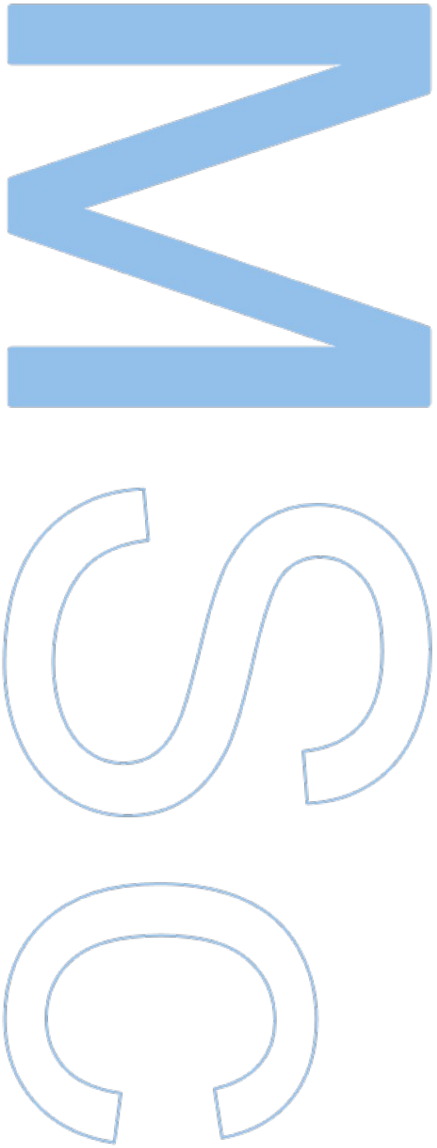
Modelling Attacks on Privacy-Preserving Machine Learning



Gabriel Vianna Saraiva
Masters in Information Security
Department of Computer Science
2023

Supervisor
Bernardo Luís Fernandes Portela
Professor Auxiliar
Faculdade de Ciências da Universidade do Porto


U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Abstract

The utilization of machine learning is becoming more and more prevalent, and this upward trend has no stop in sight. However, as this technology spreads to a new set of areas, its vulnerabilities also keep expanding. As this technology is now being utilized in more critical situations, such as predictive policing [19], not only can more vulnerabilities be exploited, but the impact of such exploits can be larger than ever, and even directly affect ordinary citizens.

The problem of security in machine learning is multi-faceted. This problem can refer to the capture/generation of data, training the models, inferences, or even the improper usage of the models themselves. Each step of the utilization of machine learning can be targeted by different attacks, with different objectives, making it essential to establish a proper common ground for discussing this topic.

The purpose of this work is to categorize the attacks on machine learning and explore a rigorous approach to designing robust systems against well-defined machine learning attacks. This was done by analyzing the already existing attacks in the literature and grouping them by requirements, entry points, and objectives into well-defined categories presented in this work. These subcategories can then be translated into rigorous security experiments, as exemplified, that can be utilized to evaluate a system's vulnerability to the specified attack.

Keywords: Security, Data, Requirements, Objectives, Models, Training

Resumo

A utilização de machine learning está se tornando cada vez mais prevalente, e essa tendência não mostra sinais de desaceleração. No entanto, à medida que essa tecnologia se espalha para novas áreas, suas vulnerabilidades também continuam se expandindo. À medida que essa tecnologia é agora utilizada em situações mais críticas, como policiamento preditivo [19], não apenas mais vulnerabilidades podem ser exploradas, mas o impacto de tais explorações pode ser maior do que nunca, e até mesmo afetar diretamente cidadãos comuns.

O problema da segurança em machine learning é multifacetado. Esse problema pode se referir à captura/geração de dados, ao treinamento dos modelos, às inferências ou até mesmo ao uso impróprio dos próprios modelos. Cada etapa da utilização de machine learning pode ser alvo de diferentes ataques, com diferentes objetivos, tornando essencial o estabelecimento de uma base comum de conhecimento adequada para discutir esse tópico.

O propósito deste trabalho é categorizar os ataques à machine learning e explorar uma abordagem rigorosa para projetar sistemas robustos contra ataques ao machine learning bem definidos. Isso foi feito através da análise dos ataques já existentes na literatura, agrupando-os por requisitos, pontos de entrada e objetivos, em categorias bem definidas, apresentadas por este trabalho. Essas subcategorias podem então ser traduzidas em experimentos de segurança rigorosos, como exemplificado, que podem ser utilizados para avaliar a vulnerabilidade de um sistema ao ataque especificado.

Palavras-chave: Security, Data, Requirements, Objectives, Models, Training

Contents

List of Tables

List of Figures

Listings

Acronyms

AI Artificial Intelligence

AWS Amazon Web Services

DCC Departamento de Ciência de Computadores

FCUP Faculdade de Ciências da Universidade do Porto

IP Internet Protocol

ML Machine Learning

TCP Transmission Control Protocol

Chapter 1

Introduction

1.1 Context

Machine Learning has been increasingly utilized in all sorts of critical situations that can directly affect society. Some of these situations are predictive policing [19], which can be summarized as the utilization of machine learning for predicting whether an individual has a high probability of committing a crime or not. Another situation is disease diagnostic and treatment [23], where this technology is utilized to aid in the early detection of fatal diseases, speeding up the start of treatment and increasing the survival rate of patients. Due to this spread utilization of this technology, it needs to be secure to guarantee that it performs its task correctly.

This motivates the European Ethics Guidelines for Trustworthy AI[10], that even though is constituted by seven points, only two mainly correlate with this research: Privacy and Data Governance and Technical Robustness and Safety. The first point states that all data present in Artificial Intelligence (AI) systems have to respect privacy and data protection, which is commonly the target of attacks in machine learning systems, such as the Membership Inference attacks, where the objective is to infer whether an individual entry was or was not part of the training dataset for a machine learning model, threatening the privacy of individuals, depending on the information present in the training dataset. The second point affirms that AI has to be secure, but this is a very broad term that can mean many different things. For example, a very common attack in Machine Learning (ML) is Data Poisoning, which has nothing to do with the concept of explainability in AI, which is a part of the Transparency topic present in the ethics guidelines previously mentioned, but both of them fall into the general term of

"Secure AI". Another example of this is when talking about attacks that envision provoking misclassifications by the model, such as Data Poisoning and Adversarial Samples. When these attacks are successful, how can we determine if the fault is in the attackers themselves or if the model is too prone to perturbations, is not well defined.

Defining and reasoning about security in the context of machine learning is a very complex issue, as seen above. This is partly due to many different approaches and vectors that need security and the existence of a variety of terms that are utilized interchangeably, sometimes even with different meanings, by different authors, companies, and projects. These terms are essential to the understanding of which aspect of Machine Learning Security we are talking about since this whole discussion is very dependent on context, more specifically where in the pipeline of Machine Learning is the attack occurring, and what's its objective. Talking about security in a rigorous way requires very concrete definitions of said security. Due to the importance of this aspect, a parallel to how this is handled when discussing security in a cryptographic environment was drawn.

1.2 Objectives

With these concerns in mind, the work presented here is divided into two parts: categorizing the attacks present in the literature in a more defined way, and defining security experiments (security games) that allow the evaluation of security against a specific attack vector. As expected, there are countless attack vectors, so this work will be focused on a subset of those, constituted of well-behaved attacks.

The first part is a more defined outline of the many terms already existent in literature, as a means to facilitate the discussion around the subject. In other words, as the objective of this work is to draw concrete models to describe attacks, the first step is to determine definitions that are as less ambiguous as possible. For example, the terms "Adversarial Attacks" or "Adversarial Samples Attack" can be found in the literature meaning different types of attacks, that, within this paper, will be later separated in the terms "Data Poisoning" and "Adversarial Samples Attacks", to make sure such different methods of performing an attack on a machine learning environment and system becomes clearly distinct. So, later this will allow rigorous characterization and differentiation between these attacks.

The second part has the objective of defining such categories as games, to try and understand

better what actions, information and access is needed by the attacker to be "successful" in said categories. These games are very general descriptions, that try to quantify security with the help of game theory, very similar to the games described in Cryptography, which are security experiments – pseudo-code – that define precise rules for qualifying security guarantees that the systems want to provide, and against which adversaries. Generally, the game is similar to a template for an attack, where different attacks of the same type can be inserted to verify their effectiveness against predefined parameters. The objective of the games defined in this work was to be very non-specific such that the largest possible quantity of already existing (and also future) attacks can be represented, as a means to qualify their correctness and appropriate framing of existing research. The more general, and less complex the game turned out to be, the more the "term/category" the game represents is deemed well-defined. Later, real-world attacks can be represented by these games, by fitting their specific details into them.

Due to the non-existence of consensus when it comes to the utilization and coinage of new terms, the devised way in this work to approach the categorization of the terms was to separate them into categories, taking into account their objectives, where the attack occurs in the pipeline and also the required information and access needed to perform such attack. The creation of these more general categories was not easy because many of the already existing terms were so similar and had a large portion of meaning and usage overlap. Even though a hard cutoff had to be done, with the objective to make it easier to understand and to talk about, and as such, it may present itself as a necessary re-adaptation and relearning of some of the most utilized and popular terms, to adapt to the guideline described here.

This work concludes with a practical example of a ML attack, and an instantiation within the proposed experiments. This validates the proposed experiments as something that can be instantiated with a real attack, as well as expanded with adequate countermeasures.

1.3 Organization

The rest of this paper is divided into five more chapters. In Chapter 2, a more general view of the concepts mentioned in this paper is given. Chapter 3 is about the categories of attacks defined by this work and how they correlate to the existing literature. Chapter 4 demonstrates how those categories can be transformed into security experiments or games. Chapter 5 contains an experimental attack and describes how it can be fit into the games described in the previous

chapter. And finally, Chapter 6 is the conclusion of the research and where it's left off for future work to be done.

Chapter 2

Background

This chapter describes all important technologies and terminology that are needed to understand the work written on this paper, as was specified in the Introduction Section.

2.1 Machine Learning

Machine learning [15, 36] is the usage of computer systems and solutions that do not require direct human interaction or explicit instructions and try to draw inferences from patterns in data, in order to approximate a function that isn't known. Simply put, it's a solution for processing large amounts of data and drawing conclusions from understanding and finding patterns in it. It does try to mimic the way humans learn, gradually improving results, based on predetermined metrics.

A machine learning implementation and usage can usually be visually represented by a diagram that shows its pipeline. Here I'll introduce a very simple and generic machine learning pipeline so it becomes clearer which steps are usually present, and in which order they're executed.

