to not compromise the performance of the model on the intended task. Low performance of the model raises potential suspicions, thus high fidelity of the attacked model to a benign model is desired.

- *Robustness*: The hidden data should be robust against removal attempts.

- *Efficiency*: An attack that requires a high amount of computing resources could significantly increase the duration of the operation, raising the chances of detection. Although training machine learning models can be a time-intensive process, excessive additional computing time could raise alarms and potentially lead to the unveiling of illicit activities. Thus, the attack should ideally not require disproportionate computational resources.

- *Secrecy*: The existence of the hidden data should not attract attention. In order to successfully avoid detection through code inspection, the adversary must ensure the injected malicious code is indistinguishable from benign code, which can be achieved by various code-hiding techniques (introduced in Section 3.3).

Defender's Goal : The goals of the defender are

- *Confidentiality*: Defender aims to maintain data confidentiality, thus ensuring its non-disclosure.
- *Defense Effectiveness*: If the threat to confidentiality is to be reduced by employing mitigation strategies or defense techniques, the goal is to design them in a way that results in high defense effectiveness - meaning potential hidden data is removed from the model.
- *Model Effectiveness* It is in the defender's interest for the model's performance to be high on the intended task. Therefore, when designing a defense strategy, an important aspect to consider is to avoid a sharp decrease in model utility.

Computing Environment

As the steps in an ML pipeline consist of various operations on the training data in order to build a model, the code and the respective ML pipeline provided by the adversary require unrestricted access to the training data and the model, during the training process. Once an adversary's code is applied to the data, instead of performing an exfiltration attack, the malicious code could potentially utilize a network connection, for example uploading the data to an external location. To prevent this, the defender is assumed to run the provided code in a secure computing environment that disallows outbound network connections, as shown in Figure 1.1. Such isolated environment restricts unauthorized data flow. The adversary is assumed to not be able to monitor the execution of the provided code by the client, or the defender's (client's) data during this execution, i.e. no communication between the adversary and the process takes place at execution. Furthermore, this isolated environment could potentially implement time constraints on code execution.

4.2 Threat Model: White-box scenario

This section describes the capabilities of an attacker in a white-box scenario. The adversary needs to have machine learning expertise to build a functional model upon the data. In order to apply this expertise to the underlying data, specific knowledge and resources are required. Moreover, assumptions about the actions the adversary's code can perform are made.

- **Adversary's Knowledge**: In order to build a machine learning model on the underlying data, the code provider (adversary) needs to be provided with data description and structure, at minimum the number and the meaning of the attributes and types of data they contain (e.g. that 'age' is an attribute containing integer values, or that 'occupation' is a categorical attribute containing 14 categories of string values). This requirement is not specific to a code provider with malicious intent but is needed in benign scenarios as well. It allows for appropriate data preprocessing (such as handling missing values, encoding, scaling, etc.). Furthermore, in order to define suitable neural network architecture with an appropriate cost function, the characteristics of the target attribute need to be known. The attacker must also have the knowledge that the final model will be published in a white-box access setting, providing the attacker with access to the model parameters. Otherwise, exfiltration is impossible.
- **Defender's Knowledge**: The defender owns the data and has therefore full knowledge of the contents of the dataset they want a model built on. The defender has access to the code provided by the adversary but can be assumed to have limited abilities regarding understanding of the code content or be unable to carry out code inspection (due to the volume of the code, or lack of resources associated with inspection).
- **Adversary's Actions**:
  - Controlling the training process.
  - Injecting malicious code into the benign code provided to build a model.
  - Hiding the injected malicious code in order to avoid detection by code inspection.
  - Transforming the training set in order to modify the model parameters (within the code, on site of the code execution).
  - Measuring the change in model performance after performing the attack as compared to a benign setting (within the code, on site of the code execution)
  - Accessing the model parameters once published by the client in order to reconstruct the secret.
- **Defender's Actions**:
  - Accepting or rejecting the final model.
  - Quantifying model utility.
  - Employing mitigation strategies and defenses.
  - Publishing the final model (allowing white-box access to the model parameters).

The model of the white-box scenario attack and defense is summarized in Figure 4.1. It shows the access and the actions of the attacker and the defender. It also shows additional tracking done within the experimental evaluation of this thesis.

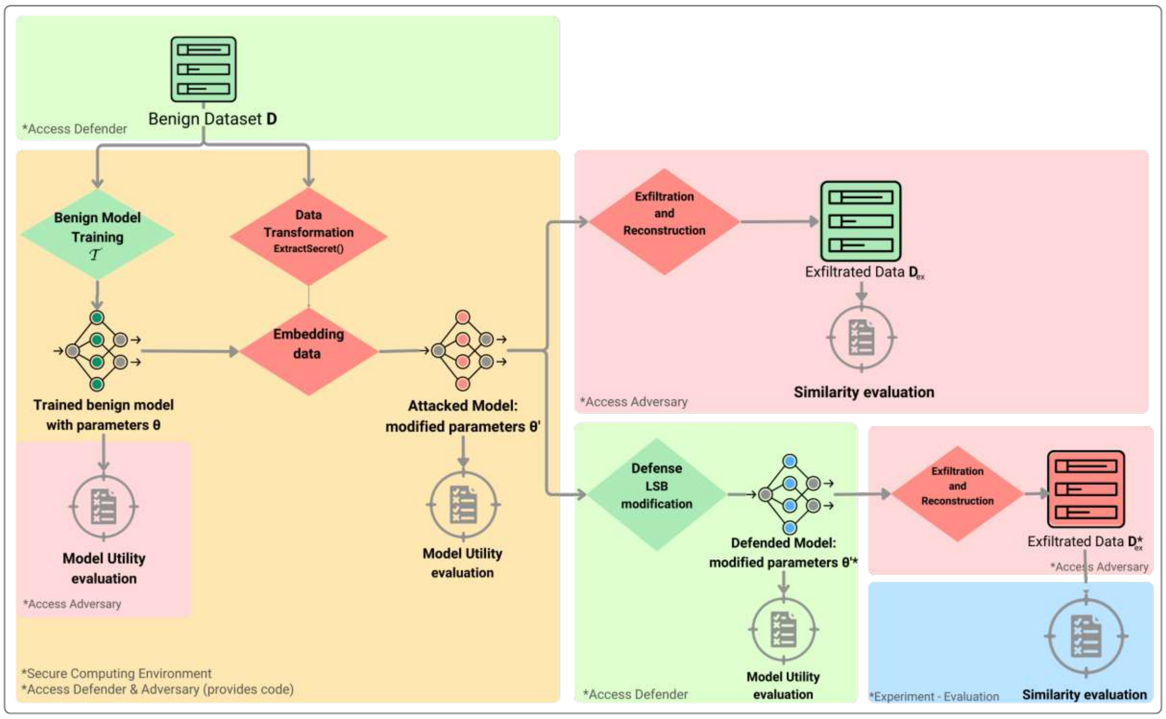

Benign Dataset D
*Access Defender
Benign Model Training
Data Transformation ExtractSecret()
Embedding data
Trained benign model with parameters θ
Attacked Model: modified parameters θ'
Model Utility evaluation
*Access Adversary
Model Utility evaluation
*Secure Computing Environment
*Access Defender & Adversary (provides code)
Exfiltration and Reconstruction
Exfiltrated Data D_{ex}
Similarity evaluation
*Access Adversary
Defense LSB modification
Defended Model: modified parameters θ'*
Model Utility evaluation
*Access Defender
Exfiltration and Reconstruction
Exfiltrated Data D^*_{ex}
*Access Adversary
*Experiment - Evaluation
Similarity evaluation

Figure 4.1: Overview of the design of the Least Significant Bit Encoding Attack & Defense. The green background denotes the access and actions of the defender, the red background shows the access and actions of the adversary. The yellow background depicts the isolated computing environment on the defender's side, where the adversary's code is executed. The blue part denotes the evaluation that can be performed neither by the adversary, nor the defender, however, it is performed within the experiments in this work.

4.3 Threat Model: Black-box scenario

- **Adversary's Knowledge**: Similarly to a white-box attack, the code provider (adversary) needs to be given a data description and information regarding the structure of the data. In order to provide a suitable ML pipeline, the attacker is assumed to be provided with the same information as in the white-box attack, i.e. the number and meaning of attributes and types of data the respective attributes contain. Additional knowledge of data distributions allows the adversary to perform different modes of the trigger set generation and is therefore assumed for a subset of the subsequent experiments. Furthermore, only the adversary has the knowledge to construct the trigger dataset used for exfiltration. Contrary to the white-box scenario, the attacker does not need to know beforehand whether the model will be published with white-box or black-box access. Black-box access is assumed, although the adversary can abuse the model to exfiltrate in both settings, as long as access to the predictive functionality is provided.

- **Defender's Knowledge**: Similarly, as in the white-box scenario, the defender owns the data and has therefore full knowledge of the training dataset. Despite access to the code, precise knowledge of its contents is not assumed.
- **Adversary's Actions**:
  - The same actions as in the white box setting, i.e.
    - ∗ Manipulating model creation by influencing the learning process through the inclusion of malicious code within the benign code provided to build a model.
    - ∗ Hiding malicious code in order to avoid detection by code inspection.
    - ∗ Monitoring the model performance and similarity of potentially exfiltrated data during the training process (within the code, on site of the code execution).
  - Generating trigger dataset (the generation is deterministic to allow for generation after gaining access).
  - Transforming the training set into labels of the trigger samples.
  - Accessing the predictive functionality of the model (a prediction API) and submitting queries once the model is published.
  - Reconstructing the data from the predictions on the generated trigger samples.
- **Defender's Actions**:
  - The same actions as in the white box setting, i.e.
    - ∗ Accepting or rejecting the final model.
    - ∗ Quantifying model utility.
    - ∗ Employing mitigation strategies and defenses.
    - ∗ Publishing the final model (allowing the use of the predictive functionality of the model).

The model of the black-box scenario attack and defense, depicting the access and actions of the attacker and the defender, is summarized in Figure 4.1.

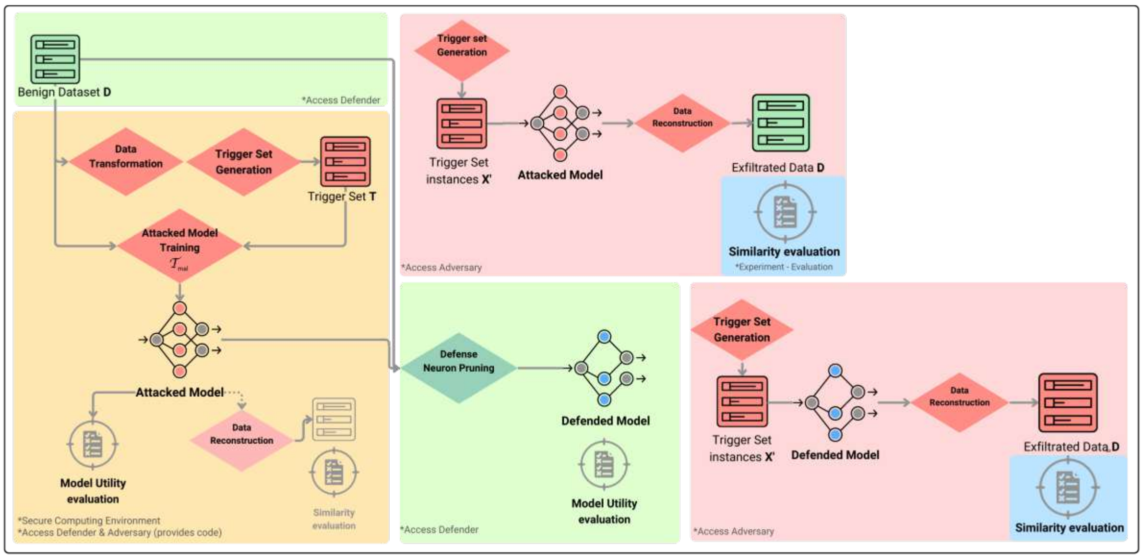


Figure 4.2: Overview of the design of the Exfiltration Attack & Defense in the black-Box scenario. The green color denotes the access and actions of the defender (client). The red parts of the figure depict the access and the actions of the adversary. The yellow part of the figure shows the isolated computing environment on the client's side, in which the adversary's code is executed.

CHAPTER 5

Experiment Design

This section provides a detailed description of the design of the experiments carried out and the evaluation of their results. The experiment design includes the data description and exploration, as well as a comprehensive overview of the required data preprocessing. The design of all the variations of the attack methods and their respective defenses are thoroughly explained, including the description of the architecture of the employed machine learning models. Subsequently, the evaluation process of the effectiveness of both the attacks and defense techniques, as well as the model utility are described. The principles of the CRISP-DM framework [70] are followed for the design and evaluation of the attacks and the corresponding defenses. Detailing these processes aims to ensure the transparency and reproducibility of the experiments and their results.

5.1 Dataset

The dataset used in this thesis is the 'Adult Income Dataset' from the UCI Machine Learning Repository[1]. It was chosen in order to perform the data exfiltration attack and defense experiments on it, as this type of dataset can be classified as valuable, and in this case also sensitive, since it contains personal information such as relationship status or income level of individuals. Furthermore, it is a tabular dataset, so it fulfills the condition of investigating the attack behavior on this type of data.

5.1.1 Data Exploration

The dataset consists of two parts stored in separate files: a training dataset and a separate testing set.

The leftmost part of Figure 5.1 shows the two data files - training and test set. Altogether, they contain 48,842 samples - 32,561 in the training dataset, and 16,281 in the testing

[1]https://archive.ics.uci.edu/dataset/2/adult

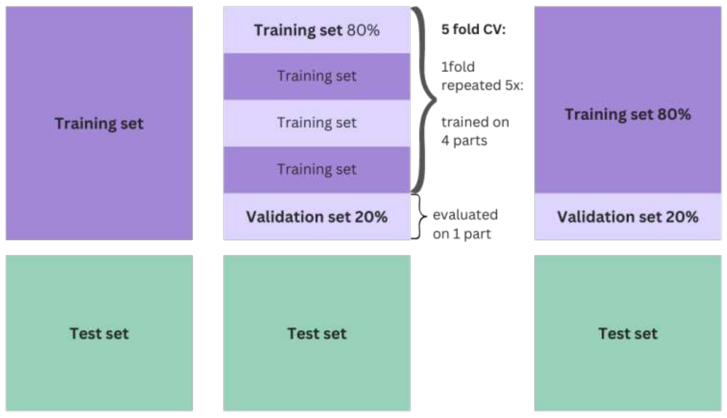


Figure 5.1: Original dataset and the splits used for Grid-Search and subsequent experiments: leftmost column shows the original data - a training set and an independent test set

Table 5.1: Counts of unique values per feature in the training dataset

Column name	Type	Unique Values
age	Continuous	73
workclass	Categorical	8
fnlwgt	Continuous	21,648
education	Categorical	16
education_num	Categorical	16
marital_status	Categorical	7
occupation	Categorical	14
relationship	Categorical	6
race	Categorical	5
sex	Categorical	2
capital_gain	Continuous	119
capital_loss	Continuous	92
hours_per_week	Continuous	94
native_country	Categorical	41
income	Categorical	2

set. There are 14 features, out of which eight are categorical and six numerical, and one target attribute, which is binary. The task is to predict the income level of an individual - whether their income is more than \$50,000 or less than or equal to \$50,000.

The features included in the dataset, along with the type of data they contain and the number of unique values per feature, are listed in Table 5.1.

Figure 5.2 shows the distributions of the values in each column of the dataset. From this

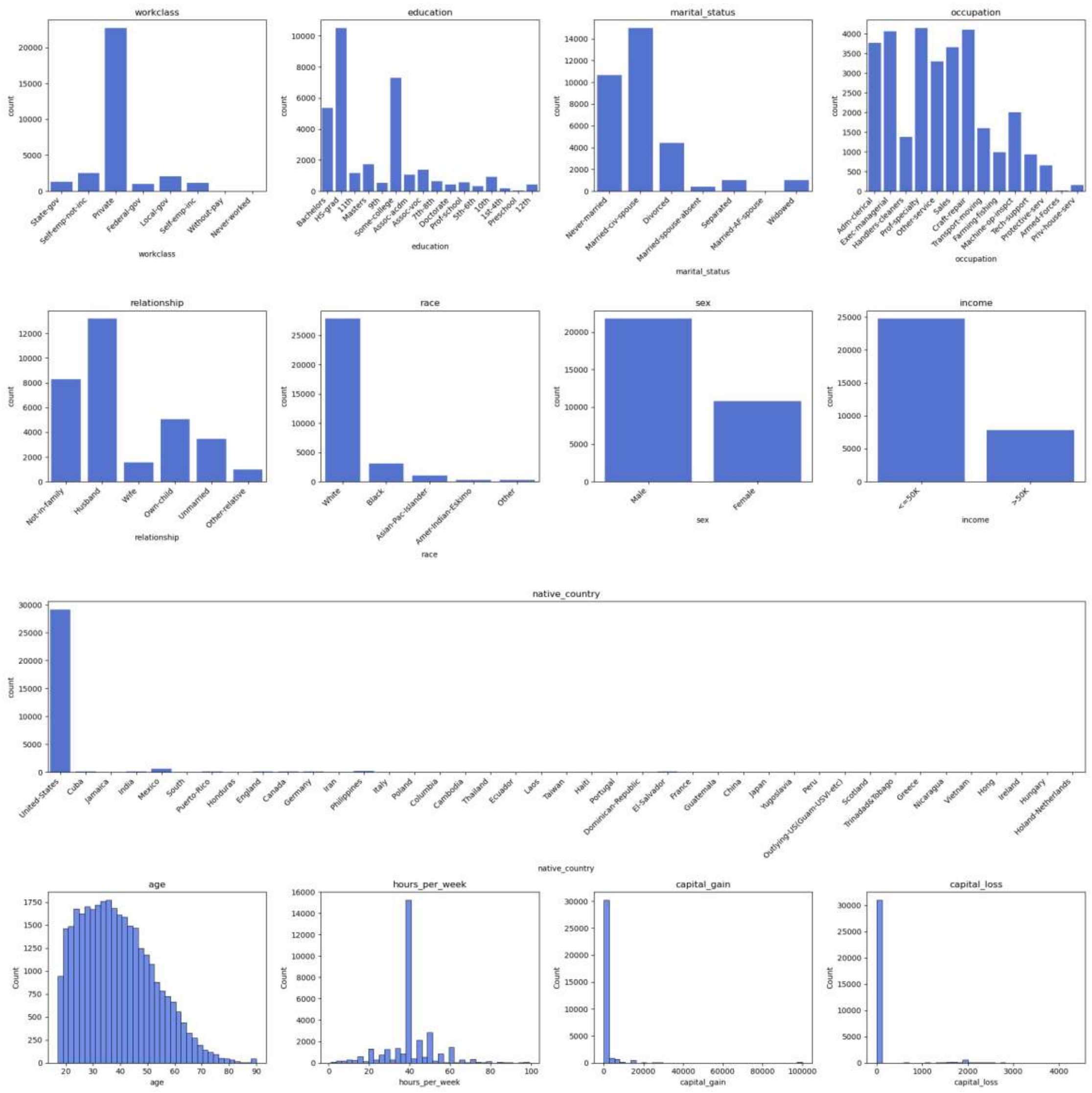


Figure 5.2: Distributions of the features in the training dataset: Top three rows represent the categorical features, bottom row shows the histograms of numerical features

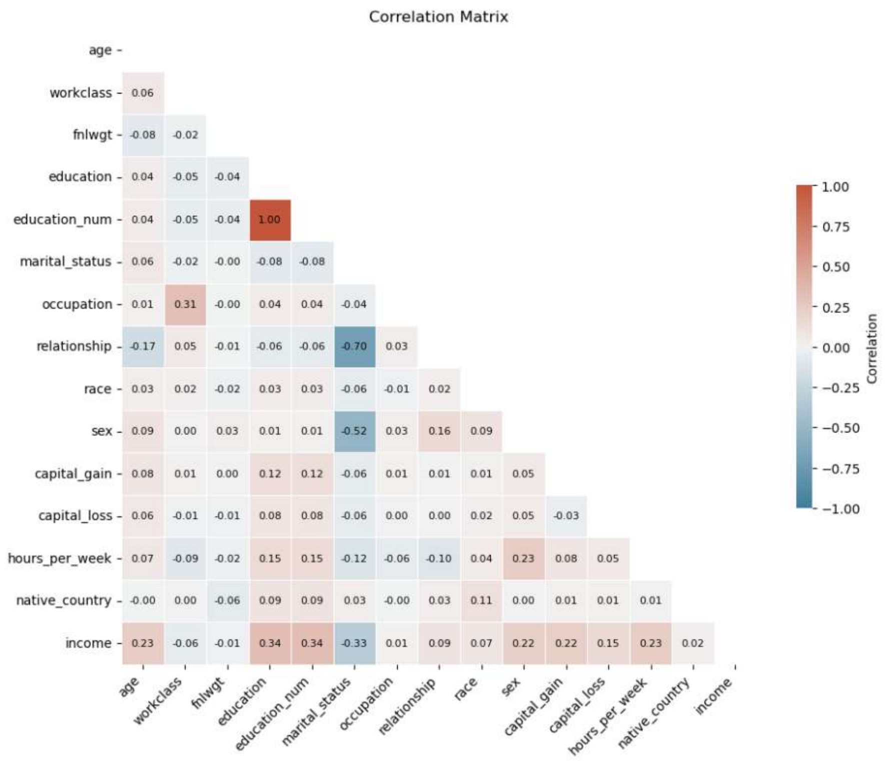


Figure 5.3: Feature Correlation matrix depicting the pairwise correlation values between the attributes in the training dataset

plot, we can observe that most people included in the dataset are employed in the private sector, and most work 40 hours a week. The most common education level is a high school graduation. The most common marital status is married, followed by people who have never been married. The capital gain and capital loss attributes are very left skewed, with most people in the dataset having either a low capital gain or a low capital loss. The attribute 'native_country', although not shown in this figure, is also not balanced, with 85.59% of the people listing the United States of America as their native country, while the rest of the samples contains 40 other countries.