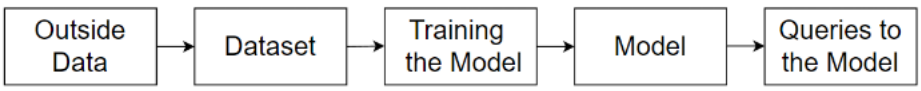


Figure 2.1: Simple machine learning Pipeline

As figure 2.1 shows, there are five main steps in a machine learning pipeline. The first step is to collect the data that will be processed. This can be done by many different means and have many different sources. After that, the data is then normalized and grouped into something

that's called a dataset. This dataset is the actual data that will be read by the machine learning system. This dataset is then processed utilizing algorithms and other mathematical operations, all dependent on the system, training the model. A model is then generated, that represents all the learning done from the dataset, represented by a program that can make decisions and classify patterns based on such dataset. This model is then made available to be utilized for the classification or regression of new data, applying what it knows on queries and giving a result.

For example, a machine learning model that tries to identify animals can be created by gathering different pictures of animals and organizing such pictures into a dataset, with the correct label for each (the species name in this case). The model is then trained on such data, generated and then people can submit other images for it to try and classify based on the patterns that it learnt from the pictures in the dataset, and outputting the label that it considers more correct for said picture.

The pipeline represented above is actually a simplification of a real machine learning pipeline and will be considered when discussing and defining attack categories due to simplicity reasons, facilitating the composition of attack categories without restricting the application to more realistic scenarios. Most recent applications of machine learning do not rely on such a step-by-step organization, it's more of a never-ending cycle. This means that all the steps are always being improved upon and expanded, in order to try and have the best possible model available at any given time, taking into account variations of data due to a shift in patterns in the real world. Outside data is always being collected, datasets are always being expanded and updated, the model gets retrained after certain intervals of time, to produce the more updated and accurate model available at the time, and as such, the queries to the model are better responded to.

There is also the possibility that the information that is to be used for training the dataset to be very sensitive and cannot be shared outside it's source [25], such as hospital records for example [9, 16]. The necessity of keeping local information private while also requiring the utilization of different sources of information for training the most accurate model seems counter-intuitive, but for said reason, a different method of machine learning was developed, called Federated Learning [47]. In this method of machine learning, different clients (sources/machines/users) work together to train the best possible model without sharing their private information with each other. This is done by utilizing a function of aggregation to join all the produced values from each clients training session into something that will train and create the unified model.

2.2 Information Availability Notation

When interacting with machine learning models and many other systems, the user interaction can be qualified as Black-box or White-box, depending on the amount of information displayed by the system [17, 28]. For example, a predictive model made available by Amazon Web Services (AWS) where the client can only give inputs and receive outputs from the model is considered Black-box, but if the client would have access to intern values of the inner workings of the model, that would be White-box.

This terminology is simple to understand when thinking about actual boxes where operations are performed, such as demonstrated by the figures 2.2 and 2.3, the Black-box does not reveal any additional information other than the input and the output, while the White-box makes visible which operations were performed in order to obtain its result.

Figure 2.2: Black-box example

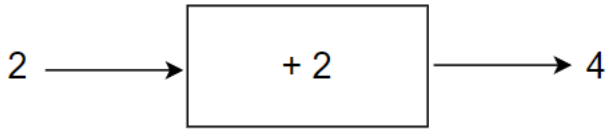


Figure 2.3: White-box example

2.3 Security Experiments

In the scope of cryptography, there is a core necessity of defining security in a concrete and quantifiable manner. For drawing the definitions, and also testing upon those same paradigms, modern approaches rely on experiments [41]. Those consist of an adversary against an experiment. The actions that the adversary can perform define the adversarial power, while the experiment itself, and its result, define the success of said adversary.

For example, one such game is called Indistinguishability under Chosen-Plaintext Attack,

or IND-CPA for short. In this experience, the security concept of Indistinguishability is what is being tested, and the attack type being performed is Chosen-Plaintext Attack. This game exemplifies a situation where the attacker gives the cryptographic system two different plain text messages of the same length, the system encrypts one of them, and then the attacker has to guess which message was encrypted. If the success rate of the attacker is only negligibly higher than the random pick success chance (50%), the cryptographic system is considered IND-CPA. Formally, the game [3] can be described as:

IND-CPA:

$b \leftarrow\$ \{0, 1\}$

$K \leftarrow\$ K()$

$m_0, m_1 \leftarrow\$ \mathcal{A}()$

$C \leftarrow \mathsf{Enc}(k, m_b)$

$b' \leftarrow\$ \mathcal{A}(C)$

Return $|m_0| == |m_1|$ AND $b' == b$

Figure 2.4: IND-CPA cryptographic security game.

This is a very rigorous and strict security game example but leaves some aspects as abstracts, such as Enc() and $\mathcal{A}()$. While both of these are abstract in the game description, Enc() is to be defined by the system to be tested, while $\mathcal{A}()$ is left abstract as a means to signify that such a system can be tested against any possible adversary, proving that the system is then secure against any $\mathcal{A}()$.

Security guarantees can be more or less strict, with some guarantees being composed of others. So, for the example granted, a system being IND-CPA also implies it being Indistinguishable under attacks where the opponent does not have the ability to choose the message to be encrypted or even knows it. Some security characteristics can also imply other ones, such as Non-Malleability implying Indistinguishability, as proven by [4].

The are many different applications for this style of defining security, such as the security experiment of proving unforgeability for a MAC (Message Authentication Code), which is another popular example. These games enable the “provable security” approach, where security can be demonstrated by setting a concrete security bound, and in turn, comparing the security of two very distinct systems.

The objective of this work is to utilize a similar approach for defining games for the attack

vectors here defined, prioritizing defining as concretely as possible two main aspects: The adversary power (what an adversary can do on a real system) and the victory condition (what does an attack have to do to be considered successful). An example of a victory condition can be how a model's performance has to be impacted by a specific attack.

Chapter 3

State of the Art

This chapter will present the most relevant information correlated to this work, obtained through the literature that already exists.

The main takeaway from the vast literature was the gathering of existing terms and their utilization, in order to lay a foundation for this work. First, a general view of the attack categories that were defined by this work will be presented, including their positioning on the machine learning pipeline, in regard to requirements, objectives and entry points. Second, I'll give the definition of each attack category and also specific subcategories, pointing out some examples for better understanding. Only then some works to represent the utilization of said terms will be cited, mostly in order to highlight the different usage of the same terminology throughout the literature, and also to shine a light on how the lines were drawn and categories defined.

3.1 Defining Attack Types

The first steps were collecting the most used terms and trying to understand what they meant generally. After reading many different sources, six categories of attacks were formed. These categories were mainly groupings of attacks that had similar objectives, requirements and points of entry in the machine learning pipeline. The ability of querying the model, and analysing the returned output is assumed to be available to every adversary situation, as most of the attacks require this ability to be performed, and the only one that does not can utilize it to evaluate the attack performance. These six categories are displayed together with their requirements, entry points and objectives on our simplified machine learning pipeline in Figure 3.1, and are as

follows, including our definition:

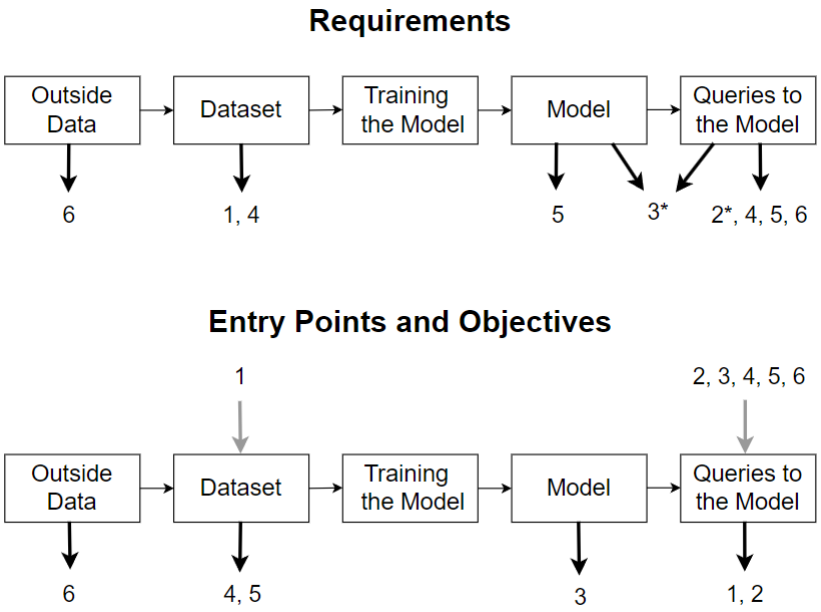


Figure 3.1: Indication of all attack categories requirements, entry points and objectives on the simplified pipeline

1. Data Poisoning: Manipulating the training dataset in order to impact the trained model. Requires a method of modifying/corrupting the training dataset, which also signals its entry point and has the objective of impacting the to-be-trained model performance, when queried.

2. Adversarial Samples: Crafting queries that generate a clear misclassification by the model. Requires different amounts of information returned by the model as a result of querying the model, which is its entry point. The objective is to craft specific queries that are misclassified by the model. The exception in requirements is the subcategory of Transfer-based attacks, which require access to a large portion of the training dataset.

3. Model Extraction: Obtaining a counterfeit copy of an existing model. The requirements for this category are the most varied, with most requiring the ability to query the model and/or some information about its inner workings. The entry point is also through the queries, while the objective is to obtain a counterfeit model, which is a copy of the target model, that performs the same function. The requirement exception is the attacks classified as Data-free Knowledge Distillation, which requires only publicly available information about the training dataset.

4. Model Inversion: Obtaining information pertaining to the training dataset by querying the model. Requires partial information from a subset of the training dataset and has the entry point of querying the model. The objective is to obtain some other information contained in the training dataset.

5. Reconstruction Attacks: Obtaining the original training dataset by querying the model. Requires information about the inner workings of the model, specifically feature vectors. The entry point is querying the model, and the objective is to reconstruct the raw original training dataset.

6. Membership Inference: Finding out if a specific entry was present in the training dataset. The objective is to find out if a certain data entry was present in the training dataset. These attacks require additional information pertaining to the target entry and the output of confidence values by the model when being queried. Querying the model is it's entry point.

Still referencing Figure 3.1, the entry points locations on the pipeline are represented by the grey arrows coming from the top, while the objectives are represented by the black arrows at the bottom.

In the following sections, we will present concrete definitions for each of the attacks already mentioned, which will later allow for definitions of security experiments. For each of them, the existing literature pertaining to the attack will also be mentioned, and comparisons between the definitions established and other previous works will be drawn, in order to support the definitions themselves.

3.1.1 Data Poisoning

We will define this category of attacks as follows: The adversary will first interact with the dataset, altering it according to specific security criteria. Its goal will be to make the model deviate by a specific value, which can be subdivided into concrete corruption goals: Fix, Backdoor, and Model breaking.

- Fix denotes the goal of fixing the prediction of a given label by a concrete value.
- Backdoor denotes the goal of forcing a specific query to deviate from the untampered model training.

- Model breaking denotes the goal of reducing the general efficacy of the model by a concrete value.