The target attribute is imbalanced, with over 75% of the samples with an income of less than $50,000. More specifically, the training set holds 75.92% samples belonging to the '<=50K' income category, and 24.08% to the '>50K' category. The test set has a comparable proportion of the two categories.

Figure 5.3 depicts the correlation between the features in the dataset calculated using the person correlation coefficient. We can observe that the features 'education_num' and 'education' are perfectly correlated. This is because the 'education_num' is a label-encoded version of the 'education' attribute, representing the levels of 'education' ordered from the lowest to the highest level. The features 'relationship' and 'marital_status' are also showing a positive relationship. This is also expected, as they contain some redundant information. We can also observe that 'education_num' has the highest positive correlation to the target attribute 'income' among the other features. On the other hand, 'marital_status' attribute has the highest negative correlation to the target attribute.

5.1.2 Data Preprocessing

This section describes the necessary preprocessing steps taken before the data is used for building a classification model. As the training and testing datasets come as two separate files, no splitting was necessary for this reason. Figure 5.1 shows the splits of the data used for hyperparameter tuning; a 5-fold cross-validation is performed (the middle part of the figure), and further elaborated more closely in Section 5.1.3. Figure 5.1 further depicts the split for the subsequent attacks and defense experiments. For the black-box attack, a validation set of the size of 20% of the training data is used by the attacker for performance tracking, shown as the rightmost part of Figure 5.1, and further elaborated in Section 5.4.1. The preprocessing is done on each set separately to avoid potential data leakage and the subsequent overconfident evaluation of model performance [61, 73].

Features Due to the perfect correlation of the 'education' and 'education_num' features, the feature 'education' is removed from the dataset. The feature 'fnlwgt' has no predictive power and is therefore removed as well. The feature 'relationship' is removed as well due to its redundancy. To further reduce the feature space, 'capital_loss' is subtracted from 'capital_gain' to create a new feature named 'capital_change'. The variable 'native_country' is transformed into a binary attribute. As seen in the Figure 5.2, the majority of the samples have the value 'United-States' in this column. The rest is scattered over 40 different values. Therefore this attribute has been transformed into a binary attribute containing the values 'US' and 'Non-US'.

Missing Values A few of the features contain missing values, namely the feature 'workclass' contains 1,836 missing values, 'occupation' 1,843, and 'native country' 583 missing values in the training set. In order to handle the missing values, k-Nearest-Neighbors imputation is performed with $k = 3$. We show that this imputation leads to results comparable to the state-of-the-art, as our best final performance on the test set with an accuracy of 85.11% (reached by a model with 1 hidden layer with the hidden layer size of 164 neurons). The results obtained on the test set is shown in order to compare with a previous study that used this dataset. Our result is comparable to a result obtained by J.Poulos and R.Valle [53], where an accuracy of 85.6% was reached on the test set. The network architecture these authors have used was, however, larger, and consisted (besides an input and an output layers) of 2 hidden layers, with a hidden layer size of 1,024 neurons. However, it is important to mention that the test set serves strictly for evaluation only, and not for model selection – for the subsequent experiments a hyperparameter search using cross-validation is done, performing model selection based

Table 5.2: Explored Hyperparameter Grid

Hyperparameter	Layer size (m)	# hidden layers (l)	Batch size	Learning rate (α)	Dropout	Optimizer
Values	1 (*input size)	1	128	0.1	0	adam
	2 (*input size)	2	256	0.01	0.1	sgd
	3 (*input size)	3	512	0.001	-	-
	4 (*input size)	4	-	-	-	-
	5 (*input size)	5	-	-	-	-
	10 (*input size)	-	-	-	-	-

on the scores achieved on the validation set instead - the test set is treated as separate, unseen data.

Encoding & Scaling In the following step, one-hot encoding is applied to the categorical data. The numerical features are standardized.

5.1.3 Hyperparameter Tuning

The hyperparameter tuning is not an explicit priority for the subsequent experiments, as the aim of experiments was not to optimize the performance of a benign model, but to observe the change in the performance of the different models with the attack and defense functionalities and to allow for comparisons among various settings of the experiments. Nevertheless, the tuning is performed in order to find hyperparameters that lead to acceptable performance of the base models which are later attacked (and defended), and to provide an evaluation in a realistic and fair setting: our model is not chosen in a way to demonstrate specific aspects of the data exfiltration attack and defense capabilities, disregarding the actual machine learning task, but we demonstrate the attacks and defenses on a model that is likely to also be chosen in a real-world setting for the actual predictive task.

In order to find the hyperparameter combination that results in optimal classification performance, hyperparameter tuning using grid-search is carried out. It is performed using 5-fold cross-validation by training the model with different combinations of hyperparameter values and evaluating its performance. Altogether, in the final run of the grid-search, 1,080 different models were tested. The explored hyperparameters consisted of the number of hidden layers in the MLP, layer size (in multiples of input size), learning rate, batch size, dropout, and optimizer. The values for each of the explored hyperparameters can be seen in Table 5.2.

Figure 5.4 shows the runs executed in the hyperparameter search. Each line represents one model with a different combination of hyperparameters. Each axis shows a different hyperparameter, with the values along the respective axis showing the explored values. The rightmost axis shows the final results of the 5-fold cross-validation, with the color indicating the value of the arithmetic mean of the accuracy score across all five folds. The differences due to batch size and dropout in the outcome are not visible from this

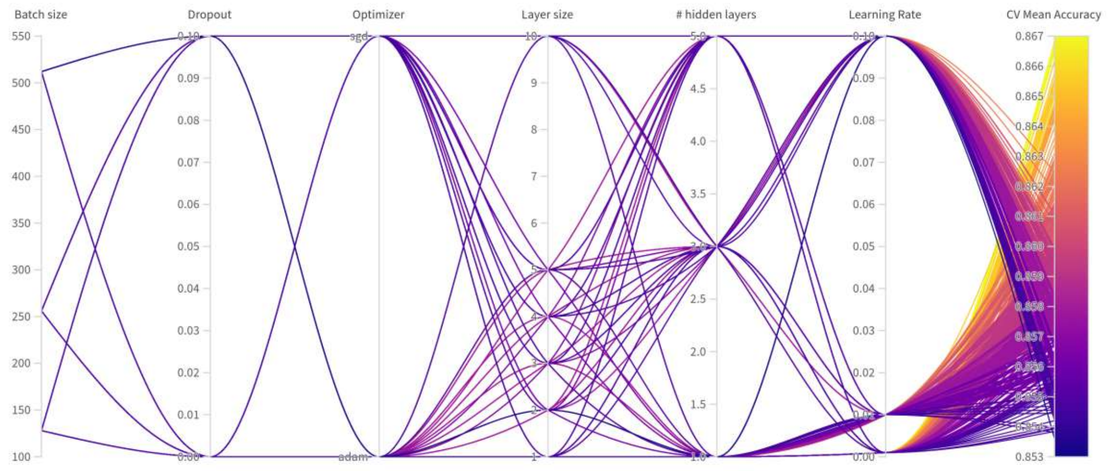
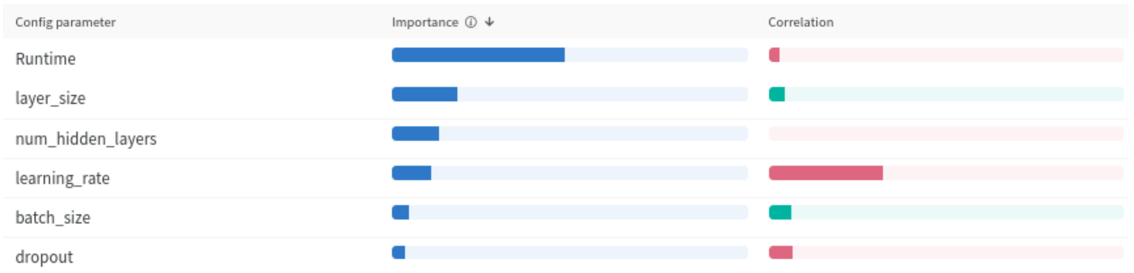


Figure 5.4: Results of model hyperparameter search to determine optimal settings for subsequent attacks. Each line represents a model with a different combination of hyperparameters in the grid search. The axes represent the explored value of hyperparameters, with the right-most axis depicting the results of these models with respect to the mean validation accuracy in 5-fold cross-validation



Config parameter	Importance ⓘ ↓	Correlation
Runtime		
layer_size		
num_hidden_layers		
learning_rate		
batch_size		
dropout		

Figure 5.5: Hyperparameter importance and their correlation with respect to the mean validation accuracy in 5-fold cross-validation

plot, but with the optimizer, we can start to see that the Adam optimizer leads to higher scores than stochastic gradient descent. The layer sizes of 1 and 10 show lower scores than the values in between. The number of hidden layers does not seem to have a large impact on the final performance. We can observe that the lower learning rate values lead to higher scores.

The importance of each of the hyperparameters (of numerical values), along with runtime, and their correlation to the arithmetic mean accuracy resulting from 5-fold cross-validation can be observed in Figure 5.5. The correlations indicate the nature and significance of linear relationships between individual hyperparameters and the model effectiveness (mean accuracy). The resulting value lies in the range $[-1; 1]$. Although these correlations demonstrate evidence of a relationship between a hyperparameter and the mean

Table 5.3: Hyperparameters of models used in attack and defense settings

Dropout	**Batch size**	**Learning rate α**	**Optimizer**
0	512	0.001	Adam

accuracy they do not necessarily imply causality. These correlations are susceptible to outliers, which may transform an otherwise strong relationship into a moderate one. Moreover, correlations can only grasp linear relationships between hyperparameters and the calculated metric. More complex relationships would not be identified by correlation measures. The importance signifies how useful each hyperparameter was with respect to the mean accuracy (obtained through 5-fold cross-validation). This metric is provided by the WandB experiment tracking tool [8]. The importance measures are calculated by employing a tree-based model.

Therefore, we can observe, that layer size has been identified as the hyperparameter with the highest importance, and is slightly positively correlated with the mean accuracy. The number of hidden layers has a slightly lower importance, in spite of this relationship not being of linear nature. The value of the learning rate α is negatively correlated to the resulting model accuracy: the higher the value of the learning rate (as shown in Table 5.2, the lower the mean accuracy, and vice-versa. This aligns with the observation from Figure 5.4, where we can observe the correlation of the hyperparameters to the model performance obtained by cross-validation, and we can see that the lower values of the learning rate, as well as the Adam optimizer, yield higher accuracy score (depicted by lighter color).

Batch size and dropout are showing lower importance, with batch size having a positive relationship to the mean accuracy, whereas dropout is negatively correlated with this metric. The best-performing model resulting from the 5-fold cross-validation is obtained with the following hyperparameters: 1 hidden layer of size 3 (123 neurons), a dropout of 0.0 (i.e. no dropout), a learning rate of 0.001, a batch size of 512, and *Adam* optimizer. The best-performing model hyperparameter values used for the subsequent experiments are shown in Table 5.3. The achieved accuracy on the test set is 84.46%. The zero-rule baseline accuracy on the test set is 75.43%. When retrained on the full training set, the model achieves 84.68% accuracy.

5.2 Design of Attacks & Defenses: Common settings and Evaluation

The model hyperparameters are kept constant across all experiments, in order to ensure comparability and observe the impact on attack and defense success and model utility based on the various settings of the attacks and defenses. Otherwise, a change in model effectiveness could be caused either by a change in model hyperparameters, or the change in attack and/or defense settings. Furthermore, the experiments are performed on

Table 5.4: Attack & Defense behavior investigated on models of following sizes

# of hidden layers	**1**			**3**			**5**		
Layer size (×input size)	**1**	**3**	**5**	**1**	**3**	**5**	**1**	**3**	**5**
# of neurons in per hidden layer		41			123			205	
# of parameters in a model	1,764	5,290	8,816	5,208	35,794	93,276	8,652	66,298	177,736

models where the model size varies (depending on layer size and the number of hidden layers). This, along with the number of parameters, in each model can be seen in Table 5.4. It results in nine models of different sizes. The layer size refers to the number of neurons in the hidden layer **h** and is expressed as a multiple of the input size. The input size is equal to the number of features in the preprocessed data m, excluding the target feature, i.e. $m - 1$, the number of neurons per hidden layer is, therefore, $layer\_size \times (m - 1)$. In the following chapters, the various model sizes are therefore referred to as **Number of hidden layers × layer size**, e.g. 3×3.

Figure 5.6 shows the overview of all experiments and the different settings explored. The explored model sizes are depicted in this figure. Further attack settings are explained and elaborated in the sections corresponding to the attack-specific design - white-box in Section 5.3 and black-box in Section 5.4.

5.2.1 Evaluation

This section details the evaluation process of the attacks and the corresponding defenses.

Attack & Defense utility are both measured by comparing the exfiltrated data $\mathbf{D}_{ex}$ to the original data $\mathbf{D}$. The only difference is that a successful attack leads to a high similarity, whereas a successful defense leads to a low similarity. Therefore, a metric that calculates this similarity is implemented. Since the experiments are done on tabular data, the metrics implemented by [63] are not applicable, since they focus on other data types (such as Mean Absolute Pixel Error - MAPE for image similarity). When working with tabular data, different types of attributes typically need to be considered. As shown in Table 5.1, the dataset consists of attributes of continuous numerical values, an ordinal attribute and categorical attributes. The similarity metric, therefore, needs to encompass this information. Thus, for each pair of instances (from normalized datasets $\mathbf{D}_{ex}$ and $\mathbf{D}$), for each pair of values belonging to

- **Categorical attributes**: Hamming distance is calculated.
- **Numerical attributes**: Euclidean distance (absolute difference) is calculated.

This way, the distance between the two samples, i.e. the error of exfiltration, is calculated. The resulting values are added per instance, over all instances, and averaged over the length of the dataset. Subsequently, the error is subtracted from 1 to represent data

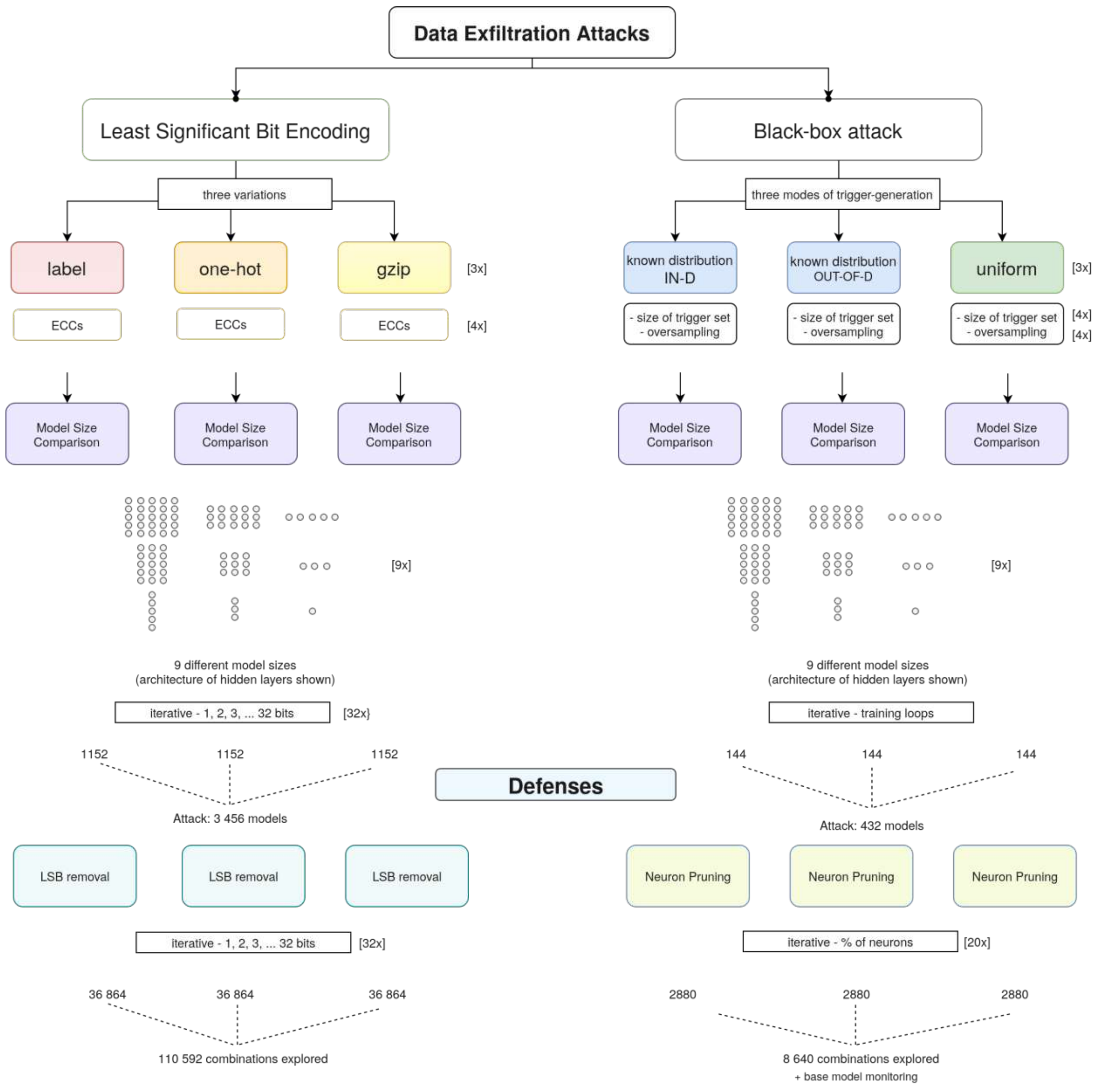

Data Exfiltration Attacks
Least Significant Bit Encoding
Black-box attack
three variations
three modes of trigger-generation
label
one-hot
gzip
[3x]
ECCs
ECCs
ECCs
[4x]
known distribution IN-D
known distribution OUT-OF-D
uniform
[3x]
- size of trigger set - oversampling
[4x]
[4x]
Model Size Comparison
[9x]
9 different model sizes (architecture of hidden layers shown)
iterative - 1, 2, 3, ... 32 bits
[32x]
iterative - training loops
1152
1152
1152
144
144
144
Defenses
Attack: 3 456 models
Attack: 432 models
LSB removal
Neuron Pruning
iterative - 1, 2, 3, ... 32 bits
[32x]
iterative - % of neurons
[20x]
36 864
36 864
36 864
2880
2880
2880
110 592 combinations explored
8 640 combinations explored
+ base model monitoring

Figure 5.6: Overview of the experiments conducted within this work, showing the variants of the attacks and the comparisons made. The bottom part depicts the defense approaches. The overall numbers of experiments conducted is shown.

similarity. The final value of the metric can be expressed in terms of percentages, with 100% indicating that the $\mathbf{D}_{ex}$ and $\mathbf{D}$ are identical.