Additionally, we consider two scenarios for the attacks, in which the adversary can influence the input dataset: General Poisoning (GENP) and Federated Learning. The first considers a general setting, where the adversary can influence a threshold of the overall dataset. The second considers a federated learning setting, where the adversary can fully influence a threshold of participants in the Federated Learning Process.

Fix

The objective of these attacks is to try and fix a large quantity of the classification done by the model, to be a certain target label. For example, if considering a binary classification model, the target can be to make the model to classify the highest amount of samples as 0. If the percentage of samples being classified as the label 0 increases by a certain percentage with the attack, it is successful.

There are some small deviations with this objective, such as the attack should try to turn the highest amount of 1 into 0, instead of just increasing the total amounts of 0, especially if an extra objective is for the tampering not to be clearly perceptible to the users.

This type of attack can be very impactful if made against models that can influence markets [2]. If, for example, a product represented by label 0 gets to be recommended more, then some profit is generated at the expense of another product. And in this case, the case described in the last paragraph does not really matter, if increasing profit is the only objective of the attack.

Backdoor

These attacks focus on trying to get a certain pattern to be misclassified by the model. This pattern can be a single characteristic of an entry or a combination of them. A very simple-to-understand example is a model that classifies access to a system as a threat or non-threat. If we suppose one of the attributes of a sample is the origin IP address for the access, and based on the data that trained the model, a certain IP X is always a threat, the objective of this attack can be to try and make the model classify samples with that IP X as non-threat instead.

As mentioned in the previous attack type, discretion is another vector of qualifying the success

of this attack, but in this case, it is even more important because even the name Backdoor already suggests something that is done undetected. So the success of this attack is then measured by achieving the change of classification in a certain pattern, as mentioned above, together with keeping the rest of pattern classifications unchanged, so it is as stealthy as possible.

Model Breaking

This last type of Data Poisoning attack has the objective of trying to make the model the least accurate as possible. If considering a binary classification model, it would be flipping the label of the highest amount of sample classification. If talking about a model that classifies between ten different labels, it would be only making the accuracy drop as low as possible, with the distribution between labels not really mattering.

Data Poisoning in the Literature

The first important point to make when it comes to Data Poisoning, which correlates also with the next category, Adversarial Sample Attacks, comes to the fact that these two categories are often bundled together as seen in Claudia et al [7]. In this cited paper, Data Poisoning is just a subcategory of Adversarial Attacks. This Adversarial Attacks category is characterized as "injection of malicious data samples, to manipulate the model and to disclose information about the original data being used for training or inference purposes". No specification of where the malicious data is injected is mentioned, and for that reason alone, the descriptions of Data Poisoning and Adversarial Sample attacks given in this work are justifiable, as they are more rigorous and draw this separation in relation to adversarial power. The usage of the wording "injection" also does not take into account the actions of deleting or modifying existent entries of the training dataset, as a method for performing Data Poisoning.

Jing Lin et al [24] makes the same separation between Data Poisoning and Adversarial Attacks. It mentions that the latter can only act upon the testing phase of the pipeline, or the already deployed model, not being able to impact the training process, while the former can manipulate the training process. This same paper describes Data Poisoning as manipulating the training data in an attempt to either: decrease overall performance (i.e. accuracy), induce misclassification, or increase training time. These three different objectives fall into the general bucket that is impacting the performance of the training step, and also the final model product.

Another subdivision that this paper mentions is targeted and untargeted attacks, definitions that fit well the subcategories Backdoor and Fix for targeted and the subcategory Model Breaking for untargeted. Later in the paper, the authors describe three different example attacks for Data Poisoning, but the main point for the first two attacks is **how** the attacks are performed, instead of what their objectives are, which is the main characteristic utilized in this Dissertation for the division of Data Poisoning attacks. The last attack though is the Backdoor attack, which is consistent with the previous definition of the Backdoor Data Poisoning attacks.

On the other hand, Goldblum et al [13] define Backdoor attacks as needing a specific trigger during test time to generate the desired objective. This terminology of Backdoor seems to be more related to the more general term backdoor utilized for cyber security situations, diverging from our definition, supported by the paper above [24]. Other than that, this paper also defines a category of Data Poisoning, labelled *Training-only Attacks* that fits our description of Data Poisoning in general, specifying no access at test time, and only tampering with the training dataset.

Aghakhani et al [1] gives a very fitting description of Data Poisoning if compared to the description given above. This work gives an example of our definition of Backdoor attacks without using said terminology, but it is the exact same situation, where an attacker wants, by tampering with the training dataset, to force the model to give a specific misclassification during inference.

Biggio et al [5] mention that sometimes the attacker does not have full access to the training dataset, being only able to add entries to it, by participating in the collection of data but giving fake information. This paper also describes an attack where the main focus is to decrease the model accuracy, fitting perfectly into our Model Breaking scenario.

There are many other papers such as [38], [29] and [22] that are either about the mathematical analysis of how to perform Data Poisoning attacks the most efficiently way, or about how to avoid security systems that are placed to detect Data Poisoning. These papers are interesting when talking about the implementation aspect of the problem, but since this section is about giving a more theoretic approach to defining machine learning attacks into categories, they are not relevant in that aspect. Even these papers, though, define the adversary actions as tampering with the training dataset, in different manners.

3.1.2 Adversarial Samples

We will define these attacks as the following: The adversary will interact with the model, making queries to it, and then modify these queries in specific ways, in order to obtain results aligned with the specific security criteria. Its goal is to craft queries that will be misclassified by the model, which can be obtained through many "rounds" of interaction with the model. The amount of information required by each subcategory of these attacks varies, and as such, four different attacks were defined: Decision-based, Score-based, Gradient-based and Transfer-based.

One popular example that fits this category of attacks is the misclassification of traffic sign images [34]. In this case, there is a model that classifies images of traffic signs. If an image of a stop sign that is correctly classified, with high associated confidence values, by the model, and it's modified in a way that to a human it still looks like the same picture, but the model classifies it now as another sign, that's an Adversarial Sample attack. This Adversarial Sample can be reached in many different ways, and also with different objectives. An example of those objectives is obtaining any label misclassification, or a target label misclassification when the adversary's objective is to make the model classify an input as another specific label. The example of image classification was used as a simple way of explaining, but this can be also applied to regression models.

Decision-based

The first subdivision of our Adversarial Samples attacks category is Decision-based, where the only information required by these is a hard label classification by the model. Utilizing the same animal picture classification model example as before, this model would only output the name of the species that mostly matches the picture in the input, without any score or mentioning the other labels that ranked following the first one.

This type of Adversarial Sample is the most reliable, due to requiring little to no information [6]. This also makes these attacks more applicable in real client and model interactions, so it poses a very real threat to systems and platforms that offer this type of service to the public.

Score-based

These attacks are very similar to the Decision-based ones, as they are also fairly common in real-world examples, due to only requiring information to be outputted by the model, instead of some inside information. The difference is that here the confidence values are needed to be represented on the model output, for this attack to be performed. The presence of such information, makes the cycle of forging an adversarial sample much easier to quantify the success of any perturbation, resulting in a much easier attack to perform. For example, if a model returns the label Dog with 85% confidence for a specific figure input, and after some perturbation of said figure, it returns Dog with 70% confidence, the attacker can be sure the perturbation represents a positive result for its objectives.

As with any other attack, more information being available is never a bad thing, and for both Decision-based and Score-based attacks, it is plausible for the model to also return a few more labels, representing the next labels that were closest to being the first result. This can represent another avenue for performing such attacks because even if the objective is to generate any misclassification or a target misclassification, increasing the confidence of the model that the input is supposed to be another label, can be utilized to achieve success in these subdivisions of Adversarial Sample attacks. Especially for Score-based, where if a model classifies a figure as Dog with 55% and Cat 50%, it may be easier to increase the Cat label confidence value than to drop the Dog one.

Gradient-based

In this subdivision of attacks, it is required, in addition to the information required by both subcategories above, access to model information, including the gradient of loss, in relation to different inputs. By making small perturbations to the input and analysing the gradient of loss, the adversary crafts its adversarial sample.

Transfer-based

In this case, having access to the full training data (or at least a large and well-representative portion) is necessary as the goal is to use that data to generate a new model, that will behave similarly to the original and will be used as a replacement. It is very closely related to model

extraction in a way because the model can alternatively be obtained that way, as seen on [33]. This substitute model is then utilized to find adversarial samples that are misclassified by the substitute in the hopes that it will also be misclassified by the original model, that's why the name "transfer-based".

As seen here, these attacks require a type of White-box access to the original, as it needs the dataset, in order to emulate a parallel model. With access to this parallel model, the difference in classification between the two models can also be utilized to adjust the fake model to make it closer to the original, in turn helping the generation of the adversarial samples.

Adversarial Samples in the Literature

As described in the Data Poisoning section, these attacks are often mixed together, even though a conclusion that they had to be separated was drawn. This line was established based upon the fact that Adversarial Sampling attacks do not require writing access to the training dataset, including the addition of new entries, not requiring access to the training dataset at all in most subcategories. This category of attacks though is much ampler than the previous category, due to having many different requirements for attacks that can still have the same objective. All subcategories differ on the amount or type of information needed for performing such an attack, but all of them have the objective of crafting a sample that gets misclassified by the target model. By reading the literature, if you remove all Data Poisoning attacks from the label of Adversarial Attacks, what remains is usually the Adversarial Sampling described by the above subcategories. For example, this paper [7] fits into this logic perfectly, so by removing the Data Poisoning attacks from the Adversarial Attacks label, we have exactly the four subcategories defined above.

Goodfellow et al [14] describes something called adversarial examples, which consist of finding samples that due to the worst possible perturbation, in relation to the model, output a misclassified label with high confidence values. While this description fits our general Adversarial Sampling, the paper goes on a tangent of **why** such things are possible, relying on pointing out that the linear nature of neural networks is the cause for this vulnerability.

Papernot et al [32] also affirms that adversarial samples are only created after the training process, at test time, and do not alter the process in any way. The definition of adversarial sample in this work is also the same as the one given in this document, where "an adversarial sample is an input crafted to cause learning algorithms to misclassify". It is also mentioned

that adversarial samples are created by adding distortions to benign samples and exploiting the generalization learned by Deep Neural Networks.

An example of a Transfer-based Adversarial Sample attack is described by [33], where a technique that can be classified as a Model Inversion attack is utilized, by querying in a specific method, a synthetic dataset can be obtained, that is good enough to train the substitute model utilized then to perform the Transfer-based attack. It also mentions some difficult aspects of training a substitute model, including, but not limited to, having to decide on an architecture without knowledge of the original model architecture. It follows by explaining the specific steps on how to train such a model and perfect it for the target usage.

Brendel et al [6] mention all four subcategories described above, whilst saying that Decision-based attacks are the most robust, as it is not affected by simple defence mechanisms that work against every other subcategory, making that specific subcategory applicable to hard Black-box scenarios, and also calls the information required by specially Gradient-based attacks and Score-based attacks unrealistic to be available since most real-world examples do not offer such information. It also proposes an avenue for Decision-based attacks and draws comparisons to other existent attacks such as [8] and [35].

Carlini et al [8] is another example where this category of attacks is mentioned as *adversarial examples*, but it has the same definition as the Adversarial Sample category. It points out the work by Szegedy et al [43] as the source of this terminology. That paper by Szegedy et al explains how even though machine learning models are robust to small random perturbations, offering stability in their classifications, a small non-random perturbation can cause a misclassification, by optimizing the input to induce this scenario.

3.1.3 Model Extraction

We will define these attacks as follows: The adversary will utilize some information about the target model, according to the subcategory of the attack, to generate a counterfeit model similar to the original, which approximates a similar function.

There are many types of Model Extraction attacks, as many of them offer different methods to perform this attack and can also differ on which type of models they can be used on. These subdivisions are Equation-solving, Path-finding, Class-only and Data-free Knowledge Distillation.

Equation-solving

This subdivision of our definition of Model Extraction attacks tries to extract the model by solving a mathematical equation. For a i dimensions input, it requires $i + 1$ queries on the input vector. It is based upon class probabilities and confidence values, making the required information close to a Score-based Adversarial sample attack, but together with publicly available data from other sources, the requirement of information can be reduced. This type of attack is not applicable on decision trees [44]. The knowledge of the dimension of the original training dataset is also required for performing this attack.

Path-finding

This subdivision on the other hand is focused on decision trees. The goal is to recreate the internal tree generated by the model as a way to recreate the model itself. This can be executed by recreating the path from each leaf on the tree, and requires information about the tree, confidence values and data features, making this a more information-intensive requirement than the previous attack, for example.

In this subcategory, the name is given by the main attack of the genre, but other attacks that focus on stealing parameters from the models in an attempt to train a parallel version are not discarded, just disregarded as a main topic since most attacks of this genre fit under Path-finding and the ones that don't, usually have different requirements and methods, as seen on [46], making them too similar to other categories or relying on other attacks.

Class-only

Class-only attacks require concrete samples from the training dataset to be executed and are done upon binary classification models. The fact it requires information from the training dataset makes this type of attack very different from the previous two, as the location of the required information is different, targeting an earlier step of the pipeline. In case of collusion of attackers, something that has been increasingly studied in this field, someone with the capacity of doing a poisoning of the dataset, for example, can provide the required information for this attack, making it possible.

Data-free Knowledge Distillation (DFKD)

Knowledge Distillation is the process of generating from a first model, a second model, smaller in size, but with a similar function and good enough accuracy. Such process usually requires access to the original training dataset of the first model, or a least a portion of it, but many of those datasets may not be available, leaving such process unfeasible. In counterpart, Data-free Knowledge Distillation is when access to the training dataset is not required at all, enabling Model Extraction attacks on this premise.

This attack has shown that Model Extraction can be effective with only knowledge of publicly available information from training data, namely statistics information (e.g., gender and/or demographic statistics) [45]. Part of this information is what is then transformed into the dataset, but the rest can also be useful to reach the attacker's goal.

Model Extraction in the Literature

Given the existence of at least four different vectors of attacks to perform Model Extraction, as seen above, it is not a surprise that many papers decide to focus on only one subcategory. And as papers pertaining to the other categories of attacks, many papers focus on presenting a new attack, developed by the researchers, with a large focus on the analysis of its results.

Tramèr et al [44] though, mainly focus on explaining why Model Extraction is a prominent attack, giving examples in the real world. The main point presented by this work, in relating why this is the case, is that there's a growing tension in the Cloud-based machine learning scene, given that the usage of said models is widely available, but behind a pay-per-query wall, while the training data and the model itself are highly privacy sensitive. These two aspects are protected due to different reasons, the training data can be privacy sensitive due to containing information that due to laws and regulations, cannot be leaked outside a controlled environment, such as information from a hospital pertaining to patients. Or the training data could be utilized for training another model, as seen in the Class-only subcategory of attacks. The model itself being publicly available could also lead to information pertaining to the training dataset being leaked, cycling back to the point just made, but also can be copied, as the objective of Model Extraction attacks. Also brings up the point of having the model publicly available, lets outsiders learn more about it, and how it tends to classify certain queries, making it easier to generate Adversarial Samples (Evasion Attacks), as the last step seen on Transfer-based attacks, which could be

dangerous when talking about threat detection models, for example. The paper describes three attacks, one that fits inside the Path-finding subcategory, and two that fits the Equation-solving one. The most interesting attack is the third one, where only class labels are outputted by the model, making it applicable to models that try to be secure by not outputting confidence values.

The paper mentioned in a subcategory above, by Wang et al [46] has a different approach in the sense that the objective is stealing hyperparameters to perfect the performance of the stolen model. Since the threat model displayed already assumes the attacker has access to the training dataset, the machine learning algorithm and optionally the learnt model parameters, the question proposed shifts from if the attack is possible to how to make such an attack more successful, diverging from the work done in this section, but being very interesting for future research on better modelling Model Extraction attacks, as it is done for Data Poisoning attacks in the following chapters.

Truong et al [45] gives an insight into how Data-free Knowledge Distillation attacks were created and explains their applicability. The paper explains the meaning of Knowledge Distillation and presents the common unavailability of the original training dataset due to confidentiality. It then mentions the discovery of Data-free methods of Knowledge Distillation and affirms that their existence makes Model Extraction attacks of that type plausible. This work mentions the term Generative Models, explaining that the method for DFKD requires the second model, to query the first one as a way to obtain the training data required to train itself.

3.1.4 Model Inversion

We will define Model Inversion attacks as follows: The adversary has some partial information about the training dataset. Then, by querying the model and analysing the output, he is able to infer some other information contained in the training dataset, about the same target, which is his objective.

For example, one famous case [11] is of a model trained to, by utilizing a submitted image of a person, return their personal information. In this case, Model Inversion was utilized to make artificial images (with the help of confidence values and return values from the queries made to the model), that would be so close to the original image utilized to train the model that the appearance of a person could be extracted. The only personal information the attacker had was the name of the person it was trying to reconstruct visually.

Another work done in relation to Model Inversion attacks [12] by Fredrikson et al, also defines Model Inversion attacks, in a very similar way described above. An adversary, given a trained model and some unknown information, attempts to reach some sensitive and private attribute contained only in the training dataset. The work bets on a correlation between these three factors, which would make this attack possible. In this paper, the target itself is information about patients' genotypes, the unknown attributes are demographic information about the patients, and the variable outputted by the model is warfarin dosage.

He et al [20] treat Model Inversion as an optimization problem, and apply it both to White-box and Black-box scenarios, while keeping the same definition presented by Fredrikson et al [11, 12].

3.1.5 Reconstruction Attacks

We will define Reconstruction Attacks as follows: The adversary utilizes information about the model, especially about feature vectors, and by querying it, tries to reconstruct the original training dataset. Its biggest difference from the category above, is the need for the information about the inner workings of the model, without requiring partial knowledge of the original training dataset.

Even though it is hard to classify this as Black-box, studies have shown this information can be acquired through other means than having the model itself leak this information. Some examples are to infer the feature vectors by analysing a similar model or having access to another dataset with the same distribution [39] and together with the Black-box access try to find out the needed additional information. This can also be done through different attacks [46].

Rigaki et al [37] groups Membership Inference and Model Inversion attacks together under the Reconstruction Attack category, but as mentioned above, this grouping is not justified due to the fact that any type of Reconstruction attack requires extra information about the model, not being applicable in hard Black-box scenarios. While some attacks pertaining to the other two categories assume a more Gray-box type of scenario, the type of information required is not usually about algorithmic and mathematical specifications, as it is in this category. Another factor that suggests the differentiation between Model Inversion and Reconstruction Attacks, is while both of them can require some information about the training dataset, one can utilize structural information, such as the knowledge of it being a subset of another dataset [39], as it is with Reconstruction Attacks, while Model Inversion ones require information about the

training data itself, such as an outside attribute. It can be argued that performing a Membership Inference attack on every single member of a training dataset could consist of a Reconstruction attack, but since this is not usually the most efficient approach for reaching such an objective, it still made sense to divide them into two subcategories, without mentioning that the requirements can be different as well.

3.1.6 Membership Inference

This category of attacks will be defined as follows: By utilizing outside information about said entry, and querying the model with said entry, to make use of confidence values returned, the adversary infers if the entry was present in the training dataset. The real objective is to be able to obtain some other information about the target data entry, that can be assumed due to its participation in the training dataset. Our definitions of Model Inversion and Membership Inference attacks are similar, but they differ when it comes to which type of information about the target data entry is required and whether the goal information is present in the training dataset or is inferred.

A very common example of this category of attacks is an adversary trying to infer the participation of a target in the training of a model that is associated with a specific disease [21]. The confirmed participation of said entry in this training dataset implies a high chance of said target having the associated disease.

This attack is based upon the fact that models tend to return a way bigger confidence value when exposed to data present in the training dataset, as a prediction itself is not needed and the result is known, which could be described as a symptom of the models overfitting to the training dataset. In theory, as said on [39], this could be used to make a full-on Reconstruction Attack, but it is not applicable as it is not efficient enough to reconstruct the entire original raw dataset by these means.

Shokri et al [40] affirm that machine learning models leak information about the dataset in which they were trained. It also presents a method for performing Membership Inference attacks by training another model that would have the ability to identify whether a specific classification done by the original model was affected in a way that signifies the presence of said data in the training dataset.

Nasr et al [31] reinforces the idea of Deep Neural Networks being susceptible to Membership

Inference attacks due to them *remembering* information about their training data. This paper also presents how effective a White-box vs. a Black-box attack is and presents the difference between a stand-alone pipeline scenario, similar to our generic machine learning pipeline, and a Federated Learning scenario, describing how the attacker could potentially be the aggregator. The same authors, in this other work [30], focus on mitigating the risks of Black-box attack scenarios by trying to defend against specifically the most effective attacks, and give a similar description of Membership Inference attacks and the lack of security relating to the leakage of information by the model, as expected, that fit our description of this category.

Long et al [27] question the need for overfitting to be present to make the model vulnerable to Membership Inference attacks, concluding that it is sufficient, but not required. This is done by introducing a new type of attack, called Generalized Membership Inference Attack (GMIA), where the information to be inferred can be done by querying multiple different queries instead of relying solely on the one characterized by the target. This proves that generalization is not enough to protect the training data against Membership Inference attacks, deeming them more threatening than given credit for. This last point is an extension and reaffirmation of this other paper [26] by the same authors, which focused on measuring membership privacy, explaining why Membership Inference attacks succeed.

Song et al [42] also present methods to circumvent the failures of previous Membership Inference attacks against models that were not overfitted to the training dataset. In the example given, the target model presented a similar accuracy when classifying test inputs versus training inputs, proving the non-overfitting. The main goal of the paper is to utilize Membership Inference attacks against publicly available natural language processing models so users can identify the usage of their personal work as part of the training dataset, which was performed without their authorization, going against the GDPR.