This similarity metric can be generally applied to samples of tabular data, however, it does not take the ranges of the data into account, which means that even if nonsensical data is exfiltrated, a certain level of similarity will be assessed by this metric. To force the similarity to be 0, the (expected) ranges of values in each column could potentially be taken into account (e.g. if an exfiltrated value for the column 'age' is ≥ 100 similarity is set to 0). This would entail a dataset-specific crafting of a similarity metric, which requires specific domain knowledge. The more generic metric described above is therefore applied within this work, as it serves to identify the success of attacks and defenses.

Model Utility In order to quantify model utility, the model's performance on the original task needs to be measured. Here, the attacker and the defender both share the same goal - preserving the model utility. To evaluate how well a model performs on the task, the performance measures introduced in Chapter 2 are employed: accuracy, precision, recall, ROC AUC, and F1 score are computed on the training set and a separate test set, as shown in Figure 5.1, for all experiments conducted. For the black-box attack, the adversary uses a validation set (rightmost part of Figure 5.1) to track the performance during training and decide e.g. on early stopping. The final results are however also compared on the separate test set, to ensure a fair comparison across all configurations (i.e. combinations of attack & defense settings)

5.3 Design of White-Box Exfiltration Attack & Defense

The following section presents the details of the attack and defense design within the context of the white-box scenario. The variants of the attack given the employed encoding strategies are detailed, each in terms of their characteristics. The data transformations associated with each variant are described, and the required model parameter transformations are explained. The objective is to provide a comprehensive overview of the attack architecture to allow for the understanding of the attack deployment process, as well as the defense technique.

5.3.1 Least Significant Bit Encoding Attack

In order to carry out the data hiding, the adversary constructs the training pipeline identical to a benign training setting. Once the training is performed, the attacker then embeds the training data into the parameters θ of the final train model based on the model capacity. This is achieved by modifying the weights and biases of the benign model using them as a covert channel - data is encoded into the least significant bits of these numbers.

For the experiments, in the first step, base models (benign models) are retrained on the full training set with the hyperparameters found by grid-search as described in 5.1.3.

5.3.2 Model Capacity

The capacity of the model depends on the number of parameters in the model and the number of bits b used for data hiding. The attacker, therefore, determines the capacity of the model by calculating the number of bits available for hiding by multiplying these two numbers. From the attacker's perspective, a large model is desirable due to the larger number of parameters it offers for data hiding.

5.3.3 Data transformation

The training data that the attacker intends to exfiltrate is transformed into its bit representation. Three variants of the transformation for the subsequent encoding are explored within this thesis.

Gzip transformation

This type of transformation for encoding is based on the implementation from the underlying study by Song et al. [63]. It has been adapted for use with tabular data. Gzip compression is applied to the training data in order to reduce the amount of information to be encoded into the model. The resulting data is transformed into a byte string, which is transformed into its bit representation. The data is now ready to be encoded into the parameters. One aspect to consider with this transformation is the capacity of the model. The size of compressed data in bits needs to be determined beforehand, in order to trim the training data to the desired size, so that once compressed, it fits into the LSBs of the model.

- **Advantages**: This is a fairly simple and for an attacker an easy-to-execute approach. Compressing the training data before it is encoded into the parameters of the model allows for a larger volume of the data to be encoded into the parameters, and subsequently exfiltrated than with other transformation variants.
- **Disadvantages**: As a result of the nature of data compression aiming to utilize space as efficiently as possible, minor errors can lead to a pronounced impact on the success and result of decompression. The susceptibility of a compressed gzip file to small errors and file corruption compromises the robustness of this approach. This is the case also when the attacker is able to reconstruct known building blocks of the file format. If the attacker embeds any identifiable parts of a gzip file into the parameters, it makes the attack more susceptible to detection (e.g. a gzip file header can be read from the LSBs, and the data hiding thereby detected)

Label-Encoding Transformation

For the following transformation, label encoding is applied to the categorical attributes of the training dataset D. The numerical values remain unchanged in this step. The number of attributes (excl. the target variable) in the dataset is represented by m. As the adversary aims to exfiltrate the target attribute as well, we denote the number of attributes in D by $m+1$. Subsequently, each value x_{ij} and the corresponding y_i from the training dataset D is transformed into its bit representation so that each value now has a length of 32 bits.

$$x_{ij} \mapsto \mathbf{s}_{ij} \quad \text{where} \quad i \in \{1, 2, \ldots, n\}, \quad j \in \{1, 2, \ldots, m+1\}, \quad \text{and} \quad \mathbf{s}_{ij} \in \{0, 1\}^{32}$$

The transformed dataset has the following form:

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1m+1} \\ s_{21} & s_{22} & \cdots & s_{2m+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nm+1} \end{bmatrix}$$

Specifically, the 'Adult Income' dataset contains 12 columns after preprocessing. This label-encoding transformation, therefore, results in 384 bits representing one instance of the dataset D, meaning 384 bits are necessary in order to hide one data sample in the model parameters.

- **Advantages**: This method offers a straightforward procedure for data transformation in the context of data exfiltration. The main advantage is that it is more robust to bit-flip errors and thus less susceptible to minor errors compared to the gzip file encoding.
- **Disadvantages**: The amount of bits needed to represent the dataset is significantly higher, thus requiring the capacity of the model to be high as well in order to exfiltrate a meaningful amount of data. When working with a model of the same size, this approach effectively lowers the capacity of the model with respect to the amount of data that can be exfiltrated.

One-hot Encoding Transformation

In order to reduce the number of bits needed to represent one sample, the data is transformed in the following way. Each categorical value is one-hot encoded. One-hot encoding eliminates the need for the values from the categorical attributes to be represented by 32 bits each, as every value from each category from each attribute can be represented by (a combination of) either 0 or 1. One-hot encoding of the categorical features results in 38 variables. This means 38 bits are necessary to represent the categorical values of each sample (as opposed to 256 bits in label-encoding transformation). Each column containing rational or negative numerical values is represented by 32-bit single-precision floating-point values, otherwise, the numerical columns are represented by integer values. The numerical values are transformed into the corresponding bit representation. The integer values, when transformed into their bit representation, require a different number of bits depending on their value, thus resulting in different lengths of the bit strings. In order for the adversary to reconstruct the data once the final model is published, the same length representing each of these values is necessary. To address this issue, first, the length of the longest bit representation max_len of the integer values is calculated, and padding is added to all the other values until max_len is achieved. Second, the adversary needs the knowledge of the max_len variable. As defined in Section 4.1.1, no communication or signaling from the model to the attacker is possible during the code execution. For this reason, the attacker treats max_len as a 32-bit floating-point value and transforms it into its bit representation. This is then encoded into the model parameters together with the training data. Once the max_len is recovered, the training data can be reconstructed.

- **Advantages**: The one-hot encoding transformation combines the general concepts of the two previously introduced techniques. This method offers a reduction in the number of bits needed to represent the data, as compared to label-encoding transformation, and an increase in robustness compared to the gzip file transformation. In the 'Adult Income' dataset, the one-hot encoding transformation results in a 91-bit representation of one data instance. This results in a 76.4% decrease in the number of bits necessary to represent one sample, in comparison to the result of the label-encoding transformation. By reducing the number of bits needed for encoding of one instance, the capacity of each model, with respect to the amount of data that can be encoded, is increased. A further advantage is that this method offers more robustness than the gzip file encoding, with lower susceptibility to minor errors.

- **Disadvantages**: The amount of bits needed to represent the dataset is higher than the gzip file encoding. The susceptibility to errors is higher than the label-encoding transformation.

For each of the transformation settings, in the following experiments, the transformed training data is encoded into the least significant bits of the parameters, with an increasing number of bits b_a utilized, from 1 until 32 bits are reached (when the whole parameter is changed).

5.3.4 Defense: LSB Sanitization

In order to defend from the LSB Encoding attack, the aim is to remove the hidden information from the model. This can be achieved by modifying the parameters of the attacked model θ'. For example, the least significant bits of the parameters can be sanitized. Algorithm 5.1 shows the steps of the chosen defense. Within this work, the value of all LSBs b_d is set to 0. The parameters are therefore modified once again - and the defense returns a model with parameters θ'^*. This way, the hidden information is partially removed. How much information is removed depends on the interplay between the attack bits b_a and the defense bits b_d. However, neither the attacker nor the defender has the knowledge of how many bits the other party is considering. Therefore, the

Algorithm 5.1: LSB defense by sanitizing least significant bits of model parameters

```
Input: trained model parameters $\theta'$, number of bits $b_d$ to sanitize per parameter
Output: Sanitized model parameters $\theta'^*$
for each parameter $p'$ in $\theta'$ do
    $p'^* \leftarrow$ set $b_d$ lower bits of $p'$ to 0s
end
return $\theta'^*$
```

subsequent experiments explore all of the possible combinations (i.e. 32 b_a × 32 b_d).
Figure 5.7 shows the process of parameter modification, along with an example of a

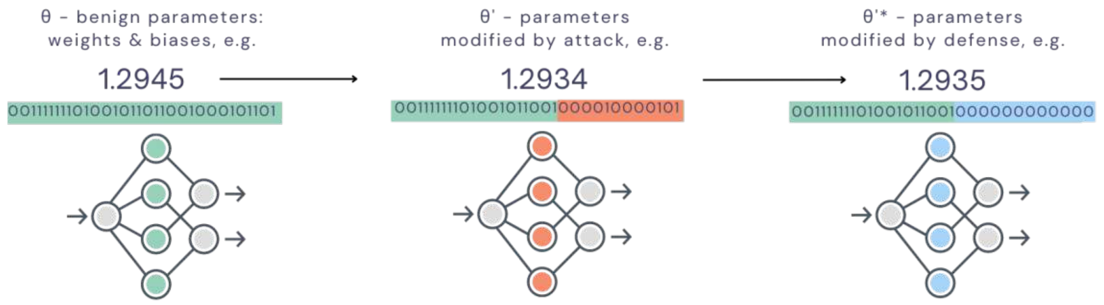


Figure 5.7: LSB attack by data hiding and defense by parameters sanitization: example of parameter value change

parameter value. In this example, $b_a = b_d = 14$, which results in 14 LSBs of a parameter from the benign model being replaced by bits of training data, which modifies the benign parameter value from 1.2945 to 1.2934. The defender sets the values of a b_d number of LSBs to 0. The parameter value is further modified by this defense, which results in 1.2935.

Since neither of the parties knows how many LSBs the other party modifies, this can result in the removal of all of the information if $b_a \leq b_d$. This in turn results in a change in model performance that is equal to the scenario where a base model is modified by the defense (i.e. as if no attack took place, and the defense was applied in a benign setting). The case when $b_a > b_d$ leads to the preservation of some hidden training data (i.e. secret).

5.3.5 Attack Adaptation

When anticipating a detection method or an active defense, the attacker can employ further techniques to reinforce the attack. We describe two options:

- **Encryption**: The adversary has the option to encrypt the data before hiding it in the model parameters. The underlying study [63] uses the AES CBC encryption scheme. It makes the hidden data more difficult to detect, should the defender look for any pattern in the LSBs, or attempt to reconstruct any information from LSBs. However, it reduces the robustness of the attack against defense techniques. If the encrypted data is hidden in the bits of parameters and subsequently changed, the attacker is not able to decrypt the data correctly. The decryption will yield a different plaintext than the original data, so when reconstructed, it will not match the training data and the attack is rendered useless. For this reason, encryption is not applied to the subsequent experiments.

- **Error Correction Codes** In order to increase the robustness of the attack against a potential defense, the adversary can encode the training data using error correction codes. For this purpose, Reed-Solomon encoding is employed. However, there is a trade-off between utilizing the ECCs and the capacity: as the encoding adds additional symbols to the data to detect and correct potential errors, the final amount of bits to be encoded into the model parameters increases. This means that with the same model size, less data can be exfiltrated. The adversary decides on the trade-off they are willing to accept.

Both of these options would be applied to the secret s before the transformed data is encoded into the parameters (as the penultimate step in Algorithm 3.1)

5.4 Design of Black-Box Exfiltration Attack & Defense

This section explains the details of the attack and defense design in the black-box scenario, along with the details of the design of experiments. The variants of the trigger set generation are described and the investigated settings are detailed.

5.4.1 Black-Box Exfiltration Attack

The overview of the attack design is shown in Algorithm 3.3. The trigger set generation is carried out according to Algorithm 3.4. Since a dataset with a binary target attribute is used to demonstrate the attack, one trigger sample is needed to represent one bit of training data. The auxiliary knowledge K, in this case, represents the knowledge of dataset characteristics as described in Section 4.3. For a subset of attacks, it also comprises knowledge of probability distributions. The model is trained on the benign training set and a generated trigger set. In each epoch, first, a full training pass over the benign data $\mathbf{D}$is performed. Subsequently, within the same epoch, a full pass on a combined adversarial dataset is done. This combined dataset contains the benign training data as well as the trigger set samples. i.e. $\mathbf{D} \cup \mathbf{T}$, which aligns with the training strategy from the original paper of Song et al., where this attack was performed on image and text data [63].

The settings explored in the black-box scenario attack are:

- **Knowledge** K: mode of trigger set generation (based on attackers knowledge K of training data probability distributions). It determines how the trigger samples are generated. Three different options are chosen depending on the knowledge of the attacker:
  - The attacker has **knowledge** of training data distribution
    - **In-distribution** data is generated
    - **Out-of-distribution** data is generated
  - The attacker has **no knowledge** of the training data distribution
    - **Uniformly distributed** data is generated
- **Trigger ratio**: ratio of the trigger set size to the benign training set size (in %), indicates the number of trigger instances g to be generated and learned by the model: $g = trig\_ratio \times len(\mathbf{D})$
- **Oversampling**: simple trigger set oversampling by repetition, as chosen by the adversary. Each trigger sample is then repeated multiple times. In this work, we explore four settings: (i) $1 \times g$ - no repetition, (ii) $2 \times g$, (iii) $3 \times g$, and (iv) $4 \times g$.

The training process of the MLP utilized in the black-box data exfiltration attack scenario (with malicious code implementing data hiding functionality) is depicted in Algorithm 5.2.

Evaluation The performance of the model is evaluated after each epoch on the validation set $\mathbf{D}_{val}$ (consisting of benign data only) and the trigger set $\mathbf{T}$. In contrast to the models in the white-box setting, which are trained for 10 epochs, these models are also trained further for up to 120 epochs, to allow more detailed observations of their behavior. The dataset split used for the evaluation is shown in the right-most column of Figure 5.1. The final base model and the final attacked model performances are compared using a separate test set. Additionally, the attacker uses 20% of the training data as a validation set. This type of evaluation is used by the adversary to monitor the trade-off between the performance on the benign training set and the performance on the trigger set. The continuous evaluation at each training iteration is important, as the attacker has to achieve two conflicting key objectives:

- **Attack success**: Achieve a high similarity of exfiltrated data $\mathbf{D}_{ex}$ to the original data $\mathbf{D}$. As one option, the performance on the trigger samples can also be measured – the higher the performance on the trigger samples, the higher chance the attacker has to obtain exfiltrated data with high similarity to the original training set. Alternatively, the *similarity* of exfiltrated data $\mathbf{D}_{ex}$ to the original data $\mathbf{D}$ at each training iteration can be monitored as well, although the reconstruction and similarity calculation at each step leads to increased overall runtime. In the following evaluation of the experiment results, we use a similarity metric defined in Section 5.2.1 to measure the attack success.

Algorithm 5.2: Training algorithm $\mathcal{T}_{\text{mal}}$ augmented with malicious DEA functionality

```
Input: Training data $[\mathbf{X}_{train} \,|\, \mathbf{y}_{train}]$, Validation data $[\mathbf{X}_{val} \,|\, \mathbf{y}_{val}]$, Trigger set
       $\mathbf{T} = [\mathbf{X}' \,|\, \mathbf{y}']$, learning rate $\alpha$, batch size $N$, total epochs E
Output: Attacked model $\hat{f}'$
Initialize parameters $\theta'$ ($\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$) randomly for all layers $l$ of a network $f'$.
Let $Loss_{min}$ be the minimum validation loss observed so far, initialized to $\infty$.
for epoch in E do
    for batch in $[\mathbf{X}_{train} \,|\, \mathbf{y}_{train}]$ do
        Forward pass:
        $\hat{\mathbf{y}} \leftarrow$ forward_pass$(f', \mathbf{X}_{train})$.
        Compute loss:
         $Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_{i_{\text{train}}}\log(\hat{y}_{i_{\text{train}}}) + (1 - y_{i_{\text{train}}})\log(1 - \hat{y}_{i_{\text{train}}})]$
        Backward pass (backpropagation):
        Compute gradients: $\frac{\partial Loss}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial Loss}{\partial \mathbf{b}^{(l)}}$ for all layers $l$.
        Use optimizer to update parameters $\theta'$:
        $\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha\frac{\partial Loss}{\partial \mathbf{W}^{(l)}}$ and $\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha\frac{\partial Loss}{\partial \mathbf{b}^{(l)}}$ for all layers $l$.
    end
    for batch in adversarial_data $[\mathbf{X}_{train} \cup \mathbf{X}' \,|\, \mathbf{y}_{train} \cup \mathbf{y}']$ do
        Forward pass:
        $\hat{\mathbf{y}} \leftarrow$ forward_pass$(f', \mathbf{X}_{train} \cup \mathbf{X}')$
        Compute loss:
         $Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i\log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$ where $y_i \in \mathbf{y}_{train} \cup \mathbf{y}'$
        Backward pass (backpropagation):
        Compute gradients: $\frac{\partial Loss}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial Loss}{\partial \mathbf{b}^{(l)}}$ for all layers $l$.
        Use optimizer to update parameters $\theta'$:
        $\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha\frac{\partial Loss_t}{\partial \mathbf{W}^{(l)}}$ and $\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha\frac{\partial Loss}{\partial \mathbf{b}^{(l)}}$ for all layers $l$.
    end
    Forward pass:
    $\hat{\mathbf{y}}_{val} \leftarrow$ forward_pass$(f', \mathbf{X}_{val})$
    Compute loss on validation data
    $Loss = -\frac{1}{N_{val}}\sum_{i=1}^{N_{val}}[y_{i_{val}}\log(\hat{y}_{i_{val}}) + (1 - y_{i_{val}})\log(1 - \hat{y}_{i_{val}})]$
    Similarity Evaluation:
    $\hat{\mathbf{y}}' \leftarrow$ forward_pass$(f', \mathbf{X}')$
    Transform $\hat{\mathbf{y}}' \mapsto \mathbf{D_{ex}}$
    Calculate similarity of $\mathbf{D_{ex}}$ to $\mathbf{D}$ where $\mathbf{D} = [\mathbf{X}_{train} \cup \mathbf{X}_{val} \,|\, \mathbf{y}_{train} \cup \mathbf{y}_{val}]$
    if similarity $= 100$ then
        return optimal model $\hat{f}'$
    end
end
return $\hat{f}'$
```

- **Model utility**: Preserve the utility of the attacked model $\hat{f}'$, to stay comparable to the utility of the base model $\hat{f}$ trained on benign data $\mathbf{X}$ only. It is crucial for the attacker to prevent a significant decrease in the performance of attacked model $\hat{f}'$ on the benign data $\mathbf{X}$. The attacker conducts monitoring of the performance of the attacked model on benign data for each epoch, and can thus compare the performance of the model at each training epoch to the performance of the base model, which was trained on benign data only. For the experiments, additional monitoring of the performance of the base model is performed for each epoch.

The adversary is unable to actively control the stopping criterion based on similarity calculation at execution time, however, even in the above-described secure, isolated computing environment, the performance monitoring functionality can be embedded in the code itself. It is up to the attacker to define the stopping criterion for the training of the model. For instance, the adversary might be willing to endure a loss in training accuracy, and risk extending overall runtime, if it results in higher similarity. This is a strategic trade-off that the adversary has to take into account, depending on the objectives and constraints. This way, the attacker finds the optimal number of epochs given the attack success or model utility, or a combination of both. Additionally, the attacker needs to consider the attack runtime and its effect on the overall training runtime.

In the following experiments, an optimal number of training iterations is found when the attack success potential is high, and the *similarity* achieved is 100 percent. The resulting optimal model f'^* is saved at this point. This is the model to which the defense will be applied in the subsequent step.

Has the model not achieved a 100% similarity even after 120 training loops, the training is stopped, and the attacked model is saved after the last iteration.

The utility of the final, optimal model f'^* trained on $(\mathbf{D_{train}} \cup \mathbf{T})$ is evaluated on the independent test set. The attack success is again evaluated by data similarity calculation and the model's performance on the trigger set.

5.4.2 Defense: Targeted Local Neuron Pruning

As the data the model is trained on includes the trigger samples, it contains the learned representation of these samples as well. The aim of the defense is to remove the knowledge the model has learned from these trigger samples while preserving the knowledge learned from the benign training data. The objective is therefore to observe how the neurons behave on a forward pass of the benign training data. The neurons that do not respond to the benign data are removed, as these neurons are considered redundant to the underlying task. The assumption is that since these neurons are not responsible for the prediction on the underlying task, they might be responsible for the representation of the trigger samples (or overall not useful). Therefore, we refer to this process as targeted pruning. Algorithm 5.3 depicts this iterative process. As a first step in each iteration, the benign, original training data is passed through the network, and the activations of

Algorithm 5.3: Targeted Local Neuron Pruning

```
Input: Attacked model $f'$, number of neurons to prune from each layer $p$, benign
       training features $\mathbf{X}$, validation data $[\mathbf{X}_{val} \mid \mathbf{y}_{val}]$, evaluation algorithm
       evaluate_model_performance, $allowed\_performance$
Output: Defended model $f'^*$
$L \leftarrow$ number of hidden layers in $f'$
$n \leftarrow len(\mathbf{X})$
$j \leftarrow$ initial number of neurons in each hidden layer of attacked model $f'$
$pruned\_neurons \leftarrow p$
$current\_j \leftarrow j$
while $pruned\_neurons \leq j$ do
    $pruned\_j \leftarrow j - pruned\_neurons$
    $current\_j \leftarrow pruned\_j + p$
    Forward pass: Compute activations on benign training set:
     $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(L)} \leftarrow \text{forward\_pass}(f', \mathbf{X})$
    for $l \leftarrow 1$ to $L$ do
        Calculate the mean of activations per neuron $\mu_{h_i^{(l)}} = \frac{1}{n} \sum_{i=1}^{n} a_i^{(l)}$
        $\mathbf{h}^{(l)}_{k_1 \ldots k_{current_j}} = \text{sort}\left(\mathbf{h}^{(l)}_{1 \ldots current\_j}, \text{by } a^{(l)}_{1 \ldots current\_j}\right)$
        Remove neurons: $\mathbf{h}^{(l)} = \mathbf{h}^{(l)}_{k_1 \ldots k_{pruned\_j}}$
         $pruned\_neurons \leftarrow pruned\_neurons + p$
    end
    Forward pass: $\hat{\mathbf{y}}_{\mathbf{val}} \leftarrow \text{forward\_pass}(f', \mathbf{X}_{\mathbf{val}})$
    $performance \leftarrow$ evaluate_model_performance$(\mathbf{y}_{val}, \hat{\mathbf{y}}_{val})$
end
if $performance \leq allowed\_performance$ then
    return defended model $f'^*$
end
```

each neuron are collected. The values of the activations at each neuron averaged over all benign samples. Subsequently, these averaged activation values are sorted, separately per layer, and the chosen percentage of neurons with the lowest average activation value in each layer are removed from the network. Due to the pruning carried out at each layer separately, we refer to this as *local* pruning. At each iteration, the performance of the model is evaluated on the test set. Additionally, in order to monitor the defense effectiveness, the similarity of the exfiltrated data is monitored at all iterations. In a real setting, the defender, of course, would not be able to perform this step, as they do not possess the knowledge to construct the trigger set, but this data is required for the in-depth analysis of the defense.

CHAPTER 6

Results

This chapter presents and interprets the results of the experiments. The results of each attacks are shown, followed by the results of the corresponding defense. It elaborates on the utility of the attacks, with regards to the capacity of the models, the effectiveness of the attack, as well as the effectiveness of the attacked model on the original task. We make a comparison to the effectiveness of the base models. Additionally, we present the overhead generated by the incorporation of the exfiltration functionality with respect to the training time.

6.1 Attack: Least Significant Bit Encoding

The following section presents the results of the attack in the white-box setting, i.e. where the adversary has control over the learning process of the model and gains full access to the learned model parameters. The attack utility, along with the impact of the attack on the performance of the model is presented for the explored configurations.

6.1.1 Attack Utility

The LSB Encoding attack results in a 100% similarity of exfiltrated data to the original training data. Therefore, the utility of this attack largely depends on the capacity of the models - how much data can be hidden in the least significant bits of the model parameters.

Capacity The capacity of the model depends on the number of parameters in the model and on the number of least significant bits the adversary uses. For the models investigated, the number of parameters is shown in Table 5.4. For example, the model of size 1×1 comprises 1,764 parameters. The number of bits in this model is therefore 56,448, which is the maximum amount of bits that can be hidden in the model (utilizing all 32

Table 6.1: LSB Attack: Number of bits needed to represent the training data

	Label enc.transf.	One-hot enc.transf.	Gzip transf.
Bits per sample	384	91	~53
Bits per dataset	12,503,424	2,963,051	1,725,733

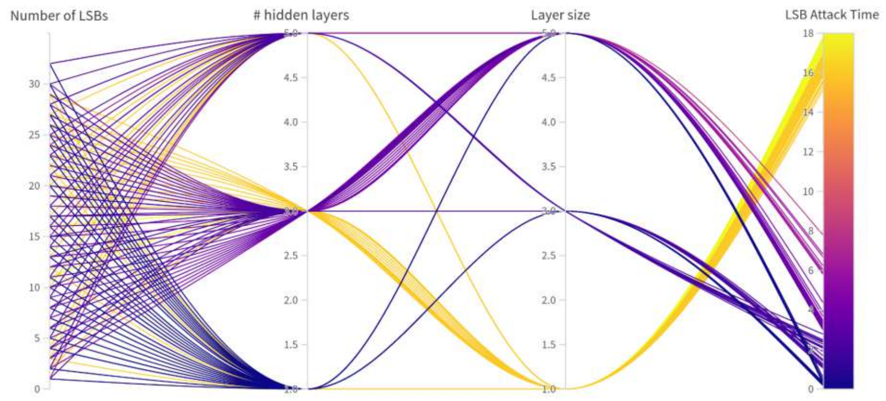
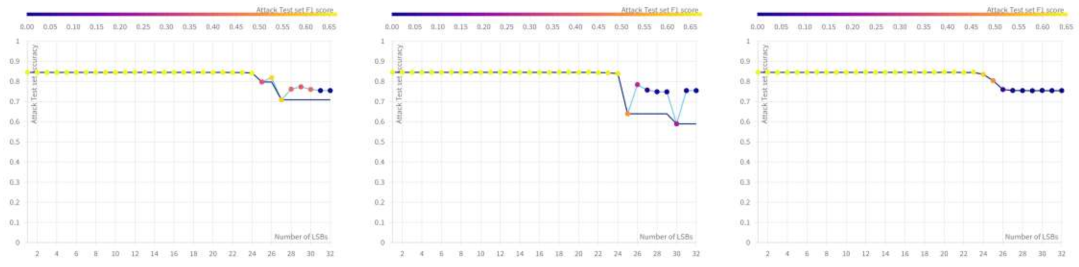


Figure 6.1: LSB Attack time (in seconds) w.r.t. attack configurations

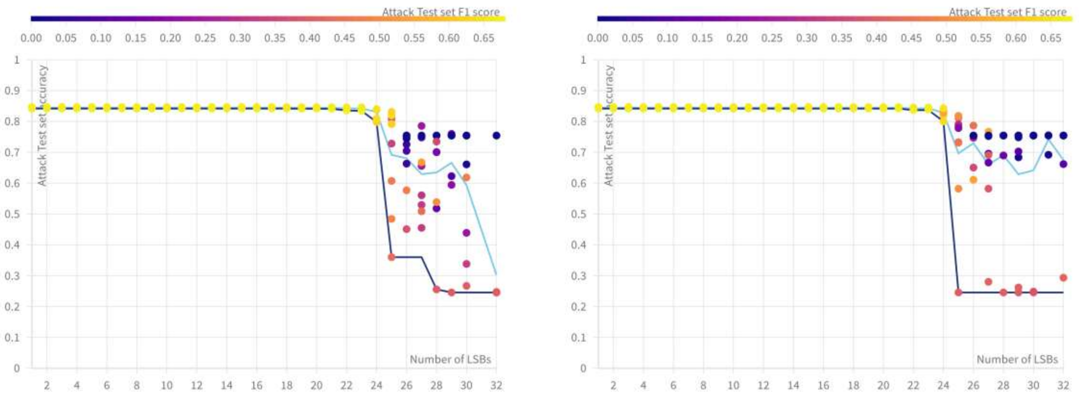
bits of each parameter, i.e. replacing parameter values fully with bits of training data). As shown in Table 6.1, when utilizing the label encoding transformation, a maximum of 147 instances can be hidden in the model. Using the one-hot encoding transformation, this increases to 620 instances of the training data, and with the gzip transformation, 1,065 samples can be hidden in the parameters.

Attack time The time required to carry out an attack is an important aspect for the adversary to consider. It consists of the calculation of the model capacity, the data transformation, and the encoding into the parameters itself. This time needs to be considered on top of the regular model training, as it increases the runtime of the provided code, potentially raising suspicions about hidden malicious activity. Figure 6.1 shows the attack time needed, for the attack using the label encoding transformation, depending on the various configurations of the models explored. The code execution time for the base model of size 1×1 was 42 seconds, therefore, the maximum overhead caused by the attack (i.e. worst-case scenario) increases the runtime by 42.9%. It is interesting to point out that there is a positive relationship between models of smaller size, and the attack time needed. This behavior can be explained by the limited capacity of small models, where longer data processing is needed in order to prepare and trim the data in order to encode it into the LSBs.



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.2: Impact of the LSB (Gzip) encoding attack on the performance of the model: The plot shows the change in accuracy with the increasing number of bits used for data hiding, marker color denotes the F1 score of the attacked model on the original task. The dark solid line indicates the minimum accuracy, and the light blue line shows the mean similarity



(a) Label encoding transformation (b) One-hot encoding transformation

Figure 6.3: Impact on the performance of the base models when the label and one-hot encoding transformations are utilized (average of models of 1×1, 3×3, 5×5 size)

6.1.2 Model utility

Modifying the least significant bits of the parameters potentially impacts the performance, as the parameter values are changed and thus the activations, confidences, and eventually rank of predicted classes are affected.

Figures 6.2 and 6.3 demonstrate the change in performance of the model after a certain number of bits of training data are hidden in the model. Accuracy is presented on the y-axis, while color of the markers represents the F1-Score, with the darkest value indicating an F1-Score of 0, which is equivalent to the zero-rule accuracy baseline. The dark blue line represents the minimum accuracy values, and light blue displays the mean

Table 6.2: The impact of the LSB encoding attack on the performance of original classification task of the underlying models. The cells show the number of attack bits b_a at which a change in performance starts to take place

Model Size	**1×1**				**3×3**				**5×5**			
Number of LSBs	22	23	24	25	22	23	24	25	22	23	24	25
Base Model Accuracy	84.44				84.48				84.47			
Attacked Model Accuracy	84.40	84.36	84.13	79.71	84.36	84.21	83.9	63.9	84.38	84.63	83.48	80.37
Δ Accuracy	-0.04	-0.08	-0.31	-4.73	-0.12	-0.27	-0.58	-15.18	-0.09	0.16	-0.99	-4.1
Zero-Rule Baseline	75.43				75.43				75.43			

accuracy. These figures generally show that the accuracy suffers no or minimal change when up to 23 bits of the parameters are used for attack. Figure 6.2 shows the impact of the attack that utilizes gzip transformation before the training data is hidden on the performance of models of different sizes. For the smaller models (of size 1×1 and 3×3), a notable drop in performance occurs at $b_a = 25$. At this point, the performance of the model of size 3×3 drops below the zero-rule baseline of predicting just the majority class. The model of size 1×1 drops to this baseline performance and predicts only the majority class, at $b_a = 27$. These two models thus have practically no remaining utility. The model of size 5×5 showcases a similar behavior, but the drop in performance is rather gradual, with $b_a = 27$ rendering the model useless. Table 6.2 presents details of these results. It shows the accuracy of the benign base model and indicates the effect of the attack at certain levels of changed bits b_a, by showing for each step of b_a the achieved accuracy and the change towards the previous step. This confirms that there is basically no effect when attacking up to 21 bits, and only a minor change with the 22nd bit. We can observe a slight increase in the accuracy of the model of size 5×5 at $b_a = 23$. This can be attributed to the model increasingly predicting the majority class, so the accuracy improves, however, there is a decrease by 1.49% in the F1-Score.

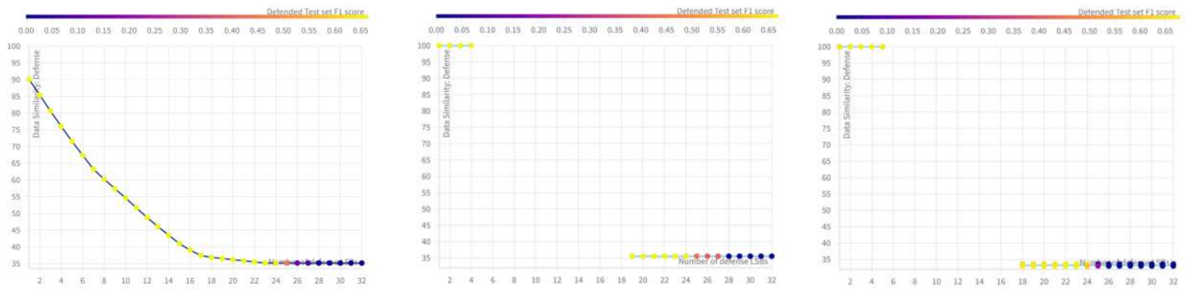
The models into which the hidden training data was encoded directly without compression, i.e. utilizing the label and one-hot encoding transformation, exhibit a comparable behavior regarding the change in model performance. Figure 6.3 depicts this behavior for models of all three sizes combined. The marker color again depicts the F1-Score, the lines represent the min and mean accuracy values. We can observe a sharp drop in minimum accuracy at $b_a = 25$. This is more pronounced for the models where one-hot transformation was used before the data, however, the mean accuracy is higher for $b_a > 25$. This can be explained better by additionally observing the F1-Score, which indicates that the models tend to predict the majority class only more often (the accuracy is, therefore, closer to the zero-rule baseline). The accuracy falling to this baseline means the models ability to predict the minority class has been removed. The lower accuracy, but higher F1-Score indicate, that the model's ability to distinguish between the classes is partially preserved. However, with the F1-Score below 50% this does not represent a particularly effective model.

6.2 Defense: LSB Sanitization

The utility of the defense is defined by the reduction in the exfiltration functionalities of the attack - a decrease in the similarity of the exfiltrated data to the original data demonstrates that the defense is effective. It is in the interest of the defender, however, to maintain an acceptable level of performance of the model. The following section therefore examines these two aspects, in order to find optimal defense settings.

6.2.1 Defense Utility

Removing the information from the least significant bits, when there is no error correction put in place by the attacker, causes the similarity of exfiltrated data to degrade immediately. In case the defender is using the same or a higher amount of bits to be removed than the attacker used for the encoding (i.e. $b_a \leq b_d$), the exfiltration functionality is fully removed. The adversary and the defender, however, do not have knowledge of the number of bits the other party is using. As shown in the previous section, an adversary is able to use up to 23 LSBs while causing no (or minimal) harm to the model's performance. A capable defender could perform a similar attack simulation, and could then reasonably assume that this is the minimal amount of bits an adversary will utilize for an attack.

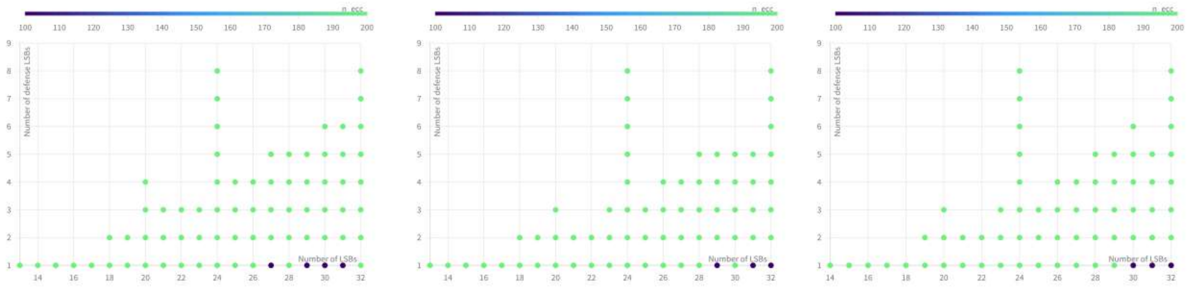


(a) No ECCs (b) 200 ECCs (c) 250 ECCs

Figure 6.4: Applying a defense of removing an increasing amount of b_d LSBs, when the attacker encoded into $b_a = 23$ LSBs. The attack (with label transformation) employing ECCs ((b) and (c)) remains successful against a defense removing four to five LSBs

If the hidden data is encoded into the parameters directly, without utilizing any compression, the change in exfiltrated data is equivalent to changing bits of the training data, in the respective transformed form. Figure 6.4 shows this decrease in similarity when an adversary hides the training data into 23 LSBs of parameters, and the LSBs are sanitized iteratively. This plot shows that a change of only one bit causes the similarity of exfiltrated data to drop below 90%, with $b_d = 12$, the similarity drops below 50%.

When gzip compression is used to transform the data before it is encoded into the parameters, the file is not recoverable after a defense is applied. Employing ECCs however, allows an attacker to adapt to an anticipated defense by increasing the robustness of the attack. Figures 6.4b and 6.4c show the change in similarity once encoding the data with additional ECCs is carried out by the adversary. It can be observed that even after four and five LSBs, respectively, are used for defense, the similarity of reconstructed data is 100%. Beyond this point, however, the data cannot be reconstructed due to the ECC decoding not being able to handle the amount of errors, or, the data is reconstructed yielding meaningless data.