Hagestedt et al [18] exemplifies how impactful the existence of Membership Inference attacks, hinting at their efficiency, is by mentioning how certain models cannot rely on data that would be useful to enhance them due to such data being privacy sensitive and being able to be obtained due to such attacks. It goes on to present a system that can, in this case, integrate such data and is secure against Membership Inference attacks.

Chapter 4

Work Development

After listing and defining different attack vectors, the objective now is to refine these same definitions into something concrete and quantifiable. With that in mind, we have chosen to narrow down the work to data poisoning, given the well-defined behaviour of the attacks.

Now, we will propose experiments that capture the different models considered in data poisoning, both at the system level—Gen or FL—and at the level of what constitutes an attack in the system—Fix, Backdoor, and Model Breaking.

A Cryptographic Approach to Defining Data Poisoning

The Data Poisoning attack consists of modifying the training Dataset in a way to generate a biased model. In all of the definitions below, a very simple binary classification model was assumed, where to any queries, the results returned are 0 or 1. This can, as expected, be expanded to more real and complex scenarios with a few modifications, but the decision to keep such a simple model as the target of the attack was made to facilitate explanations and to remove the most situation-specific variables and steps, as to generalize as much as possible.

As discussed previously, Data Poisoning attacks were divided into 2 subcategories, Generic Poisoning and Federated Learning. Each subcategory is also divided into what we consider the three main objectives of a Data Poisoning Attack:

- Fix: Maximizing or minimizing the number of classifications done by the model that corresponds to a specific label. In the case of a binary classification model, this can be

exemplified as making the model classify as many queries as possible as 1, shifting the average of queries classified as 1 to be as close as 1 as possible, where the average is calculated by the number of queries, divided by the number of queries that were classified as the label 1.

- Backdoor: Making the model misclassify a specific pattern. This can be very easily exemplified by a threat detection model, where we can assume for each query, a classification of label 0 means that it is not a threat, and 1 means it is. So for a certain combination of attributes that compose the query, the objective of this attack is to make the model classify a threat as a non-threat, returning 0 instead of 1.
- Model Breaking: Making the model misclassify the highest amount of queries as possible. In a binary classification model, this implies label flipping, from 0 to 1, and vice versa, but an argument can be made that for non-binary models, this can be even easier because the label given by a misclassification is not defined, so there are more options for this to be made.

As seen above, the win conditions all are about comparisons against the non-poisoned model, and because of that, all games consist of training two models, one poisoned and one non-poisoned, and comparing their performances.

Notation

Due to having two models, all characteristics, such as variables and constants present in both of them, but specific to each one, will be represented by the same symbols, with the addition of $'$ to identify the poisoned ones. So, the non-poisoned model will be represented by m, while the poisoned model is represented by m'.

Below, some of the variables and constants utilized to represent operations and objects pertaining to the process and the attack will be listed, but it is worth mentioning that for each specific section below, the process will be better described, including the utilization of the items listed here and some others specific to each section, which are not:

- D - Training Dataset.
- D_T - Test Dataset.
- $\Pi(D)$ - Function that represents the training of a model utilizing Dataset D.

- m - Machine Learning Model.
- A(x) - Attacker function. Used to represent the actions of the attacking party, with x representing its arguments.
- q - a single query to the model.
- L - a specific feature associated with each entry of the Dataset, to be inferred by the model.
- I($m, q/D_T, L$) - Inference of feature L from query q or Dataset D_T by the model.
- t - Threshold for corrupting the training Dataset. For example, if talking about the Genp scenario and t is 5, the difference between the original Dataset and the poisoned one can only be 5%. In a Federated Learning scenario, t is the number of participants that are attackers in a specific round.
- Count() - Function to count the number of entries of a set that were classified as the target label.
- f() - Generic aggregation function for the Federated Learning scenario.
- Compare() - Function that compares all classifications done by two models for a Dataset and returns the number of classifications that were different, in percentages.

4.1 Generic Poisoning (Genp)

This first game subcategory of Data Poisoning attacks relates to simple scenarios, considering that the model is trained only once, and the training Dataset is unified. This scenario is aligned with the simplified Machine Learning pipeline described in the Chapters above. Can also be colloquially defined as a corruption performed in a traditional scenario of training followed by inference in machine learning.

4.1.1 Fix: Fix-Genp(L, t, c)

As described above, the objective of this type of attack is to maximize or minimize the presence of a certain label in model classifications, for this specific game example, let's assume maximizing the label 1 is the goal. In this case, the objective is to modify the biggest number of queries to have a result of 1, where previously was 0, in relation to the feature L. This becomes clear when looking at the final lines, where the error is given by the subtraction of the sum of entries classified as 1 in the poisoned model by the non-poisoned one.

As shown in figure 4.1, first, the attacker poisons the training Dataset D, creating the

poisoned Dataset D'. Both models are then trained using the $\Pi()$ function, the non-poisoned model m, and the poisoned model m'. After that, the feature L is then inferred for the whole test Dataset D_T, for both models and attributed to r and r'. Then, iterating through all entries in the test Datasets, s and s' are increased for each time an entry was classified as the target label c, by the corresponding model. δ is then calculated by subtracting the number of entries classified as the target label by the poisoned model (s') from the number of entries classified as the target label by the non-poisoned model (s). Finally the game returns !Small(δ), where Small() is a predicate that returns True if δ is below a configurable security criterion.

$\underline{\mathsf{Fix\text{-}Genp}(D, D_T, L, t, c)}$:
$D' \leftarrow\!\$\ \mathcal{A}(D)$
$if \frac{|D \cap D'|}{max(|D|,|D'|)} \leq t : \mathsf{Return}(\perp)$
$m \leftarrow\!\$\ \Pi(D); m' \leftarrow\!\$\ \Pi(D')$
$r \leftarrow \mathsf{I}(m, D_T, L); r' \leftarrow \mathsf{I}(m', D_T, L)$
$s \leftarrow 0; s' \leftarrow 0$
for i in $[D_T]$:
    $s \mathrel{+}= (r[i] == c)$
    $s' \mathrel{+}= (r'[i] == c)$
$\delta = s' - s$
$\mathsf{Return}(!\mathsf{Small}(\delta))$

Figure 4.1: Fix attack for the Genp scenario, given the threshold of corruption t of the training Dataset D.

4.1.2 Backdoor: BD-Genp(L, t, q, c)

The objective of this attack is to set a specific feature L pertaining to a specific query q to be classified as the inverse of the originally classified value by the non-poisoned model. For example, let's imagine a model that works for threat detection. Now, if we manage to poison the Dataset in a way that a specific threat is now classified as a non-threat, we would have achieved our objective. The opposite is also true, we could make a different type of attack by making the model classify a non-threat as a threat, and as such, shut down the access of a service or user access, for example.

The following description is represented by figure 4.2: To start, the attacker creates the

Dataset D', by poisoning the original Dataset D. The limitation of corrupted entries t in Dataset D' compared to D is checked. Then both models are then trained using the $\Pi()$ function, the non-poisoned model m, and the poisoned model m'. Finally, the label L is inferred for query q by both models and if they're opposites, the game returns **True** and the attack was a success. The mathematical notation of an inverse is utilized considering that 0 is the inverse of 1, and vice versa, in a binary classification model.

$\underline{\mathsf{BD\text{-}Genp}(D, L, t, q)}$:
$D' \leftarrow\$ \mathcal{A}(D)$
$if \frac{|D \cap D'|}{max(|D|,|D'|)} \leq t : \mathsf{Return}(\perp)$
$m \leftarrow\$ \Pi(D); m' \leftarrow\$ \Pi(D')$
Return $\mathsf{I}(m', q, L) == (\mathsf{I}(m, q, L))^{-1}$

Figure 4.2: Backdoor attack for the Genp scenario, given the threshold of corruption t of the training Dataset D, considering a binary classification model.

The game displayed in figure 4.2 is very specific to binary classification models, but it can be more generalized with a simple change, that with the addition of a target label classification c, the final condition check can be if the value returned by $\mathsf{I}(m', q, L)$ is equal to c. In this case, there's no need for comparing both models, since the original model classification does not need to be compared with the poisoned model one. This differs a bit from all the other games, which have this aspect of comparison to the original model, but still, the context should be enough to justify it. It is also worth noting that the value of $\mathsf{I}(m', q, L)$ has to be different from the one obtained when training with D, or else this would lead to a trivial attack. This version is represented in figure 4.3.

$\underline{\mathsf{Gen\text{-}BD\text{-}Genp}(D, L, t, q, c)}$:
$D' \leftarrow\$ \mathcal{A}(D)$
$if \frac{|D \cap D'|}{max(|D|,|D'|)} \leq t : \mathsf{Return}(\perp)$
$m' \leftarrow\$ \Pi(D')$
Return $\mathsf{I}(m', q, L) == c$

Figure 4.3: A more generalized Backdoor attack for the Genp scenario, given the threshold of corruption t of the training Dataset D.

4.1.3 Model Breaking: MB-Genp(L, t, b)

This attack, represented in figure 4.4, aims to poison the Dataset so that the biggest number of misclassifications are generated by the poisoned model, by reducing the model's accuracy, in a way that a least a percentage b of entries is misclassified. The attribute t is still the threshold for corrupting the original Dataset. The first steps are similar to the other Genp Scenario attacks, such as generating the poisoned Dataset D' and training both models, m and m', with the function $\Pi()$. Next, each entry of the Dataset D_T is inferred by both models, into r or r' depending on which model infers it. The variable s is then set to 0 and for each entry in D_T, if the result of the inference present in r is different from the one present in r', s is increased by one. Finally, if the percentage given by s over the number of entries present in D_T is greater or equal than the breaking threshold b, the attack is considered a success.

$\underline{\mathsf{MB\text{-}Genp}(D, D_T, L, t, b)}$:
$D' \leftarrow\!\$\ \mathcal{A}(D)$
$if\ \frac{|D \cap D'|}{max(|D|,|D'|)} \leq t : \mathsf{Return}(\perp)$
$m \leftarrow\!\$\ \Pi(D); m' \leftarrow\!\$\ \Pi(D')$
$r \leftarrow \mathsf{I}(m, D_T, L); r' \leftarrow \mathsf{I}(m', D_T, L)$
$s \leftarrow 0$
for i in $[D_T]$:
    $if(r[i] \neq r'[i]) : s \mathrel{+}= 1$
Return $(\frac{s}{|D_T|} \geq b)$

Figure 4.4: Model Breaking attack for the Genp scenario, given the threshold of corruption t of Dataset D and target breaking threshold b.

4.2 Federated Learning

In this subcategory, all the same attacks will be fit into games like the ones above, but this time, applied to a Federated Learning scenario. In this scenario, every subcategory of Data Poisoning attacks will have two different games. The first game describes the situation where just one round of training is poisoned, while the second describes a situation that occurs from scratch, over K number of rounds. Another big difference between the Genp and FL scenarios is the possibility of the attackers performing an extra vector of poisoning, in the FL scenario.