(a) 1×1 (b) 3×3 (c) 5×5

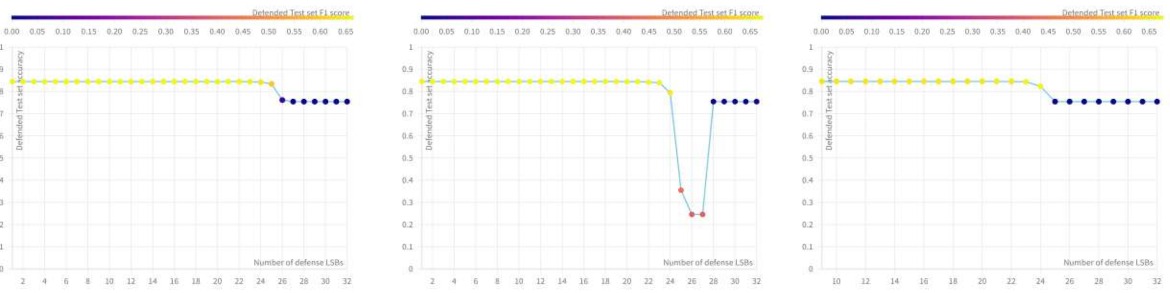
Figure 6.5: The combination of b_a and b_d where the attack (with gzip transformation) employing ECCs was successful despite applied defense. The color of the markers indicates the value of the ECC parameter applied

Figure 6.5 depicts the cases, where due to the employment of ECCs, the attack is successful (the gzip file is reconstructed leading to a 100% similarity). The color of the markers represents the ECC value used. It depicts that if the defender changes only one LSB of the parameters, the attacker would need to have used at least 13 LSBs in order to still exfiltrate the data. In the larger model, this increases to $b_a = 14$. The main takeaway from this figure is however, that the attacker is unable to reconstruct the training data when $b_d > 8$.

It is important to mention that the ECCs increase the number of bits needed to represent the underlying data. The ECCs encoding is applied on top of the result of the data transformation. Although they increase the chances of correct exfiltration, this happens at the cost of reducing the amount of data that can be hidden. For example, an ECC parameter of 10 increases the average number needed to represent a sample in a gzip format from 53 to 63 bits. The parameter value of 50 increases it on average to 77 bits, a value of 100 to 109 bits, and a value of 200 to 377 bits for one sample. For the direct encoding of data, this increase is equivalent. Therefore, as also shown in Figure 6.4, even though the reconstruction was not successful, the adversary is unable to use additional ECCs due to the limit imposed by the size of model parameters. Decreasing the amount of ECCs, however, did not lead to a successful recovery of the training data.

Sanitizing the LSBs of the parameters minimizes the chances of successful exfiltration, despite the usage of error correction functionality. The defense by sanitization is thus highly effective.

6.2.2 Model Utility



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.6: Performance of the models (of which the performance has been preserved after an attack was carried out) after a defense has been applied. The figures show the decrease in performance with an increasing number of defense LSBs being modified, with accuracy being depicted on the y-axis, and color representing the value of the F1-Score

Nevertheless, the impact of LSB sanitization on the performance of the underlying model needs to be investigated to gain a full picture of the usefulness of the chosen defense. Figure 6.6 shows the change in utility of the model given the amount of b_d sanitized. The plots depict the scores of the models, where the performance on the original task was preserved after an attack has been carried out. Similarly to the attack scenario, the performance starts to decrease after 22 bits have been modified. Once 25 bits are sanitized, the models are rendered useless.

For that reason, the defender can utilize this characteristic, and remove the hidden information. The defender is able to discard any hidden information at no cost, i.e. there is no need to give up on any model performance (unless it has already been compromised by an attack). The defense proves to be highly effective.

6.2.3 Parameter Modification

We observe that the defense that sets the values of LSBs to 0 acts as a quantization of the model parameters. It removes the precision of the parameter values, forcing them into bins. The bins depend on the bits with value 1 of the exponent of the floating-point value representation. Figure 6.7 shows this behavior on an example of weights from a model of size 1×1, with $b_a = b_d = 23$. The histograms depict the parameter values (only weights are shown). The x-axis shows the values of the parameters, the y-axis shows the number of parameters with the corresponding value. The parameters of the benign model can be seen in the leftmost part of the figure. The modified values resulting from hiding data in the LSBs of the parameters are depicted in the middle column of the figure. The parameter values with LSBs modified by the defense are shown on the right. The input-to-hidden layer weights are presented in the top row and the hidden-to-output weights in the bottom row. We can see how the precision of the values is reduced,

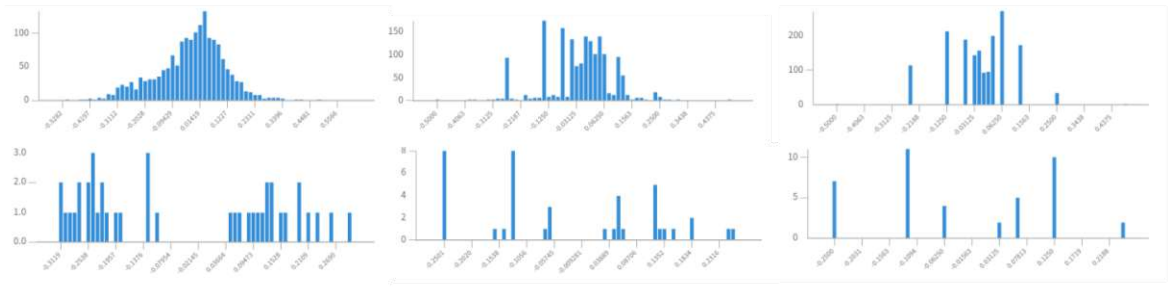


Figure 6.7: Histograms depicting the parameter values (weights) of the benign model on the left, the resulting values after the LSB encoding attack in the middle of the figure, and after defense on the right. The figure shows the weights of a model of size 1×1, with $b_a = b_d = 23$. The input-to-hidden layer weights are presented in the top row and the hidden-to-output weights in the bottom row.

causing many parameters to be reduced to the same value. We can see how the number of parameters per bin increases in the figures from left to right. For example, in the hidden-to-output parameters, in the benign model, there are 31 bins with at most three parameter values per bin, whereas, for the defended model, there are only seven bins, with up to 12 parameters per bin.

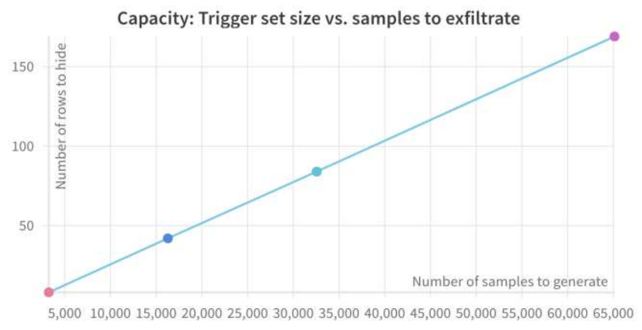


Figure 6.8: The number of trigger samples necessary in order to exfiltrate instances of the training data

6.3 Attack: Black-Box Attack

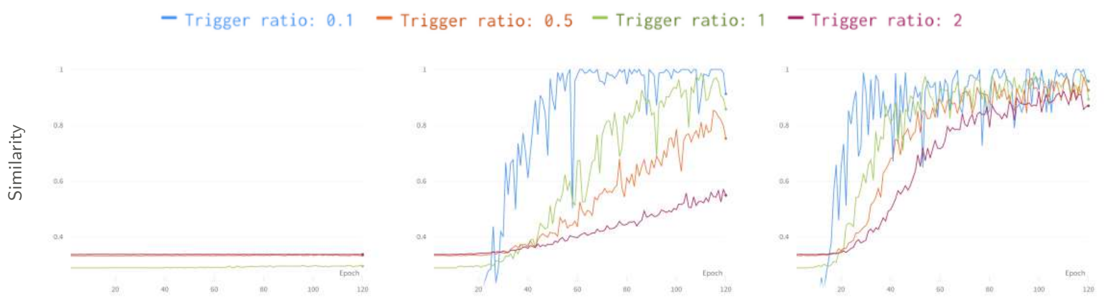
The following section presents the results of the attack in the setting where the adversary has control over the learning process of the model but does not assume full access to the model after the model is trained. The attack utility, along with the impact of the attack on the performance of the model, is presented.

6.3.1 Attack Utility

The attack utility is dependent on the learning capacity of the models, as well as the similarity of the exfiltrated data. The following section presents and discusses the results of both of these aspects.

Learning Capacity In order to explore the learning capacity of the models at hand, four different settings regarding the amount of generated samples are explored. The amount of trigger samples is expressed as the ratio to the size of the training set (0.1x, 0.5x, 1x, and 2x). With one trigger sample being labeled with one bit of the original training data, 384 trigger samples are needed to hide one data sample, as the classification task is binary. Figure 6.8 visualizes this relationship. It is easy to see that in order to exfiltrate 170 samples, over 65,000 trigger samples are necessary - which is two times the size of the training set.

We investigate the learning capacity of the models in relation to the model size. Generally, larger models are expected to have a larger learning capacity, as they have more parameters that allow them to encompass more complex relationships. Figure 6.9 confirms this behavior. It shows the *maximum* similarity results for four different sizes of the trigger set for the investigated attacked models of different sizes. It shows that the smallest models do not learn the trigger samples. The models of size 3×3 manage to learn the smallest trigger set (3,256 samples), with the best result reaching a 100% similarity of exfiltrated data to the original data, at the 57^{th} epoch. During training, the similarity of the data seems to fluctuate and does not seem to stabilize. The model also learns the larger trigger



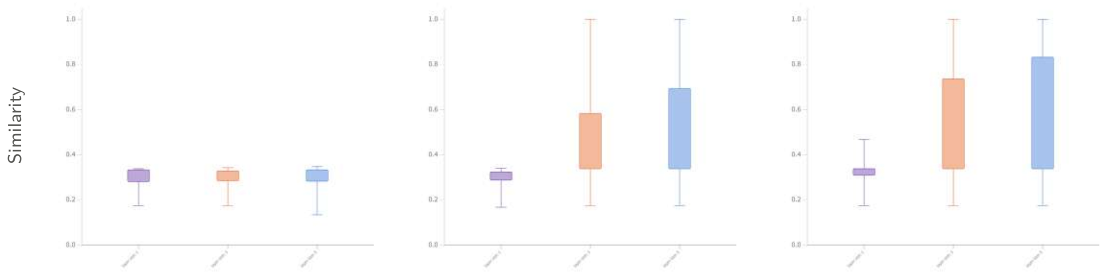
(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.9: The ability of models to learn different amounts of trigger samples (expressed as a ratio to the training set size) - maximum similarity achieved

set, which is of the same size as the training set, better than the trigger set half the size of the training set. A possible explanation for this behavior might be that since the data is labeled with bits of the training data, there are no structured common characteristics among the samples that are put to the same target class. However, the attacker has no option to selectively label similar trigger samples with one of the classes of the target attribute (0 or 1 bits of the training data) to account for learning inconsistencies resulting from such behavior, as the trigger generation needs to be deterministic in order to be recreated for exfiltration. It could, therefore, happen, that for a certain number of trigger samples, the bit representation of training data used as labels match the structure in the original data better, which results in the model learning the trigger set better than other trigger set sizes.

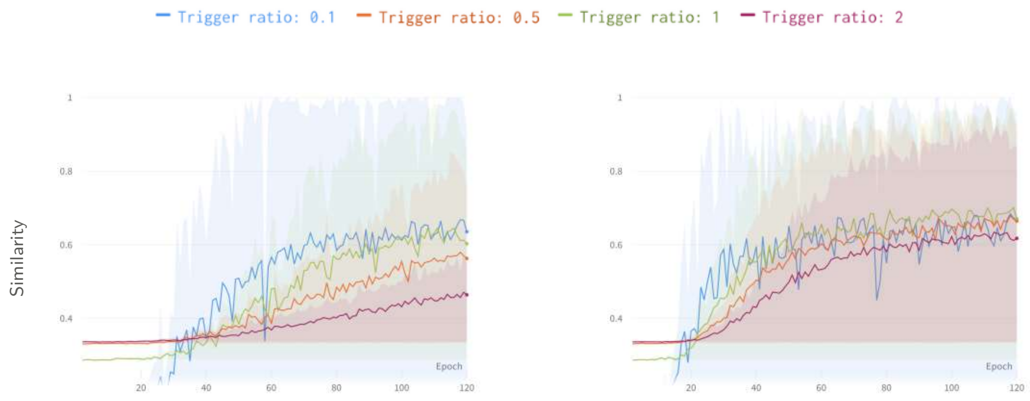
The models of size 3×3 learn the trigger set that is twice the size of the training set better than the smallest models we consider do, achieving over 50% in similarity. The 5×5 models start achieving higher similarity sooner than the 3×3 models: when learning the smallest trigger set, reaching up to a 98.84% similarity already at the 29th training iteration and a 100% similarity at the 76th epoch. Interestingly, the 100% similarity is only reached later than in the smaller model. However, this might result from the instability of trigger sample learning mentioned above. These large 5×5 models manage to learn even large trigger sets more accurately. On the largest trigger set (twice the size of the training data), they reach up to 92.89% in similarity, however, it takes longer for the similarity to start increasing. We, therefore, see that the models require more training loops to remember such amounts of trigger samples.

In order to further investigate what model size is necessary to allow learning of the trigger samples, different values for the layer size were investigated. Figure 6.10 depicts the similarity of exfiltrated data achieved by models with a different number of hidden layers. Each of the plots represents models with the same number of hidden layers, with increasing layer size (from left to right). Layer size of 1 is represented by purple color, layer size of 3 by orange color, and layer size of 5 by blue color. The models comprising one hidden layer, with layer sizes of three (×input size), and five (×input size) (shown in



(a) 1×1, 1×3, 1×5 (b) 3×1, 3×3, 3×5 (c) 5×1, 5×3, 5×5

Figure 6.10: Similarity given hidden layer size: The impact of layer size on similarity achieved by models of 1, 3, and 5 hidden layers: blue denotes models with layer size 5, orange 3 and purple represents layer size 1

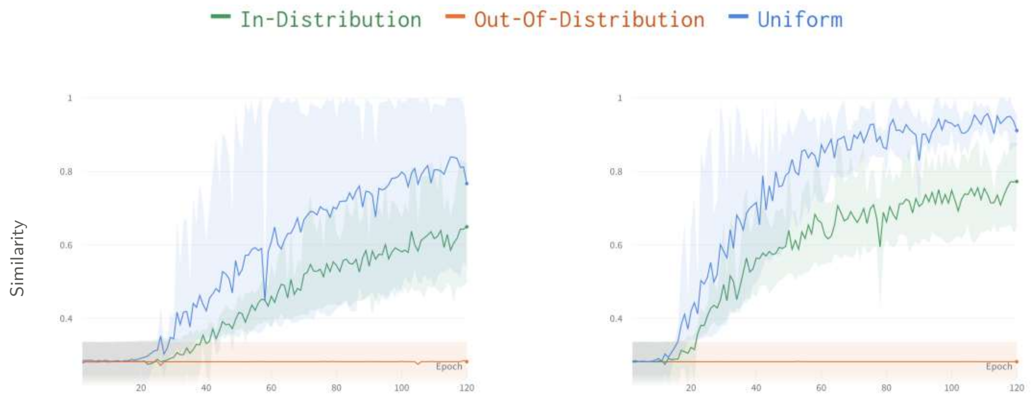


(a) 3×3 (b) 5×5

Figure 6.11: The ability of models to learn different amounts of trigger samples (expressed as a ratio to the training set size). The mean similarity of exfiltrated data to the original data is shown on the y-axis, with the shadows representing the minimal and maximal similarity values achieved

Table 5.4) were unable to learn well and achieved results comparable to the models of size 1×1. This behavior is shown in Figure 6.10a. Also the models with three hidden layers of layer size 1(×input size) performed similarly. The models with 5 hidden layers of layer size 1(×input size) achieved 50.41% in similarity after 116 epochs. We can therefore see, that in the investigated scenarios, the model requires a hidden layer size of at least three(×input size) and models with at least three hidden layers to achieve a 100% similarity on the smallest trigger set.

Figure 6.11 show the mean similarity of exfiltrated data compared to the original training data at each training iteration, grouped by the trigger set size. Figure 6.11a also shows that on average, the smallest trigger set was learned by the model quicker, whereas



(a) 3×3 (b) 5×5

Figure 6.12: Impact of different modes of trigger generation on the similarity of exfiltrated data

the model struggled with learning the largest trigger set. In comparison, for the large model shown inFigure 6.11b the trigger set size does not have a notable impact on the resulting mean similarity. It even shows that the models which learned the larger trigger set (1:1 ratio to the original training set) on average achieve a slightly higher similarity of exfiltrated data starting after 75 epochs of training. A possible explanation for this behavior is overfitting of the model on the original training data when the trigger set is smaller, as we will elaborate in detail in Section 6.3.2.

Similarity - Trigger generation mode

This section discusses how the mode of trigger generation impacted the learning of the trigger samples. As shown in the previous section, the small model of size 1×1 did not have the ability to learn the trigger samples well, and even after increasing the size of the hidden layer, the ability did not increase considerably. Figure 6.12 shows the ability of the models to learn depending on which knowledge the adversary used to generate the trigger samples. For each group of trigger set generation, the evolution of the mean similarity (incl. min-max values) is shown. If the trigger samples are out-of-distribution of the benign training data even the larger models do not have the ability to learn them. When the values of the instances in the trigger set are distributed following the distribution of the benign data, i.e. mimicking the benign data, the models achieve better results, with the mean similarity reaching over 60% for 3×3, and over 75% for 5×5 models. The best models of size 3×3 in this scenario were able to reach over 80% similarity, with models of size 5×5 increasing this similarity up to 90%. The models were able to learn the uniformly distributed trigger samples the best - they show the best performance in terms of mean similarity across all models. For the models of size 3×3, they reached up to 85% similarity on average, with the best models reaching 100% similarity, thus allowing for perfect exfiltration of the hidden data. Continuing the pattern, for the models of size 5×5, the average similarity was even higher, reaching up to over 95%, with the best models achieving perfect similarity of benign data to exfiltrated data. The potential reason why the models were able to learn the trigger set with uniformly distributed samples the best could be that the model more easily identified a pattern in this data, or simply, the generated instances labeled with the bits of training data, resembled the original training dataset more. The investigation of this notion was not carried out, as the data generation needs to be deterministic, and can thus not depend on the original training data.

Again, it can be observed, that the larger models tend to learn faster, requiring fewer training iterations in order to learn the trigger samples. For the largest models, however, especially regarding the best-performing models of this size, there seems to be more instability in the learning process of the trigger samples. This could possibly be explained by the tendency of larger models to overfit on the benign training data, causing unstable behavior on the trigger samples. This behavior is addressed below, in Section 6.3.2 and Figure 6.16.

Successful attack configurations

Overall, we identified 16 configurations (out of 432) where the attacked model led to a similarity of 100%. The common characteristics leading to the most successful attacks are **uniform** distribution of trigger samples, **0.1** trigger ratio and model size where with ≥ 3 hidden layers, and of layer size ≥ 3). The five left-most columns of Table 6.3 present the identified successful configurations.

The repetition of the trigger samples (i.e. the oversampling) did not have an effect on the success of the attacks. The oversampling results are not depicted, as the 16 most successful experiments contained all four settings of this simple oversampling, which led to identical performance regardless of the number of repetitions.

Attack time Figure 6.13 shows the difference in model training times per each epoch between the training of the base models vs. malicious training of attacked models. Blue color represents the training of the base models, while orange represents the attacked models. The models are grouped by the number of hidden layers, with increasing layer size (from left to right, indicated by the color shade). We can see that due to the forward pass and backpropagation being performed twice, and on additional data per each epoch (as presented in Algorithm 3.3), the epoch runtimes increase in comparison to the base model epoch runtimes. As expected, the depth of the network and layer size also impact the epoch runtimes.

Similarly, Figure 6.14 depicts the total training time of the models which successfully learned the trigger set and produced exfiltrated data samples identical to the original samples. The grouping, ordering, and color represent the same characteristics as in the plot above (Figure 6.13). The number of epochs at which the training of the attacked model is stopped is shown in Table 6.3. The reason to compare models trained for a

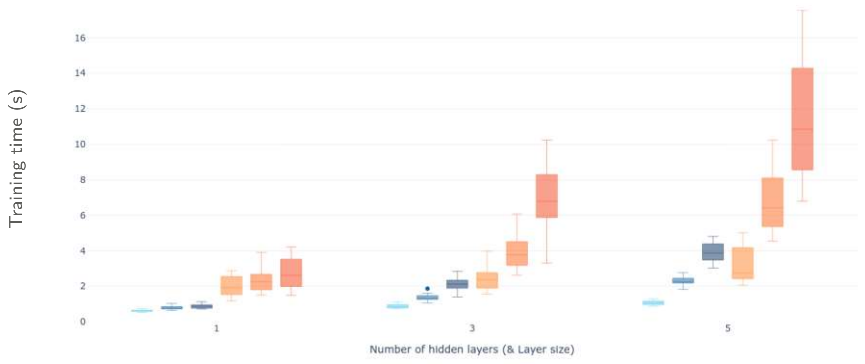


Figure 6.13: Increase in training time per epoch after inclusion of data exfiltration functionality, shown per number of hidden layers ordered by layer size, with blue color denoting the base models and orange the attacked models. The darker the color, the larger the size of hidden layers

different number of epochs is to compare models successful at exfiltration and also to show how much training time is required for the adversary's training to reach a similarity of 100% and therefore exfiltrate instances identical to those of the original training data. For this reason, an adversary might resort to a smaller model, as it results in a shorter runtime, lowering the chances of the attack raising suspicion.

6.3.2 Model utility

Figure 6.15 depicts the continuous evaluation of one attack setting during the training process. The blue lines show the performance of the base models, with the darker shade indicating the accuracy, and the lighter blue shade showing the F1-Score. The red line represents the validation set accuracy of the attacked model, while the orange line depicts the respective F1-Score. Further metrics, as described above, are tracked, although not shown here. A zero-rule baseline on the validation set is depicted by grey color. It serves for comparison of the model utility - once the performance decreases below the baseline, the model performs no better than a model predicting exclusively the majority class. The green line shows how the similarity of the exfiltrated data to the original data changes after each epoch, i.e. how successful is the model at exfiltration attack at each epoch. The metrics are tracked on the test set for each epoch as well to allow for comparison within the experiments in this work. We treat the test set as a separate set, and the score on the test set represents the score the model would achieve once deployed.

In Figure 6.16, the impact of the attack on the mean performance of the models is shown. The results are shown on the test set, in order to compare the final performance of the models. The accuracy of the model stays above the baseline for the whole course of

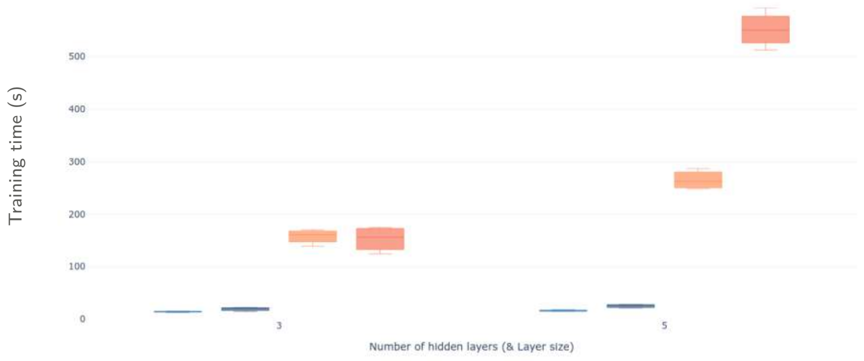


Figure 6.14: Increase in total training time after inclusion of data exfiltration functionality: Total training time of the successful attacks needed to in order to achieve 100% similarity shown per number of hidden layers ordered by layer size, with blue color denoting the base models and orange the attacked models. The darker the color, larger the size of hidden layers

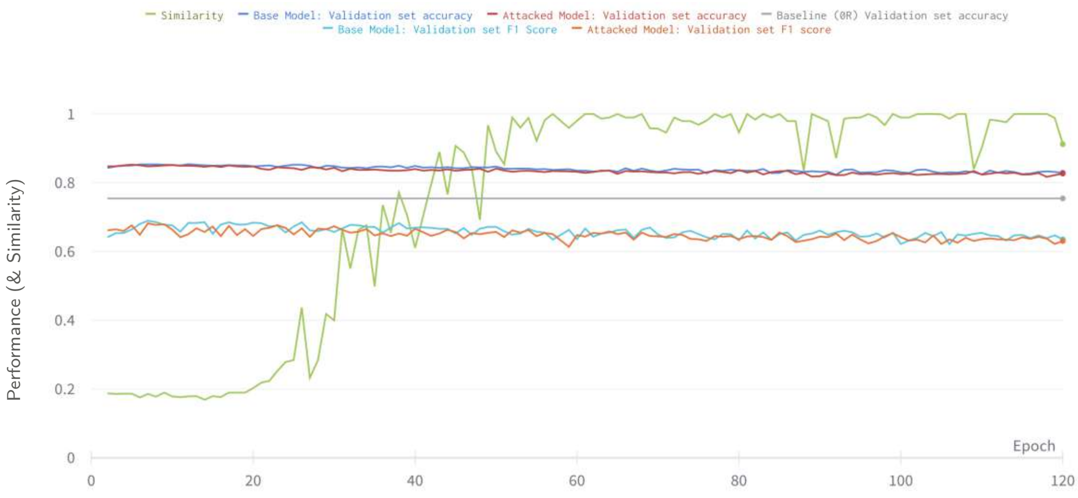
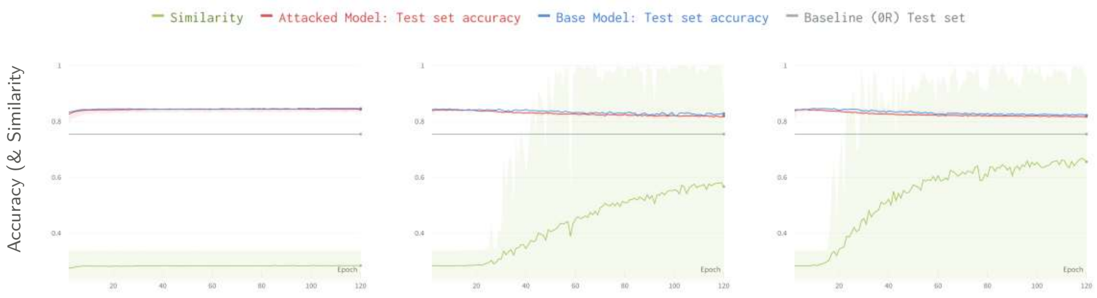
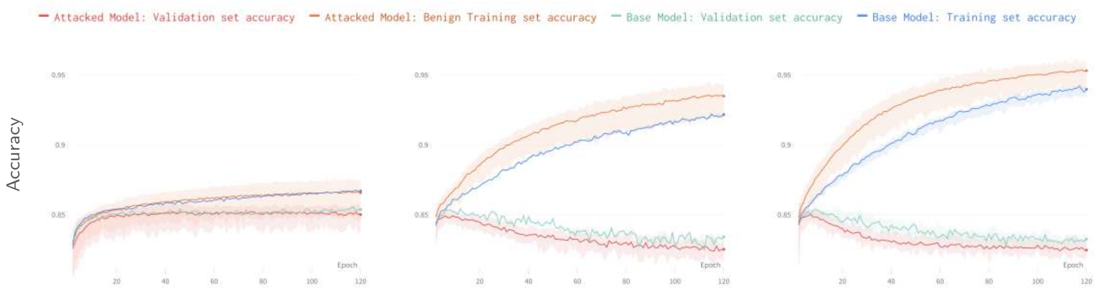


Figure 6.15: Example of performance tracking in experiments, and by an adversary on a model (the example shows a model of 3×3 size, 0.1 ratio of trigger samples to original data samples, no oversampling of triggers, uniform trigger generation). The y-axis represents the performance of the model, as well as of the attack (the metrics are shown in terms of percentages)



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.16: The impact of the attack functionality on the model performance for three different model sizes (shown on the test set), where the accuracy of the attacked model is depicted by the orange color, the accuracy of the base model by the blue color, and similarity is shown in green. The mean values are shown, with the shadows indicating the minimum and maximum values reached at each epoch



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.17: Performance of the base vs. attacked models (in terms of accuracy) on the training and validation set at each epoch, depicting the overfitting of the models that takes place when trained for a high number of epochs. The performance of the base models is represented by blue and green lines, while red and orange colored lines denote the performance of the attacked models

training for all models. Looking more closely, it can be observed that the performance of the model of 1×1 size increases at a slower rate on the underlying task than the other models' performance. We can observe that the attacked models' performance is slightly lower than the base models' performance during the training process. With increasing training iterations, the models' performance is decreasing. However, the accuracy of the base models is decreasing at a similar rate as the accuracy of the attacked models. Therefore, this decrease can be rather attributed to the overfitting of the models on the underlying task. But as shown in the section above, and also depicted by the green color in Figure 6.16, training for many more epochs is required in order for the models to learn the trigger samples.

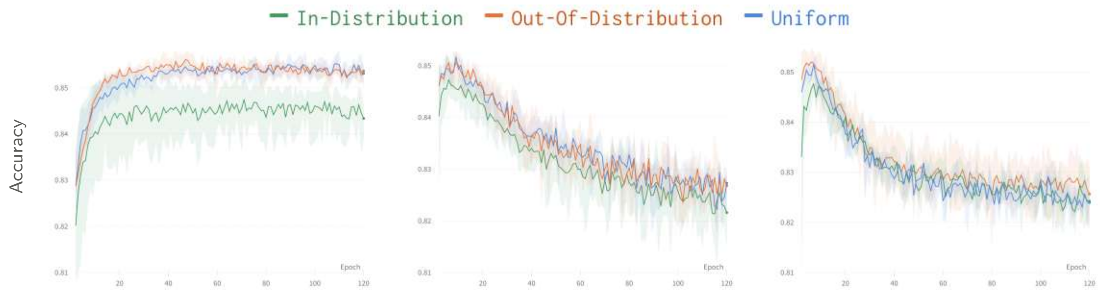
Table 6.3: Difference of the attacked model performance to the base model performance for models that achieved 100% similarity

#Hidden layers	Layer Size	Trig.ratio	Gen.mode	Opt.epoch	Δ Accuracy	Δ Precision	Δ Recall	Δ ROC AUC	Δ F1 Score	Δ Epoch Time
3	3	0.10	uniform	57	**-1.51**	-7.18	**4.92**	**0.65**	**-0.21**	1.26
3	5	0.10	uniform	40	-1.63	**-3.71**	-2.76	-2.01	-3.21	1.30
5	3	0.10	uniform	65	-2.20	-4.36	-5.70	-3.38	-5.08	3.21
5	5	0.10	uniform	76	-2.28	-7.69	2.43	-0.69	-2.23	6.52

Figure 6.17 shows the overfitting behavior of the models. We can observe how the training and validation accuracy of both the base and the attacked models behave with increasing training epochs. After a certain point (approximately 25 epochs for models of size 1×1, below 10 epochs for larger models), the models start learning the training set better, which leads to a degradation of their performance on the validation set. This is especially true for the larger models. This is the reason why base model training would be stopped at 10 epochs. The additional training epochs of the base model are depicted in order to show that it is not only the inclusion of the trigger set that leads to the degradation of performance but also the nature of the training itself. However, as shown above, these extensive training iterations are necessary in order for the models to learn the trigger set.

Table 6.3 shows the impact of the attack on the utility of the models most successful w.r.t. the attack. It depicts the differences in the performance expressed in terms of multiple metrics - the sign and the color depicting the increase/decrease in model performance. The bold print indicates the most favorable change in performance. As mentioned in the sections above, the repetition of the trigger samples led to the same performance of the attack as if there was no repetition. It has also had no impact on the performance of the models on the original task. Therefore, the 16 most successful attack results can be reduced to the 4 rows in this table, as the additional 12 differed only in having more repetitions of the trigger set. The table also shows how many training iterations were needed until a 100% similarity was reached by each of the models. The accuracy of these models on the original task decreases based on the number of hidden layers, and also with the increasing layer size. For the smallest and largest successful model shown, the recall improves through the implementation of the attack functionality, however, this of course comes with the trade-off regarding precision. As expected, the training time per epoch increases. For the smallest model shown here, training time for one epoch takes 86.72% longer than an epoch of a base model (although this is a high increase in training time, in our experiments, this increase in absolute numbers corresponded to 1.26 seconds, which is not necessarily noticeable for the defender). This difference in the epoch time needed for a base model vs attacked model also increases with model size.

To further investigate which configuration affects the model utility, Figure 6.18 shows the mean performance of the models of different sizes given the mode of trigger generation. For the smallest model, it can be observed that including trigger instances generated with the same distribution as the training data in the training of the attacked model leads to an overall lower performance of the model. We could interpret this as the trigger instances "confusing" the model since they introduce samples similar to the training data



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.18: The impact of trigger generation mode on the model performance for 3 different model sizes (shown on the test set)

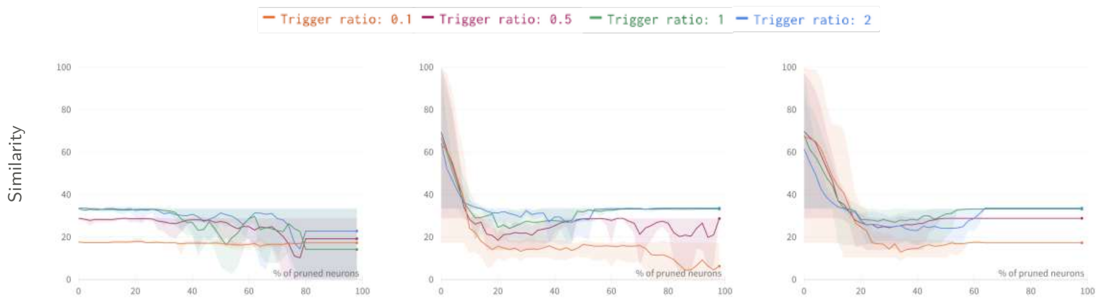
but with essentially random labels (bits of training data). Therefore, any pattern the model learned from the benign data is being invalidated, and the model is thus misled. The out-of-distribution data impacts the model performance the least, especially at the beginning of the training. Towards the later training iterations, the models' behavior is comparable to when uniformly distributed trigger samples are included. We can also observe very little degradation of the performance of the smaller model, which, as discussed above, has likely two reasons: the small model does not have the capacity to overfit on the original training data and also does not learn the trigger samples. The middle-sized models experience a larger decrease in performance, which does not seem to largely depend on the mode of trigger generation, although the in-distribution data causes the models to have a slightly lower performance. The largest models showcase even a smaller difference in impact on the model performance given the mode of trigger generation. It can be seen, that it takes this model more training epochs to learn the in-distribution data than the other trigger instances.

6.4 Defense: Targeted Local Neuron Pruning

As a defense in the black-box setting, we try to identify those neurons that are responsible for learning the triggers, and then remove them from the model, to de-activate the exfiltration capability, as described in Section 5.4.2. The behavior of both the model performance and the similarity of exfiltrated data to the original data was observed at each iteration of neuron pruning.

6.4.1 Defense Utility

Neuron pruning has been applied to all the models investigated. The results below demonstrate an effective removal of the learned representation of the trigger samples. The largest investigated models, of size 5×5, on average lose their successful exfiltration capabilities after pruning 10% of the neurons, as the similarity drops below 50%. The middle-sized models, (i.e. 3×3, the smallest ones that were able to achieve a 100%

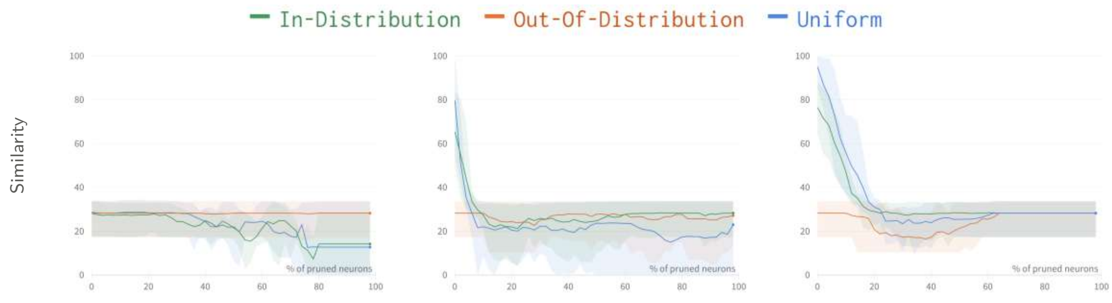


(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.19: Impact of targeted neuron pruning on the mean similarity of exfiltrated data given the ratio of trigger instances to benign training instances (the bounds indicate the minimum and maximum values)

similarity after an attack) respond to the pruning as a defense faster than large models. The defense causes a sharp drop in the similarity of exfiltrated data: on average, the similarity decreases to less than 50% already after pruning 6% of the neurons. As shown in Section 6.3.1, the smallest models investigated within this work did not learn the trigger samples well. However, the defender does not know whether (and which) attack functionality has been implemented into the model. Therefore, the defender performs neuron pruning here as well, removing some of the learned information after pruning 30% of the neurons. Figure 6.19 shows the results of pruning w.r.t. the ratio of trigger samples to the benign training data. With an increasing amount of learned trigger samples, the pruning procedure removes the learned information faster, requiring the removal of fewer neurons to neutralize the attack. This behavior is more pronounced for the largest models (5×5). We can only speculate about the reason for this behavior. For example, the model learns smaller amounts of data at a faster rate, thereby reinforcing the learned information through further training, which leads to this information to be not as easily removed. Therefore, the information could be learned better also by neurons responsible for learning the original data. Removing the neurons not responsible for primarily learning the original data might enable the model to retain this information for longer. Another possible explanation is that simply the pattern in the smallest generated trigger set w.r.t. to the target variable is more similar to the original data (i.e. trigger samples similar to those samples from the original data belonging to a positive class are (coincidentally) labeled as a positive class more often), and therefore the responsibility of learning both the trigger and the benign samples is spread more evenly throughout the neurons in the network. With a larger trigger set this pattern might be broken. However, this would require further investigation. It can be observed that some of the models that learned the smallest amount of trigger samples, i.e. with 0.1 ratio to the training set, retain a 70% similarity of the exfiltrated data even after 16% of neurons have been pruned.

Previously (Section 6.3.1), it was observed that the mode of trigger generation leading



(a) 1×1 (b) 3×3 (c) 5×5

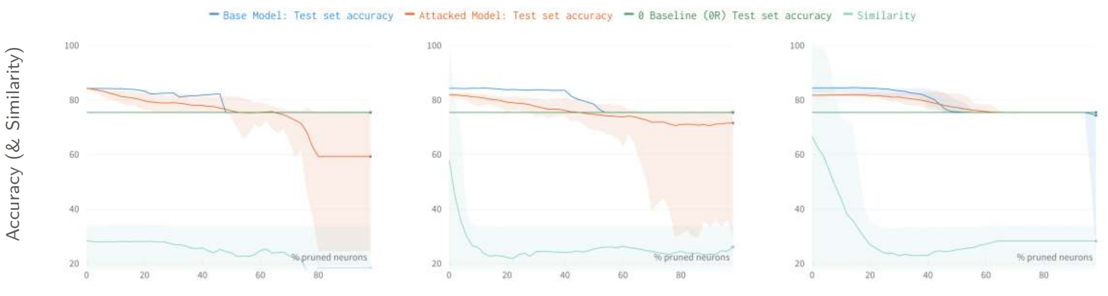
Figure 6.20: Impact of targeted neuron pruning on the similarity of exfiltrated data given the mode of trigger generation

to the best attack utility was the generation of uniformly distributed instances. In the 3×3 models that utilized the uniform generation, a sharp drop in attack utility can be seen, showing the effectiveness of the defense. Although the in-distribution data was not learned perfectly during the attack, the removal of this information is slightly slower and thus more difficult than in the uniform distribution group, i.e. the decrease in similarity was steeper for the models that learned uniformly distributed triggers. The learned in-distribution triggers lead to a slightly more robust attack against this defense, however, if pruning is continued, it eventually also leads to the removal of the learned information. For the 5×5, the information learned about the uniformly distributed and in-distribution triggers is being removed at a similar rate, although the starting points differ. Out-of-distribution data was not learned well by any model, reaching only low values of similarity, which does not constitute a successful attack. The applied defense causes slight changes in resulting similarity for the larger models that utilized this generation.