This means that the attacker can just poison the training Dataset of the client he's taking over and perform the generation of the attribute v to be aggregated in a legitimate form, or he can manipulate the v attribute itself, not utilizing the training Dataset, poisoned or not, together with the model generated by the last round of Federated Learning.

4.2.1 Fix: Fix-FL1(L, n, t, c)

In this case, the objective is to modify the biggest number of queries to have a result of 1, where previously was 0, in relation to the inference of the feature L. This becomes clear when looking at the final lines, where the error is given by the subtraction of the sum of 1 in the model post-attacker participation by the model of the previous round, where no attackers were present.

The following description correlates to the attack present in figure 4.5: First, for each non-attacker participant i, the attribute v_i is generated utilizing the related Datasets D_i and function $\Pi()$. Next, for each attacker j, more attributes v are generated, but this time by the Attacking Function $\mathcal{A}()$, utilizing all Datasets D_i as input. Then, two models are trained, m and m', utilizing the generic aggregation function f(). The first model is trained only utilizing the non-attacker generated attributes v, and the second is trained utilizing all attributes generated in previous steps. After that, the feature L is then inferred for the whole test Datasets D_T, for both models and attributed to r and r'. Then, iterating through all entries in the test Datasets, s and s' are increased for each time an entry was classified as the target label c, by the corresponding model. δ is then calculated by subtracting the number of entries classified as the target label by the poisoned model (s') from the number of entries classified as the target label by the non-poisoned model (s). Finally the game returns !Small(δ), meaning that if it returns True, the attack was a success.

4.2.2 Fix: Fix-FL2(L, K, c)

This second game, displayed in figure 4.6 is very generic and can fit many different Fix attacks on a Federated Learning scenario as the number of participants both attackers (t) and honest participants (n) are decided each round (r), making it more configurable. This attack is also able to capture attacks in a specific time window, such as two months of a system that has been up for years, for example. This is done by the function Round(), which also returns the training Datasets $D_1..D_{n+t}$ to be utilized by each client, including attackers. K is the number of rounds

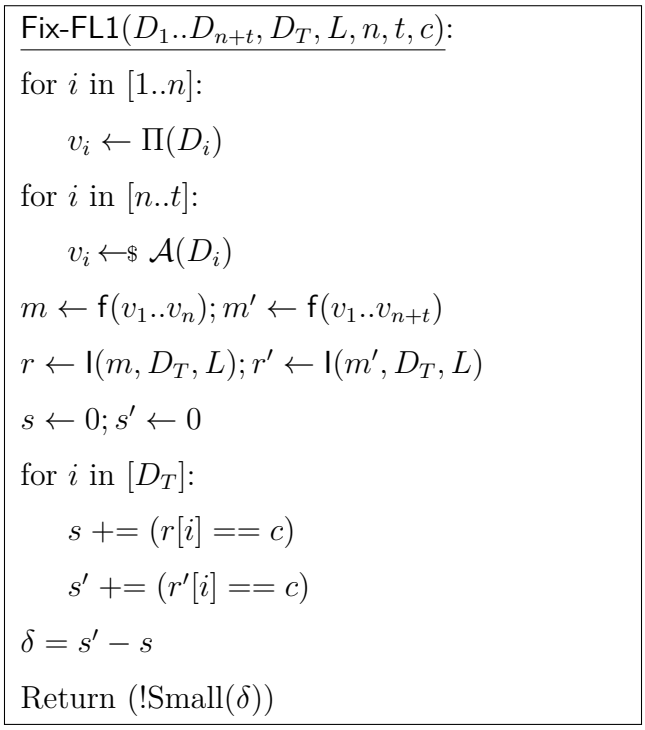


Figure 4.5: Fix attack for the Federated Learning scenario, given the threshold of t attackers out of n participants, and the target label c to be fixed.

that will be performed, and as such, for each round, after the participants are decided, two models are generated by aggregation of the corresponding attributes vHi and vi and utilization of models from the previous round, to generate the models mHr and mr. The H in both the attributes and model represent that these are the honest-only variables, they represent what would be the result of the model each round if no attackers ever joined the FL process (t = 0). This notation was chosen over the previous m and m' one due to the fact that the Data Poisoning can skip some rounds, and a way to track what would happen in a scenario where no attacker ever joined on any previous round is very beneficial for testing and performance evaluation purposes. As expected, the vi generated by the attackers is done utilizing an attacking function $\mathcal{A}()$. The results of each round are calculated and then stored in the corresponding index of the result vector, such as $\delta[r]$, for round r. Finally, the whole vector is utilized as the attribute to calculate the success or failure of the attack. Compared to the game in the last subsection, this one substitutes r for res to signify the result, since r is utilized for round number.

The fact that each round result is stored in the δ vector, allows for, with a small adaptation, the calculation of the attack success in any desired interval of rounds, being that all of them (last - first), just one $\delta[r]$ or even specific rounds. Also allows for the observation of how lasting the impact of poisoning one or many rounds can have on the following ones. It is assumed that

during the first round, m_{Hr-1} and m_{r-1} are just empty, and the $\Pi()$ function trains the models just with the Dataset as an argument since a previous round model does not exist yet.

Fix-FL2(D_T, L, K, c):
for r in $[0..K]$:
  $(n, t, D_1..D_{n+t}) \leftarrow Round()$
  for i in $[1..n]$:
    $v_{Hi} \leftarrow \Pi(m_{Hr-1}, D_{i,r})$
    $v_i \leftarrow \Pi(m_{r-1}, D_{i,r})$
  for i in $[n..n+t]$:
    $v_i \leftarrow\$ \mathcal{A}(m_{r-1}, D_{i,r})$
  $m_{Hr} \leftarrow \mathsf{f}(v_{H1}..v_{Hn}); m_r \leftarrow \mathsf{f}(v_1..v_{n+t})$
  $res \leftarrow \mathsf{I}(m_{Hr}, D_T, L); res' \leftarrow \mathsf{I}(m_r, D_T, L)$
  $s \leftarrow 0; s' \leftarrow 0$
  for i in $[D_T]$:
    $s \mathrel{+}= (res[i] == c)$
    $s' \mathrel{+}= (res'[i] == c)$
  $\delta[r] = s' - s$
  $m_{Hr-1} \leftarrow m_{Hr}; m_{r-1} \leftarrow m_r$
Return $(!\mathsf{Small}(\delta))$

Figure 4.6: Fix attack 2 for the Federated Learning scenario, given the target classification c to be fixed, and over the span of K rounds.

4.2.3 Backdoor: BD-FL1(L, n, t, q, c)

The objective of this attack, represented in figure 4.7, is to set the inference of a specific feature L pertaining to a specific query q to the inverse of the originally classified value by the non-poisoned model. For example, let's imagine a Federated Learning model that works for threat detection. Now, if the attacker participant manages to influence the model generated in this round in a way that a specific threat is now classified as a non-threat, the attack is successful, and vice-versa in case of targeting a previous non-threat query.

The first few steps are equal to the Fix-FL game, resulting in all the attributes v being generated, both for attackers and non-attackers, and the models m and m' being trained. The

last step is to infer the Label L for query q from both models and then compare if they're opposites, returning True if they are, signalling a success.

$\underline{\mathsf{BD\text{-}FL1}(D_1..D_{n+t}, L, n, t)}$:
for i in $[1..n]$:
    $v_i \leftarrow \Pi(D_i)$
for i in $[n..t]$:
    $v_i \leftarrow\!\!\$\ \mathcal{A}(D_i)$
$m \leftarrow \mathsf{f}(v_1..v_n); m' \leftarrow \mathsf{f}(v_1..v_{n+t})$
Return $\mathsf{I}(m', q, L) == (\mathsf{I}(m, q, L))^{-1}$

Figure 4.7: Backdoor attack for the Federated Learning scenario, given the threshold of t attackers out of n participants, and the target classification c out of query q.

As with the Genp scenario, this attack is very fit to binary classification models, so with the same type of alterations, figure 4.8 displays another version that would be able to fit a more varied selection of models, with the same caveats.

$\underline{\mathsf{Gen\text{-}BD\text{-}FL1}(D_1..D_{n+t}, L, n, t, q, c)}$:
for i in $[1..n]$:
    $v_i \leftarrow \Pi(D_i)$
for i in $[n..t]$:
    $v_i \leftarrow\!\!\$\ \mathcal{A}(D_i)$
$m' \leftarrow \mathsf{f}(v_1..v_{n+t})$
Return $\mathsf{I}(m', q, L) == c$

Figure 4.8: A more generalized Backdoor attack for the FL scenario, given the threshold of t attackers, n legitimate participants, and target label of classification c in relation to query q.

4.2.4 Backdoor: BD-FL2(L, K, q, c)

There are still two different games here, one for binary classification, and the other more generic and not as strong, for non-binary classification models.

This binary classification attack (figure 4.9) is the more generic version of the attack described in the last subsection, given that this version can describe a full session of Federated Learning training, including a span of K rounds. The number of participants, both the non-attackers n

and the attackers t are defined together each round as the individual clients training Datasets D_i by being the return value of the generic function Round(). The "H" in v_{Hi} and m_{Hr} is representative that these variables are independent of any kind of poisoning, including in any previous round, and serve as a metric for comparison to the model that has been or will suffer poisoning in some rounds. res is the result of inferring feature L for the query q, which is done for both models, and if the results are inverses, this means the attack was successful for this round r, and is stored in $res[r]$. This game only has the list res with the results of the inferences done each round, so the metric for deeming the entire attack successful is delegated to the generic function Verify(res), which based upon that list, defines the success or failure of the attack. The utilization of res instead of r to keep the results of each round is done due to the round number being represented by r.

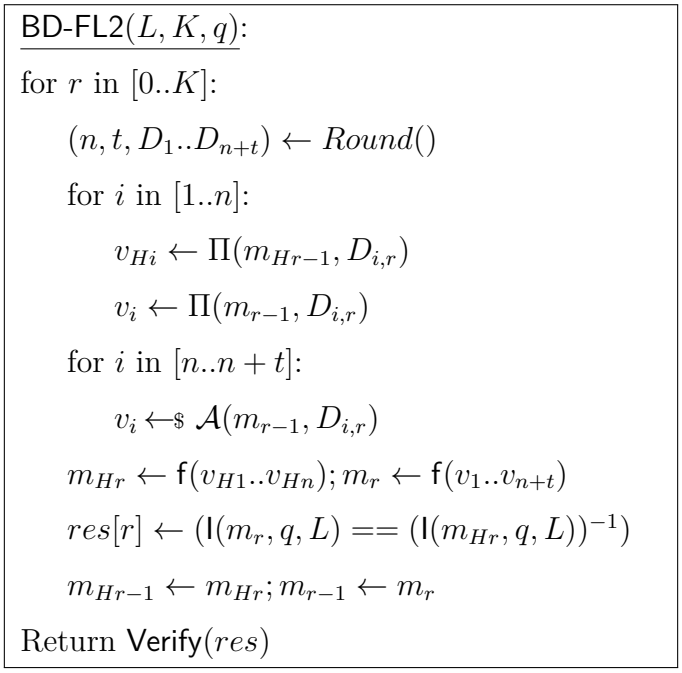


Figure 4.9: Backdoor attack 2 for the Federated Learning scenario for binary classification models, where the number legitimate participants n and attackers t are defined each round r. This attack occurs over the span of K rounds.