6.4.2 Model Utility

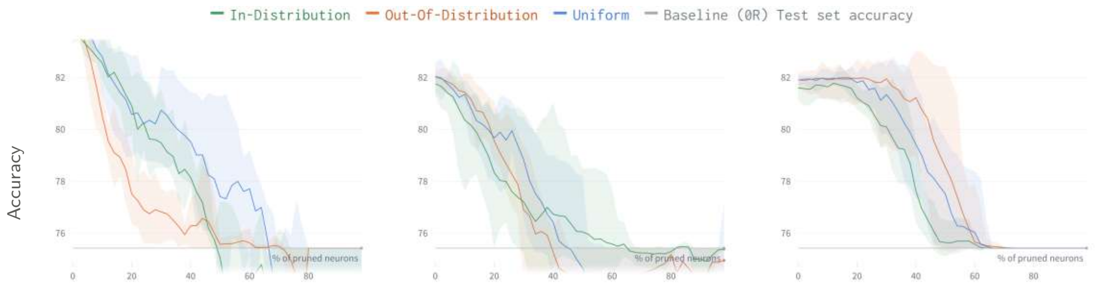
The impact of neuron pruning on the performance of the investigated models is explored on the previously attacked, as well as benign models.

It can be observed that the base models retain effectiveness on the original task at a level comparable to when no pruning is performed for a longer time than the attacked models. This suggests the universality of the defense, allowing the defender to apply the defense without losing too much performance. However, the performance of the attacked models has already been affected by the attack. Generally, the larger the model, the more pruning is needed to impact the performance of the model negatively, as the larger models have a lot of spare capacity that is not or only marginally used to represent the tasks. For the smallest investigated models, of size 1×1, the attacked models lose their effectiveness on average after pruning more than 50% of neurons. The respective base models lose the ability to differentiate between the positive and negative classes



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.21: Impact of targeted neuron pruning on the performance (accuracy) of the attacked and base models



(a) 1×1 (b) 3×3 (c) 5×5

Figure 6.22: Impact of targeted neuron pruning on the performance (accuracy) given the trigger generation mode of trigger samples at attack

completely after 54% of neurons are pruned from the network. Figure 6.21 depicts this by including the zero-rule baseline.

Given the mode used for the generation of the trigger samples, an interesting behavior can be observed. The pruning of the neurons of the 1×1 models which were trained on the out-of-distribution trigger samples causes the accuracy to drop on average at a faster rate, with the in-distribution trigger generation having the second largest impact on model performance. The models trained using uniformly generated triggers lose their performance on the original task at a slower rate. In the 5×5 models, however, the performance of the models trained using the in-distribution samples is impacted the most by the defense. The performance of the models of this size trained on out-of-distribution samples even shows an improvement in performance on the underlying task, before dropping below baseline. This behavior can be observed in Figure 6.22

The most important aspect to pay attention to is the trade-off between the model performance and the similarity of exfiltrated data. Overall, the similarity of exfiltrated data drops at a significantly faster rate than the model performance. Figure 6.23 shows the change in the mean performance of the models which were most successful at exfiltration, i.e. the models that achieved 100% similarity at attack. The performance of the attacked

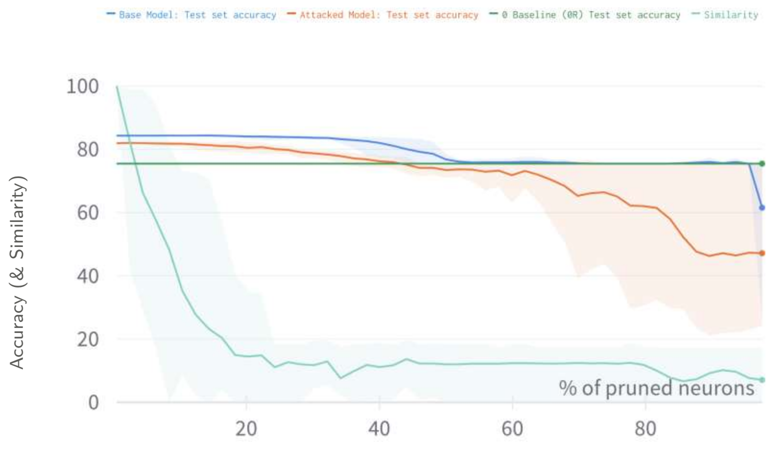
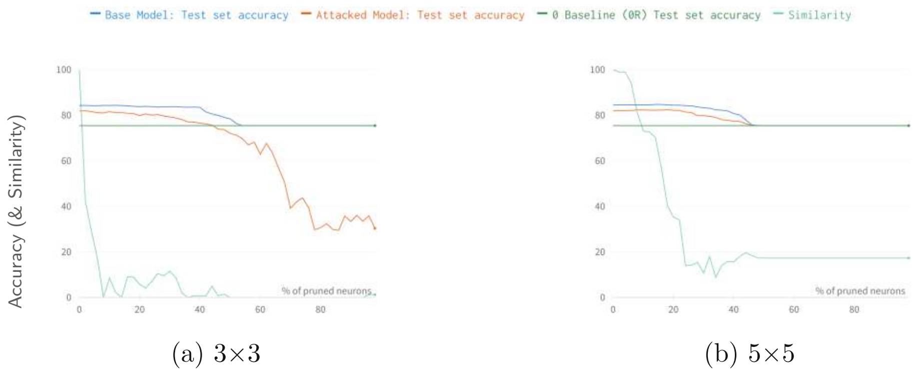


Figure 6.23: Impact of targeted neuron pruning on the performance (accuracy) of the most successful attacked and respective base models. The plot shows mean values, with the boundaries indicating the minimum and maximum values



(a) 3×3

(b) 5×5

Figure 6.24: Impact of targeted neuron pruning on the performance (accuracy) of the most successful attacked models (100% similarity at attack) of different sizes and the respective base models

models starts to decrease faster than that of the base models. However, the exfiltration functionality decreases at a faster rate, leading to a steep drop in the average similarity. Even the models retaining the similarity for the longest (as indicated by the maximum bounds) experience a major loss of the exfiltration functionality before the effectiveness of the models on the original task is fully removed.

To evaluate how the model size contributes to the impact of the defense on the behavior of the models which were successful at exfiltration, Figure 6.24 shows the change in model accuracy at each pruning step. The pruning seems to affect the base model faster when the model is larger, whereas the smaller base model allows for the removal of a larger

Table 6.4: Change in model and attack utility of the models most successful at exfiltration after application of defense

				10% pruned neurons			20% pruned neurons		
# hidden layers	**Layer Size**	**Trig.ratio**	**Trig.gen.**	**Δ B. Accuracy**	**Δ A. accuracy**	**Δ Similarity**	**Δ B.accuracy**	**Δ A. accuracy**	**Δ Similarity**
3	3	0.10	uniform	-0.01	-0.35	**-91.48**	-0.50	-2.06	-94.25
3	5	0.10	uniform	**0.23**	-0.01	-62.04	**0.13**	-2.28	-78.39
5	3	0.10	uniform	-0.30	-1.05	-67.35	-0.81	-2.84	**-95.13**
5	5	0.10	uniform	**0.08**	**0.38**	-26.91	-0.05	0.20	-64.75

percentage of neurons before the model is rendered useless. When observing the behavior of the attacked models, this trend is inverse: the larger model retains almost unchanged accuracy even when over 20% neurons are pruned. The performance of the 3×3 starts to degrade much sooner with the application of pruning. It could potentially be explained by the smaller model having a lower learning capacity, therefore the neurons responsible for learning the trigger samples are also responsible for learning the original task. Therefore, removing neurons removes the classification accuracy on the primary task at a faster rate, than in larger models. However, both models fully lose their predictive capabilities at the same time (i.e. reaching the baseline).

Table 6.4 shows the respective values of the performance after 10% and 20% of neurons are pruned. The models in this table correspond to the models represented in Table 6.3 after the attack was carried out. Figure 6.24a is represented by the first row. It shows that the attacked model of this size loses the exfiltration capabilities already when 10% of neurons are pruned, with the similarity decreasing by 91.48%. This comes at a trade-off of 0.35% accuracy decrease on the original task. Figure 6.24b is represented by the last row. Here we can observe a smaller loss of the exfiltration functionality, however, this model's performance improves slightly with pruning. The base model also improves slightly with the pruning of 10% of neurons; after 20% are pruned, the base model loses 0.05% in accuracy. The second and third rows represent the behavior of the models where the layer size is adjusted as well. The attacked model of this size experiences a slight decrease in accuracy, but at 10% pruned neurons, a large portion of the exfiltration capabilities is removed - 67.35% for the model of size 5×3, and 26.91% for the model of size 5×5.

6.5 Attack settings and optimal defense strategies

This section provides a summary of the results in order to derive general recommendations based on the presented experiments. It can be concluded that, in the white-box scenario, as the LSB Encoding attack leads to 100% similarity, the adversary will focus on maximizing the exfiltration capacity without compromising the model performance on the original task. As shown above, replacing the significand (the 23 LSBs of a single-precision floating-point value) fulfills these requirements. It allows the adversary to still safely exfiltrate the training data with the largest possible capacity before the performance of the model decreases notably. Table 6.5 shows the results of the experiments at $b_a = 23$. We can observe that the largest possible decrease in accuracy among configurations and

Table 6.5: LSB Encoding attack: Change in model accuracy on the original task at $b_a = b_d = 23$, with red color denoting the worst observed scenario

Transformation	**Δ B vs. A Accuracy** Mean	Min	Max	Std
One-hot	-0.20	0.07	-0.53	0.20
Label	-0.22	0.15	-0.99	0.26
Gzip	-0.27	0.15	-1.14	0.27
Transformation	**Δ B vs. D Accuracy** Mean	Min	Max	Std
One-hot	-0.43	-0.03	-1.70	0.55
Label	-0.27	0.03	-1.70	0.36
Gzip	-0.43	-0.03	-1.70	0.53

model sizes tested is 1.14% at this point, in some cases, the performance of the model even increases, even by up to 0.15%.

Therefore, we recommend the defender apply a defense by LSB sanitization to remove the exfiltration functionality from the model, thereby removing the hidden data. Although modifying less than 23 bits leads to a large decrease in the data exfiltration functionality, Table 6.5 shows, that removing the hidden information from the 23 LSBs decreases a model's accuracy at most by 1.7%.

Regarding the black-box setting, it can be reasonably assumed that the attacker will aim to exfiltrate lower amounts of training data, as the explored models do not have the learning capacity to learn higher amounts of trigger samples. The uniformly distributed trigger samples led to the best results of the attack, with the in-distribution data providing another, although less successful, possibility of training data exfiltration. We have shown that targeted neuron pruning is a successful defense strategy against this type of attack. It greatly decreases the similarity of exfiltrated data to the original training data, even when small amounts of neurons are pruned while having minimal impact on the model performance on the original task. Furthermore, it can be applied universally in the explored settings, as it does not lead to major decrease of performance, even when applied on a model that does not contain any hidden data. Therefore the presented pruning method is recommended in order to mitigate the chance of data exfiltration.

6.6 Comparison to the State-of-the-Art

This section provides a comparison of the obtained results to the results of the foundational paper on data exfiltration attacks [63]. In this paper, the authors have used image and text data, namely the CIFAR, Labeled Faces in the Wild (LFW) and FaceScrub datasets (consisting of images), and 20Newsgroups and IMDB datasets (text data). They investigate the behavior of these attacks on convolutional neural networks for image data, and Support Vector Machines along with Logistic Regression for classification of text data. Although the results are not directly comparable, as we have explored further application scenarios of the attacks (on other types of neural networks and other data type), we can look at the relative attack utility in terms of capacity and impact on the model effectiveness on the original task. In the comparison, we focus on the results of the experiments performed on neural networks employed for binary classification in the underlying study (mainly CNN on LFW dataset).

6.6.1 Least Significant Bit Encoding Attack

In the LSB Encoding attack, the decoding is always perfect (unless encoding errors or defenses are present), therefore the achieved similarity is 100%. Their results show that the classification performance drops after using 18 LSBs of the parameters for data hiding on multi-class classification tasks, while for binary tasks up to 22 LSBs can be used. At this number of b_a, the classification performance is not substantially impacted, and the networks experience a loss of up to 0.14% in accuracy. This result is consistent with our findings.

For the classification on the LFW dataset (training test consisting of 9,924 images, each represented by $\sim 67,536$ bits), the authors used a convolutional neural network consisting of 880,000 parameters. They, however, use gzip compression before hiding the images in the model, and do not specify the number of images they can exfiltrate. Using a label-transformation (representing each pixel value by 8 bits, as pixel values consist of integers) they would be able to exfiltrate approximately 2.88% of the dataset, i.e. 286 instances, without harming the accuracy on the primary task. This number is likely higher for compressed data, though only marginally, as image data can not be compressed much further. For comparison, in the smallest investigated model (1,764 parameters) and label transformation, we are able to exfiltrate 0.32% of the dataset (i.e. 105 instances out of 32,561). Gzip transformation increases this to 2.35% of the dataset (i.e. 765 instances), without harming the models classification performance. Using an MLP comprising 177,736 parameters, we are able to exfiltrate 32.69% of the dataset (10,645 samples, using the label encoding) without compromising model's classification performance. In order to exfiltrate the full dataset (using the gzip transformation, with $b_a = 23$) we would require a model with $\sim 75,032$ parameters.

6.6.2 Black-Box Attack

For this type of attack, the authors investigated encoding three images (0.03% of the training data) using 34,000 trigger samples, and five images (0.05% of the training data) using 58,000 trigger samples (for the LFW dataset). The ratio of the trigger samples is therefore 3.4 and 5.8 respectively. The size of the CNN is the same as for the LSB attack (880,000 parameters). They used the Mean Absolute Pixel Error to measure the similarity of the exfiltrated images to the original images. The reconstruction similarity resulted in MAPE of 18.6 for three images, and 22.4 for five images. The observed impact on the classification performance was positive (up to +0.34% in accuracy). In comparison, a model comprising 35,794 parameters investigated within this thesis, enabled exfiltration of 0.02% of the training dataset (i.e. 8 instances) with perfect similarity. The observed behavior of the neural networks seems to be similar in both studies, however, a precise, direct comparison cannot be made due to the nature of the underlying data, and the differences in model architectures necessary to learn the different classification tasks.

CHAPTER 7

Conclusion

This thesis expands upon the topic of exploitation of the capacity of neural networks and has shown the design of successful data exfiltration attacks and defenses in two different scenarios utilizing neural networks. The networks have been trained on tabular data. The training data has been exfiltrated by first hiding it in a model using a steganography approach and subsequently retrieving it from a published model. The attacks are classified by the mode of access the adversary assumes to have to the final, trained model: white-box, or black-box access.

7.1 Summary & Contributions

This section describes the contributions of this thesis, as compared to the underlying study [63]. The behavior of the Least Significant Bit Encoding attack in the white-box scenario, and the behavior of the black-box scenario attack were investigated in models trained on tabular data, in models of various sizes. For the LSB Encoding attack, three different types of data transformation for encoding into the LSBs of a model were explored and the behavior of the attack was compared. Furthermore, an active defense by sanitizing the LSBs was implemented. Its utility, i.e. its effectiveness at removing the hidden information, along with the impact on the utility of the model on the original task were investigated. A similarity metric was defined in order to quantify the difference between the original training data and the exfiltrated data. To increase the robustness of the LSB Encoding attack, thereby making the hidden data more resistant to bit sanitization, error correction codes were applied as an additional step of the data transformation before encoding into LSBs. Their impact on the data similarity of exfiltrated data after a defense has been applied was observed.

In the black-box scenario attack, three different modes of trigger set generation and their impact on the similarity of exfiltrated data, as well as the model utility, were explored. Additionally, the impact of the size of the trigger set, as well as simple oversampling (i.e. oversampling by repetition) have been explored in the same context. An active defense against this type of exfiltration attack has been tested, by applying targeted, local pruning to the trained model. Again, its effect on the similarity of exfiltrated data, as well as the model effectiveness on the target task have been measured and analyzed in all configurations listed above.

Our contributions are as follows:

- We have introduced a taxonomy of data exfiltration attacks.
- The threat resulting from third-party code employment has been demonstrated by showing that ML models trained on tabular data can be successfully exploited for data exfiltration.
- We have shown that the adversary requires minimal knowledge about the training data properties in order to perform successful exfiltration in both scenarios.
- We have shown that models proportionate in size to the original task can be exploited for data exfiltration.

Further, we present our contributions as a response to the research questions defined in Section 1.3.

1. In what way can successful data exfiltration attacks using machine learning models be designed?
    - In order to measure the success of the attacks, we have defined a similarity metric in order to compare the exfiltrated data to the original data.
    - White-box scenario: We have presented a design of a data exfiltration attack (based on [63]) abusing the representation capacity of the model parameters by encoding training data into the least significant bits of the parameters of a model trained on tabular data. Three different modes of data transformation for encoding into the parameters have been investigated. This attack leads to a 100% similarity of exfiltrated data to the original training data.
    - Black-box scenario: In order to exfiltrate data in this scenario, a model is trained on crafted trigger samples labeled with bits of training data. We have adapted an attack abusing the learning capacity of the model, proposed by [63], for use with tabular data. Additionally, we have investigated three ways to craft the trigger set, establishing that uniformly distributed trigger samples produce a successful attack leading to a similarity of 100%. Moreover, we have presented a malicious training algorithm that forces a model to learn these trigger samples.
2. To what extent are the attacks useful?

a) To what extent can data be exfiltrated from trained machine learning models by performing attacks utilizing data hiding techniques?

- White-box scenario: As mentioned above, this attack leads to a 100% similarity of exfiltrated data to the original data. The amount of data exfiltrated depends on the size of the model and the number of least significant bits used for an attack. The introduced modes of data transformation allow to reduce the number of bits needed to represent a data sample, increasing the amount of data exfiltrated. In our experiments, we have been able to exfiltrate between 2.35% of the training set (using the smallest investigated model) up to 100%, i.e. the full training set utilizing larger models, with a minimal impact on model performance.
- Black-box scenario: In the black-box scenario, we have been able to exfiltrate 8 samples (0.02%) of the training set with a 100% similarity. Larger amounts of training data can be exfiltrated, but with lower similarity.

b) To what extent does the model effectiveness change after incorporating data exfiltration functionality?

- White-box scenario: Configurations of successful attacks that do not impact the model performance on the original task have been determined. Generally, we have found that even replacing the total significand of a model parameter represented as a single-precision floating-point value does not lead to a notable degradation of performance.
- Black-box scenario: We have found that modifying the training process to incorporate the learning of trigger samples slightly decreases the accuracy of the models compared to base models (trained with a benign algorithm). We have also determined that the performance of the smaller models is more negatively affected if the trigger samples generated follow the distribution of the original training data. This impact is less pronounced when training larger models. Uniformly distributed trigger samples, which also lead to the highest similarity score, have a smaller impact on the model performance. The degradation of the model performance is however largely attributed to the overtraining needed in order for the underlying model to learn the trigger samples.

3. In what way can models used for information hiding be successfully defended from data exfiltration attacks?

- White-box scenario: The information hidden by the attacks into the model parameters has been removed by sanitizing the LSBs of parameters.
- Black-box scenario: The design of the defense against the black-box attack was inspired by the design of attacks that aim to remove a watermark from a model. It targets the part of the neural network responsible for learning the trigger samples and removes them. It is performed by observing the network's

behavior on benign data and removing the neurons that do not get activated at a pass of this original data.

4. To what extent are the defense strategies useful?

   a) To what extent do the chosen defenses decrease the success of the data exfiltration attacks?

      - White-box scenario: The interplay between the number of bits used for attack, and the number of bits used for defense has been investigated and the similarity remaining after a defense has been observed. The nature of data transformation determines the rate of the deterioration of the similarity of exfiltrated data with the increasing amount of sanitized LSBs, i.e. the similarity of hidden data utilizing label encoding transformation decreases slower than that of gzipped data. Similarly, as in the attack setting, most of the bits of each parameter (the significand) can be modified without causing a notable decrease in model performance. In the likely attack scenarios, we are therefore able to remove all of the hidden information, thus preventing exfiltration. We have also shown that in some cases, the number of defense bits does not need to exceed the number of attack bits (although this information is not available to either party) for the defense to be successful.
      - Black-box scenario: By applying targeted, local neuron pruning, we are able to target for removal of the neurons that are responsible for the learned representation of the trigger samples. We have found that this defense removes the learned trigger samples from the smaller models at a faster rate, and causes a sharp drop in the similarity of exfiltrated data already after removing 10% of the neurons. For the larger models, the drop in similarity is also significant, and removes a large amount of learned information, however, at a slightly slower rate. In these models, we are however able to prune more neurons without compromising the model's performance on the original task.

   b) To what extent does the model utility change after the application of defense strategies against data exfiltration attacks?

      - White-Box scenario: As already partially answered above, the application of this defense, depending on the number of bits used for the attack and for the defense, does not affect the performance of the model, or impacts it minimally. We reasonably assume the adversary's motivation to not compromise the model's performance on the original task. Therefore, we focus our result analysis mainly on the scenarios where the attacker does not utilize the full capacity of each individual model parameter. In this case, we are able to remove the exfiltration functionality, while not harming the model's performance too much. We have found that this defense can also be applied to benign models without (notably) impacting their performance.

- Black-box scenario: We have found that the similarity of exfiltrated data to the original data decreases at a much faster rate with activation-based neuron pruning than the performance of the models decreases on the original task. Therefore, the amount of pruning required to remove the learned representation of the trigger samples from a trained model does not cause a notable decrease in model performance. In the models most successful at exfiltration, when pruning only 10% of the neurons, we have found that the accuracy decreases at most by 0.35% in comparison to a base model, and in some cases, even increases. A possible explanation for this increase is that the neurons responsible mainly for learning the trigger samples, contribute negatively to the predictions on the original data, thus when removed, the performance on the original data improves. When pruning a larger number of neurons, the trade-off between similarity and model utility is more considerable, however, the worst noted accuracy loss in the successful models was 2.84%. We have found that applying the defense to corresponding base models decreases their performance on the original task at a slower rate, which points to the low cost of defense deployment on the defender's side.

Additionally, we present our further contributions:

- We have added an adaptation to increase the robustness of the LSB Encoding attack by incorporating error correction codes. We showed the settings where this adaptation improves the success of an attack leading to exfiltration with 100% similarity.
- We found defense settings that handle this adaptation by the attacker and safeguard the data from exfiltration.

7.2 Future Work

The future work in this field lies in exploring further attack settings and scenarios. The influence of the model hyperparameters on the attacks and defenses was out of the scope of this work. However, especially for the attack in the black-box scenario, the model hyperparameters, such as learning rate, dropout, or possibly other activation functions could potentially impact how quickly and up to what value of similarity the trigger set is learned. The behavior of the white-box attacks introduced in Chapter 3, namely Correlated Value Encoding Attack and Sign Encoding Attack, has not yet been explored in models built on tabular data. For the LSB Encoding attack, we have found that the significand of the bit representation of model parameters of a single-precision floating-point value can be replaced causing minimal impacts on the effectiveness of the model. Therefore, in order to increase the capacity of the model for the attacks, parameters of double-precision floating-point values can be utilized to potentially hide more data.

To our best knowledge, defenses have not yet been developed for the Correlated Value Encoding Attack and Sign Encoding Attack, and could therefore constitute a further research direction. Applying the developed defenses on models built on other data types, such as images or text, was not explored within this work. We believe that these defenses could potentially be successful on other types of neural networks (e.g. convolutional neural networks).

Exploration of further modifications of defense strategies could lead to improvements of defense utility.

Since the significand of each parameter value does not contribute largely to making correct predictions, representing the parameters by fewer bits, or possibly binarizing the weights can be explored in this scenario. This way, the capacity to hide data would be expected to be largely eliminated, and any data-hiding attempts would potentially result in greater performance losses on the original task.

In order to increase the robustness of the attack in the black-box scenario, pre-pruning of the model by the adversary could be implemented, as an adaptation of the pruning-aware attack for backdoors [44]. This would consist of the adversary pruning the neurons least active upon the pass of the training data, and then training the pruned network with both training and trigger data. This results in the trigger samples activating the same neurons. The attacker can therefore prune until a certain loss in model effectiveness on the original task is reached, while still preserving effectiveness on the trigger samples. Upon applying a defense, this could potentially lead to a sharper and faster decrease in the performance of the model on the original task, possibly causing the defender to not be able to apply a defense without compromising the model's performance. However, in that case, the risk for the attacker is that the final model will not be deployed (i.e. the attacker will not gain access to the prediction API needed to exfiltrate the data).

Furthermore, other defense methods such as fine-tuning or fine pruning can be applied. Weight pruning (i.e. setting selected values of the model parameters values to 0) is another possible approach, which could potentially defend against attacks in both scenarios. As this approach does not involve the computation of neuron activations, and it is thus likely a simpler approach for the defender to apply, requiring less knowledge. However, it might produce a less effective defense. Another possible defense is knowledge distillation, i.e. transferring of the learned knowledge from a complex model to a simpler model could potentially remove the hidden information, while preserving the prediction behavior on the original task. The discussed defense strategies, namely pruning, quantization and distillation could potentially improve model efficiency w.r.t. prediction time.

Moreover, other similarity metrics for measuring the similarity of two samples of tabular data could be utilized in order to measure the similarity of the exfiltrated data to the original data. For example, taking into account the ranges of values and imposing a larger penalty on values outside of a defined range could be implemented, although such calculation of similarity would be dataset and use-case specific.

The behavior of data exfiltration attacks and the corresponding defenses within this thesis need to be explored on a wider range of datasets, various tasks such as multi-class or multi-label classification, or regression problems. Furthermore, their behavior can be observed on various additional data types (e.g. audio, time-series), as, so far, they have been explored only on image, text, and tabular data.

The behavior and capacity of further network types should be investigated (e.g. graph neural networks, recurrent neural networks, ...).

Intentional memorization offers another way to force a model to 'remember' certain instances. Usually utilizing models' ability to overfit, this adversarial technique is closely related to the learning of the trigger samples in the black-box scenario, however, utilizing different techniques to extract the wanted sample. The relation of this technique to the steganography-based approaches in this thesis should be further explored.

Additionally, as the influence of an adversary on the training process is not always presumed, more work in the field of exfiltration regarding unintentional memorization occurring in neural networks and subsequent data exfiltration should be explored.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] O. F. AbdelWahab, A. I. Hussein, H. F. A. Hamed, H. M. Kelash, A. A. M. Khalaf, and H. M. Ali. Hiding data in images using steganography techniques with compression algorithms. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 17(3):1168–1175, June 2019. Number: 3.

[2] Y. Adi, C. Baum, M. Cisse, J. Keshet, and B. Pinkas. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *Proceedings of the 27th USENIX Security Symposium*, 2018.

[3] R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal*, 12(2):157–169, Aug. 2003.

[4] R. Agrawal and J. Kiernan. Chapter 15 - Watermarking Relational Databases. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 155–166. Morgan Kaufmann, San Francisco, Jan. 2002.

[5] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, Nov. 2010.

[6] B. Bashari Rad, M. Masrom, and S. Ibrahim. Camouflage in Malware: from Encryption to Metamorphism. *International Journal of Computer Science And Network Security (IJCSNS)*, 12:74–83, 2012.

[7] P. Bassia, I. Pitas, and N. Nikolaidis. Robust audio watermarking in the time domain. *IEEE Transactions on Multimedia*, 3(2):232–241, June 2001.

[8] L. Biewald. Experiment Tracking with Weights and Biases, 2020. Software available from wandb.com.

[9] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion Attacks against Machine Learning at Test Time. In *Machine Learning and Knowledge Discovery in Databases*, pages 387–402, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[10] B. Biggio and F. Roli. Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and*

Communications Security, CCS '18, pages 2154–2156, New York, NY, USA, Oct. 2018. Association for Computing Machinery.

[11] F. Boenisch. A Systematic Review on Model Watermarking for Neural Networks. *Frontiers in Big Data*, 4, 2021.

[12] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander. Towards Federated Learning at Scale: System Design. In *SysML 2019*, volume 1, pages 374–388, 2019.

[13] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander. The New Threats of Information Hiding: The Road Ahead. *IT Professional*, 20(3):31–39, May 2018. Conference Name: IT Professional.

[14] N. Carlini, C. Liu, \. Erlingsson, J. Kos, and D. Song. The secret sharer: evaluating and testing unintended memorization in neural networks. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, pages 267–284, USA, Aug. 2019. USENIX Association.

[15] A. Chakraborty, A. Mondai, and A. Srivastava. Hardware-assisted intellectual property protection of deep learning models. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[16] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 105–113, Ottawa ON Canada, June 2019. ACM.

[17] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning, Dec. 2017. arXiv:1712.05526 [cs].

[18] I. Cox, J. Kilian, F. Leighton, and T. Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687, Dec. 1997. Conference Name: IEEE Transactions on Image Processing.

[19] B. Darvish Rouhani, H. Chen, and F. Koushanfar. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 485–497, New York, NY, USA, Apr. 2019. Association for Computing Machinery.

[20] T. Donker. The dangers of using large language models for peer review. *The Lancet Infectious Diseases*, 0(0), May 2023. Publisher: Elsevier.

[21] C. J. D'Orazio, K.-K. R. Choo, and L. T. Yang. Data Exfiltration From Internet of Things Devices: iOS Devices as Case Studies. *IEEE Internet of Things Journal*, 4(2):524–535, Apr. 2017.

[22] M. Fredrikson, S. Jha, and T. Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333, New York, NY, USA, Oct. 2015. Association for Computing Machinery.

[23] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 619–633, New York, NY, USA, Oct. 2018. Association for Computing Machinery.

[24] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing Deep Convolutional Networks using Vector Quantization, Dec. 2014. arXiv:1412.6115 [cs].

[25] T. Gu, B. Dolan-Gavitt, and S. Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain, Mar. 2019. arXiv:1708.06733 [cs].

[26] W. Gu. Watermark Removal Scheme Based on Neural Network Model Pruning. In *Proceedings of the 2022 5th International Conference on Machine Learning and Natural Language Processing*, MLNLP '22, pages 377–382, New York, NY, USA, Mar. 2023. Association for Computing Machinery.

[27] A. N. Gulati and S. D. Sawarkar. A new steganography approach for image encryption exchange by using the least significant bit insertion. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, pages 998–998, Mumbai Maharashtra India, Feb. 2010. ACM.

[28] S. Gupta, A. Goyal, and B. Bhushan. Information hiding using least significant bit steganography and cryptography. *International Journal of Modern Education and Computer Science*, 4(6):27, 2012. ISBN: 2075-0161 Publisher: Modern Education and Computer Science Press.

[29] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[30] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig. Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures Using High-Density SNP Genotyping Microarrays. *PLOS Genetics*, 4(8):e1000167, Aug. 2008. Publisher: Public Library of Science.

[31] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures, July 2016. arXiv:1607.03250 [cs].

[32] G. Hua, J. Huang, Y. Q. Shi, J. Goh, and V. L. Thing. Twenty years of digital audio watermarking—a comprehensive review. *Signal Processing*, 128:222–242, Nov. 2016.

[33] IEEE. IEEE Standard for Floating-Point Arithmetic, July 2019. Conference Name: IEEE Std 754-2019 (Revision of IEEE 754-2008).

[34] B. Jayaraman, L. Wang, K. Knipmeyer, Q. Gu, and D. Evans. Revisiting Membership Inference Under Realistic Assumptions. *Proceedings on Privacy Enhancing Technologies*, 2021(2):348–368, Apr. 2021.

[35] H. Jia, C. A. Choquette-Choo, V. Chandrasekaran, and N. Papernot. Entangled Watermarks as a Defense against Model Extraction. In *Proceedings of the 30th USENIX Security Symposium*, 2021.

[36] D.-S. Jung, S.-J. Lee, and I.-C. Euom. ImageDetox: Method for the Neutralization of Malicious Code Hidden in Image Files. *Symmetry*, 12(10):1621, Oct. 2020. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute.

[37] N. S. Kamaruddin, A. Kamsin, L. Y. Por, and H. Rahman. A Review of Text Watermarking: Theory, Methods, and Applications. *IEEE Access*, 6:8011–8028, 2018.

[38] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A Watermark for Large Language Models, Jan. 2023. arXiv:2301.10226 [cs].

[39] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, Department of Computer Science, University of Durham, Durham, UK, 2007.

[40] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated Learning: Strategies for Improving Communication Efficiency, Oct. 2017. arXiv:1610.05492 [cs].

[41] E. Le Merrer, P. Pérez, and G. Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32(13):9233–9244, July 2020.

[42] I. Lederer, R. Mayer, and A. Rauber. Identifying Appropriate Intellectual Property Protection Mechanisms for Machine Learning Models: A Systematization of Watermarking, Fingerprinting, Model Access, and Attacks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–19, 2023. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[43] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural Networks with Few Multiplications. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[44] K. Liu, B. Dolan-Gavitt, and S. Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks, May 2018. arXiv:1805.12185 [cs].

[45] S. Lounici, M. Njeh, O. Ermis, M. Onen, and S. Trabelsi. Yes We can: Watermarking Machine Learning Models beyond Classification. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–14, Dubrovnik, Croatia, June 2021. IEEE.

[46] T. Morkel, J. H. P. Eloff, and M. S. Olivier. An Overview Of Image Steganography. In *Proceedings of the ISSA 2005 New Knowledge Today Conference, 29 June - 1 July 2005*, 2005.

[47] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization, Aug. 2017. arXiv:1708.08689 [cs].

[48] R. Namba and J. Sakuma. Robust Watermarking of Neural Network with Exponential Weighting. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 228–240, Auckland, July 2019.

[49] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer series in operations research. Springer, New York, 2nd ed edition, 2006. OCLC: ocm68629100.

[50] D. Oliynyk, R. Mayer, and A. Rauber. I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences. *ACM Computing Surveys*, page 3595292, Apr. 2023. arXiv:2206.08451 [cs].

[51] L. Y. Por and B. Delina. Information Hiding: A New Approach in Text Steganography. In *Proceedings of the 7th WSEAS International Conference on Applied Computer & Applied Computational Science (ACACOS '08),April 6-8, 2008*, Hangzhou, China, 2008.

[52] V. Potdar, S. Han, and E. Chang. A survey of digital image watermarking techniques. In *INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, pages 709–716, Aug. 2005. ISSN: 2378-363X.

[53] J. Poulos and R. Valle. Missing Data Imputation for Supervised Learning. *Applied Artificial Intelligence*, 32(2):186–196, Apr. 2018. arXiv:1610.09075 [cs, stat].

[54] E. Quiring, D. Arp, and K. Rieck. Forgotten Siblings: Unifying Attacks on Machine Learning and Digital Watermarking. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 488–502, Apr. 2018.

[55] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 525–542, Cham, 2016. Springer International Publishing.

[56] F. Regazzoni, P. Palmieri, F. Smailbegovic, R. Cammarota, and I. Polian. Protecting artificial intelligence IPs: a survey of watermarking and fingerprinting for machine learning. *CAAI Transactions on Intelligence Technology*, 6(2):180–191, 2021. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12029.

[57] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[58] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. D. Tygar. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, IMC '09, pages 1–14, New York, NY, USA, Nov. 2009. Association for Computing Machinery.

[59] P. Sallee. Model-Based Steganography. In *Digital Watermarking*, Lecture Notes in Computer Science, pages 154–167, Berlin, Heidelberg, 2004. Springer.

[60] S. Samonas and D. Coss. The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security. *Journal of Information System Security*, 10(3):21–45, 2014.

[61] Scikit-Learn. Common pitfalls and recommended practices — Version 0.10.1, 2022.

[62] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, May 2017. ISSN: 2375-1207.

[63] C. Song, T. Ristenpart, and V. Shmatikov. Machine Learning Models that Remember Too Much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 587–601, Dallas Texas USA, Oct. 2017. ACM.

[64] G. Suarez-Tangil, J. Tapiador, and P. Peris-Lopez. Stegomalware: Playing Hide and Seek with Malicious Components in Smartphone Apps. In *Information Security and Cryptology: 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, Dec. 2014.

[65] A. Tchernykh, U. Schwiegelsohn, E.-g. Talbi, and M. Babenko. Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability. *Journal of Computational Science*, 36:100581, Sept. 2019.

[66] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing Machine Learning Models via Prediction APIs. In *USENIX security symposium*, volume 16, pages 601–618, 2016.

[67] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 269–277, June 2017. arXiv:1701.04082 [cs].

[68] R. Van Antwerp. *Exfiltration techniques: an examination and emulation.* PhD thesis, University of Delaware, 2011.

[69] L. Weidinger, J. Mellor, M. Rauh, C. Griffin, J. Uesato, P.-S. Huang, M. Cheng, M. Glaese, B. Balle, A. Kasirzadeh, Z. Kenton, S. Brown, W. Hawkins, T. Stepleton, C. Biles, A. Birhane, J. Haas, L. Rimell, L. A. Hendricks, W. Isaac, S. Legassick, G. Irving, and I. Gabriel. Ethical and social risks of harm from Language Models, Dec. 2021. arXiv:2112.04359 [cs].

[70] R. Wirth and J. Hipp. CRISP-DM: Towards a Standard Process Model for Data Mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1, 2000.

[71] M. Wu, W. Trappe, Z. Wang, and K. R. Liu. Collusion-resistant fingerprinting for multimedia. *IEEE Signal Processing Magazine*, 21(2):15–27, Mar. 2004. Conference Name: IEEE Signal Processing Magazine.

[72] H. Xu, Y. Su, Z. Zhao, Y. Zhou, M. R. Lyu, and I. King. DeepObfuscation: Securing the Structure of Convolutional Neural Networks via Knowledge Distillation. *ArXiv*, June 2018.

[73] C. Yang, R. A. Brower-Sinning, G. Lewis, and C. Kästner. Data Leakage in Notebooks: Static Detection and Better Processes. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, Rochester MI USA, Oct. 2022. ACM.

[74] Z. Yang, H. Dang, and E.-C. Chang. Effectiveness of Distillation Attack and Countermeasure on Neural Network Watermarking, June 2019. arXiv:1906.06046 [cs].

[75] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282, Oxford, July 2018. IEEE.

[76] I. You and K. Yim. Malware Obfuscation Techniques: A Brief Survey. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300, Nov. 2010.

[77] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 159–172, New York, NY, USA, May 2018. Association for Computing Machinery.

[78] M. H. Zhu and S. Gupta. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In *Proceedings of the 6th International Conference on Learning Representations, {ICLR} 2018*, Vancouver, Canada, 2018.