As for the non-binary classification Backdoor game, most of the steps are the same, but the honest model is not needed at all, so all the training of this secondary model and keeping track of such model is not included. The main difference is that at the end, $res[r]$ is derived from comparing the inference of the label L for the query q when compared to the target classification c, and if equal, the result of the poisoning is successful for this round, at least. This version is represented in figure 4.10.

Gen-BD-FL2(L, K, q, c):

for r in $[0..K]$:

  $(n, t, D_1..D_{n+t}) \leftarrow Round()$

  for i in $[1..n]$:

    $v_i \leftarrow \Pi(m_{r-1}, D_{i,r})$

  for i in $[n..n+t]$:

    $v_i \leftarrow\!\!\$\ \mathcal{A}(m_{r-1}, D_{i,r})$

  $m_r \leftarrow \mathsf{f}(v_1..v_{n+t})$

  $res[r] \leftarrow (\mathsf{I}(m_r, q, L) == c)$

  $m_{r-1} \leftarrow m_r$

Return Verify(res)

Figure 4.10: More generalized Backdoor attack 2 for the Federated Learning scenario for non-binary classification models, where the number legitimate participants n and attackers t are defined each round r. This attack occurs over the span of K rounds.

4.2.5 Model Breaking: MB-FL1(L, n, t, b)

As mentioned before, this attack aims to poison the Datasets in a way that the biggest number of misclassifications are generated by the poisoned model, by reducing the accuracy of the model, in a way that a least a percentage b of entries are misclassifed. The comparison is done by the function Compare(m, m', D_T), which returns the difference in the percentage of classifications by both models, in relation to the Datasets D_T. The attribute t is, as in the other Federated Learning scenarios, the number of attackers allowed.

Represented in figure 4.11, the first steps are similar to the other Federated Learning Scenario attacks, such as generating all the v attributes that'll be used for aggregation, by the function $f()$, generating both the non-poisoned model m and the poisoned model m'. Next, each entry of the Dataset D_T is inferred by both models, into r or r' depending on which model inferred it. The variable s is then set to 0 and for each entry in D_T, if the result of the inference present in r is different from the one present in r', s is increased by one. Finally, if the percentage given by s over the amount of entries present in D_T is greater or equal than the breaking threshold b, the attack is considered a success.

$\underline{\mathsf{MB\text{-}FL1}(D_1..D_{n+t}, D_T, L, n, t, b)}$:

for i in $[1..n]$:

    $v_i \leftarrow \Pi(D_i)$

for i in $[n..t]$:

    $v_i \leftarrow\!\!\$\ \mathcal{A}(D_i)$

$m \leftarrow \mathsf{f}(v_1..v_n); m' \leftarrow \mathsf{f}(v_1..v_{n+t})$

$r \leftarrow \mathsf{l}(m, D_T, L); r' \leftarrow \mathsf{l}(m', D_T, L)$

$s \leftarrow 0$

for i in $[D_T]$:

    $if(r[i] \neq r'[i]) : s \mathrel{+}= 1$

Return $(\frac{s}{|D_T|} \geq b)$

Figure 4.11: Model Breaking attack for the Federated Learning scenario, given the threshold of t attackers out of n participants, and breaking threshold b.

4.2.6 Model Breaking: MB-FL2(L, K)

Represented in figure 4.12 and similar to the other Federated Learning scenario attacks, here is the version that describes an attack that can happen throughout a span of K rounds, with the number of participants, both non-attacking n and attacking t, defined each round by the generic Round() function, that also returns the training Datasets for each client that specific round. Once more, compared to the game in the last subsection, this one substitutes r for res to signify the result, since r is utilized for round number. The variable b as a breaking threshold is also not present, since this would be done after, over the list β, that accumulates the results for each round. The verification of success is substituted by the generic function Verify(), as in the last attack of the Backdoor subcategory (BD-FL2), to represent this step, which can vary depending on the objective, such as obtaining a high amount of wrong classification on the last round ($r == K$), or over specific rounds.

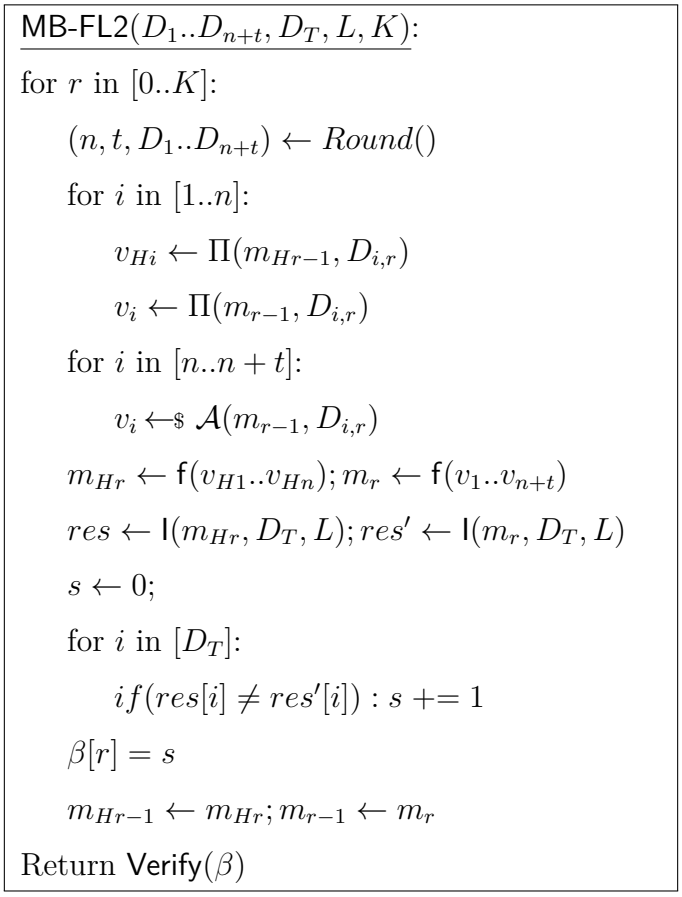


Figure 4.12: Model Breaking attack 2 for the Federated Learning scenario, over the span of K rounds.

Chapter 5

Attack Testing

The objective of this chapter is to verify if the security experiments described in the last chapter are able to capture real-world attacks. With that in mind, we'll take and perform a concrete attack, and illustrate how it fits the proposed definitions.

Our initial objective was to evaluate various attacks discussed in the previous chapter for their applicability to real-world scenarios. However, we faced a challenge since most attacks in research papers are highly specific, making it difficult to take them and run them in a plug-and-play manner. To address this issue, we collaborated with colleagues at the University of Porto and used a basic Federated Learning code example to assess machine learning attacks.

5.1 The Environment

In this section, the Federated Learning environment in which attacks were tested will be described. First, the environment itself, giving an overview of how it is set up and its general functionalities. Second, the modifications done to make the attacks correlated to this work possible will be explained and demonstrated. Lastly, the results obtained will be displayed and commented upon.

5.1.1 Overview

The concrete application of this system is to study malware at different stages and generate a federated classifier that detects them. The context for the attacks is that some participants of the federated learning process may want to be disruptive, in order to promote self-gain, for

example.

The environment in which tests were performed was written in Python, utilizing primarily TensorFlow, especially the TensorFlow Federated interface for Python. Some other classic Python libraries for dealing with data, such as Pandas and NumPy were also utilized. With the intention of serving as a type of sandbox for testing attacks, this environment is composed of a Docker image where the code is presented within a Jupyter Notebook.

The models produced in this environment have both their training Dataset and test Dataset obtained from the same data file. The data contained in this file is about network traffic from multiple malware families, and from Web browsing. The classification objects are Transmission Control Protocol (TCP) flows (sets of packets that share the same client IP address and port, server IP address and port, and protocol). The goal of the models is to identify C2 traffic amongst other "benign" traffic. C2, which stems from Command and Control, is a technique utilized by threat actors to communicate with already compromised machines, over a network. The labelling done in the Dataset is not exhaustive, meaning that some entries marked as non-C2 traffic, can indeed be C2, but the opposite never happens, every entry marked as C2, is indeed C2. The data is then divided into 30 subgroups, 0 through 29, defined by which family of malware the entry is associated with.

The code itself defines a Machine Learning environment where three different scenarios, in relation to how the Dataset will be distributed to the clients, but only two of those were implemented. The first and most simple, called Centralized, represents the scenario where only one client is present and processes all the data by itself. The second, Contextual, organizes the distribution of the data into subgroups based on the malware families associated with each data entry. The last one, which was not implemented, was the Random scenario, where the data pertaining to each subgroup is just random, so data entries related to the same malware family can be spread across all subgroups. Besides that, the Dataset was also partitioned into training, validation and test entries, at a rate of 85-10-5.

Besides the Jupyter Notebook file where the process is defined from start to finish, there are also two other Python files. The first, `federateddatasethelpers.py` is where all the functions that were written to deal with sampling and manipulating the original Dataset are. The second, `federatedtffhelpers.py`, contains many functions from the base documentation of TensorFlow Federated, but with slight modifications to override the original functions.

Process Overview

Firstly, the Dataset file is read with the help of the Pandas library and the number of clients is defined, with all tests to be presented here being done with 10 clients, which translates to 3 subgroups per client. The attribution of subgroups per client for the Contextual scenario is done in numeric order, with the first client being attributed the subgroups 0, 1 and 2, and that follows suit, until the last client get subgroups 28, 29, and 30. For the Centralized scenario, since all subgroups are processed at the same place, this attribution is unnecessary and is skipped. Some constants are then defined, mostly to set parameters of the calculations done at training time, by TensorFlow.

Next, the attack is chosen and three functions are defined to preprocess each client's data at a later time. These functions enable three different situations to be tested. The first, where no client is an attacker, the second, where some of the already defined clients will act as an attacker, and a third, where an extra client is introduced and that client is an attacker. Then, the function to create a model is defined, based upon the `create_keras_model()` TensorFlow function.

The following step, is the partitioning of data between clients, with the help of a function that generates the training Datasets for each client, and organizes them on a list, and a similar function that does the same for the validation and test Datasets. The last two maintain the original rate of positive and negative entries, but for the training Dataset, sampling at a rate of 2 negative samples to 1 positive is done. The process is then initialized, defining some optimizers to be used to evaluate the models, and also a custom aggregation function called `OutlierRejectFactory`, that was implemented to discard outliers when performing the aggregation, in order to protect from attacks.

The training step then begins, where the initial round number, the total number of rounds, and the rounds where any type of poisoning will be performed are declared. Client IDs are also defined, from 0 to n-1, where n is the total number of clients defined before. The model is created, compiled, and directories to store data pertaining to the models are created, one for each scenario.

The first model then trained is the Centralized one, starting by also defining the directory where the logs and summary will be stored, cleaning them up if not empty from tests done in the past. The TensorBoard is also reset, and then the training actually begins. Inside a for loop, each round is then trained and the metrics evaluating the results of each round are stored and

also shown on the TensorBoard. This process takes a while with the default 20 rounds. Metrics are then grouped up for a better display and analysis, shown all together by round in a table. Some printing of the best-performing rounds and best values for the metrics at any time is also displayed.

Next, the Contextual scenario model is trained, with very similar steps as the Centralized one, the main difference being the requirement of utilizing one of the three functions to preprocess the client data, including the definition of which type of attacker would be present between none, taking over clients and attacker as an extra client. Another for loop is then executed, with this one having an if-else condition to deal with non-poison and poison rounds, performing different actions based upon the round number. Similar metrics are then calculated and displayed, with the addition of evaluations done by each client's local model upon their validation Datasets, done each round, to be compared with the same evaluation done by the global model. The best metrics and rounds data dump is now done for each client, instead of being a global one.

At the end, with some values saved during the processing, the most common metrics, such as accuracy, loss, recall and negative recall, can be evaluated in relation to any client at any round of the process in relation to the test Dataset. This is where more metric evaluations can be done, and the models tested here for each client are the ones deemed more fit to deal with new data.

5.1.2 Modifications

As mentioned before, this environment was set up to test attacks on Federated Learning Machine Learning scenarios. The attacks already set up in the environment, solely focused on manipulating the aggregation attribute sent by each attacker-controlled client, in order to affect the training process as a whole, by impacting the aggregation function's ability to produce a good model each attacked round. Later this would be considered also a vector of attacks that could fit into Data Poisoning, as mentioned for the Federated Learning security games, but at the beginning, it was a necessity for any Data Poisoning attacks to only corrupt the training Dataset, not manipulate any other aspect of the training process. Due to this factor, those already existing attacks were discarded, and a few modifications to the process were made to enable attacks that only manipulated the training Datasets of the attacker-controlled clients.

The three Datasets to be attributed to each client are generated through the functions generate_train_datasets() and generate_validation_datasets(), with this second generating both

validation and test Datasets for each client. This first function, is mainly where modifications were made to test the attacks that'll be presented. The original function is displayed by the listing 5.1.

```
def generate_train_datasets_nonpoison(NUM_CLIENTS,
                        dataset,
                        subset_distributions,
                        POS_TO_NEG_RATIO,
                        NUM_CLASSES,
                        print_progress=True):
client_data_lists = { CONT: [], RAND: [], ONEC: [] }
client_data_samples = { CONT: [], RAND: [], ONEC: [] }

pd.set_option('display.max_columns', None)

for data_partitioning, total_sets in [(CONT, NUM_CLIENTS), (RAND, NUM_CLIENTS),
    (ONEC, 1)]:
    for i in range(total_sets):
        if print_progress:
            print(f'\rNUM_CLIENTS={total_sets}, Client {i+1}/{total_sets},
                {data_partitioning}', end='')

        if data_partitioning == CONT:
            sub_df = dataset[dataset['client_id'].isin(
                subset_distributions[CONT][i] )]
        elif data_partitioning == RAND:
            sub_df = dataset[dataset['rand_client_id'].isin(
                subset_distributions[RAND][i] )]
        elif data_partitioning == ONEC:
            sub_df = dataset

        sampler = RandomUnderSampler(random_state=0,
            sampling_strategy=(1/POS_TO_NEG_RATIO))

        sub_df = sub_df[(sub_df['test_set'] == 0)&(sub_df['validation_set'] ==
            0)]

        X = sub_df.loc[:,all_joys].astype(np.float64).values
        y = sub_df['c2'].astype(np.int64).values
        _ = sampler.fit_resample(X, y)
        sub_df = sub_df.iloc[sampler.sample_indices_]
```

```
        client_data_samples[data_partitioning].append(sampler.sample_indices_)
        client_data_lists[data_partitioning].append(
            dataframe_to_dataset(sub_df, NUM_CLASSES) )
    if print_progress:
        print()
if print_progress:
    print("\nConfirming the training sampling ratio:")
    print(sub_df['c2'].value_counts())

return client_data_lists, client_data_samples
```

Listing 5.1: Original function, without poisoning.

The modifications done to this function were very simple, only going around the forced sampling ratio and manipulating the training Datasets needed for each attack test. Since the main type of attacks tested were the ones where all labels C2 were set to 0, to try and reduce the recall metric as much as possible, the sampling ratio had to be circumvented, or else it would break the process. The method utilized for introducing attackers was to simulate a hostile takeover of already defined clients, so specifying which clients to corrupt the Dataset was done by only making such actions when the variable i was equal to the client number that was acting as an attacker at the current test. The modified function is displayed in listing 5.2, with the code set up to make client 1 for the Contextual scenario the one taken over by the adversary, represented by the condition "if i in [1] and data_partitioning == CONT:", performing the poisoning of setting all C2 labels to 0 in that specific training entries.

```
def generate_train_datasets_poisoned(NUM_CLIENTS,
                         dataset,
                         subset_distributions,
                         POS_TO_NEG_RATIO,
                         NUM_CLASSES,
                         print_progress=True):
client_data_lists = { CONT: [], RAND: [], ONEC: [] }
client_data_samples = { CONT: [], RAND: [], ONEC: [] }

pd.set_option('display.max_columns', None)

for data_partitioning, total_sets in [(CONT, NUM_CLIENTS), (RAND, NUM_CLIENTS),
    (ONEC, 1)]:
    for i in range(total_sets):
```

```
        if print_progress:
            print(f'\rNUM_CLIENTS={total_sets}, Client {i+1}/{total_sets},
                {data_partitioning}', end='')

        if data_partitioning == CONT:
            sub_df = dataset[dataset['client_id'].isin(
                subset_distributions[CONT][i] )]
        elif data_partitioning == RAND:
            sub_df = dataset[dataset['rand_client_id'].isin(
                subset_distributions[RAND][i] )]
        elif data_partitioning == ONEC:
            sub_df = dataset

        sampler = RandomUnderSampler(random_state=0,
            sampling_strategy=(1/POS_TO_NEG_RATIO))

        sub_df = sub_df[(sub_df['test_set'] == 0)&(sub_df['validation_set'] ==
            0)]

        X = sub_df.loc[:,all_joys].astype(np.float64).values
        y = sub_df['c2'].astype(np.int64).values
        _ = sampler.fit_resample(X, y)
        sub_df = sub_df.iloc[sampler.sample_indices_]
        if i in [2,4,6] and data_partitioning == CONT:
            sub_df.iloc[:, [80]] = 0
            display(sub_df.iloc[0:5, [80]])
            display(sub_df.iloc[0:5])
            print(sub_df.index[0])
        client_data_samples[data_partitioning].append(sampler.sample_indices_)
        client_data_lists[data_partitioning].append(
            dataframe_to_dataset(sub_df, NUM_CLASSES) )
    if print_progress:
        print()
if print_progress:
    print("\nConfirming the training sampling ratio:")
    print(sub_df['c2'].value_counts())

return client_data_lists, client_data_samples
```

Listing 5.2: Modified function to be capable of poisoning specific client's data.

Another modification that was done, was to remove the utilization of the custom aggregation function, `OutlierRejectFactory`, in favour of the default aggregation function provided by TensorFlow Federated, as aggregation attributes generated by clients with the corrupted Datasets would always get rejected. And as such, this different type of Data Poisoning attack was enabled to be tested, an attack that could be fit into the Fix game description, since setting all labels to 0, the attacked client model would have no incentive to classify any original 0 labelled entries as 1, and comparing the recall would serve fine as a metric of evaluating the success of the attack, the closer to 0 it gets, the more successful the attack was. Even though many different attacks and variations should be also tested, this was not done at this time. Variations such as trying to flip the labels instead of setting them all to 0, in order to mimic a Model Breaking attack, or trying to set only a specific pattern to negative, to implement a Backdoor attack.

5.2 Testing Results

The machine utilized to perform the attacks was a 2018 MacBook Pro, running the Ventura 13.4.1 Operating System, with a 1.3 GHz Quad-Core Intel Core i5 processor, Intel Iris Plus Graphics 655 1536MB graphics unit and 8 GB of memory. The TensorFlow version utilized was 2.5.0, while the TensorFlow Federated version running on the environment was 0.19.0. The docker version utilized to run the container was 4.20.1.

Due to the long time duration of any attack testing and the constant limitations of the machine in which they were performed regularly crashing the process, not as many runs as envisioned were performed, mainly limiting to tests where a number of clients had all the entries of their local training Datasets labels C2 set to 0, and the metrics recorded to be analyzed were all the four main metrics (accuracy, loss, recall and negrecall) at the end of the first round, and at the end of the whole training process, after the last round. Both the evaluation of the metrics done by the already implemented TensorFlow Federated functions and functions implemented to evaluate the global performance (which were deemed better).

The test results here presented were obtained with the previously mentioned Data Poisoning method, and with a total of 10 clients participating in each round, for 20 total rounds. The baseline result, without any poisoning, is displayed in table 5.1.

Clients id	Round	Accuracy	Loss	NegRecall	Recall
-	First	0.9278	0.2951	0.9501	0.4859
	Last	0.9616	0.1393	0.9655	0.8825

Table 5.1: Metrics after the first and last rounds, without poisoning.

5.2.1 One Poisoned Client

The results shown in table 5.2 were obtained by setting only the specified client ID to have the poisoned training Dataset, where all entries had their C2 attribute set as 0.

Clients id	Round	Accuracy	Loss	NegRecall	Recall
0	First	0.9592	0.2890	0.9912	0.3263
	Last	0.9686	0.1205	0.9857	0.6294
1	First	0.9542	0.2389	0.9854	0.3357
	Last	0.9636	0.1255	0.9793	0.6537
2	First	0.9517	0.2613	0.9992	0.0094
	Last	0.9643	0.1196	0.9849	0.5573
3	First	0.9954	0.2541	0.995	0.1701
	Last	0.9714	0.1196	0.9886	0.6310
4	First	0.9556	0.2856	0.9949	0.1762
	Last	0.9667	0.1258	0.9802	0.7003
5	First	0.9625	0.2659	0.9944	0.3296
	Last	0.9673	0.1263	0.9795	0.7252
6	First	0.9488	0.2962	0.9965	0.0039
	Last	0.9659	0.1248	0.9798	0.6898
7	First	0.9555	0.2330	0.9950	0.1729
	Last	0.9689	0.1179	0.9854	0.6427
8	First	0.9586	0.2770	0.9983	0.1723
	Last	0.9637	0.1214	0.9829	0.5834
9	First	0.9570	0.2270	0.9885	0.3335
	Last	0.9690	0.1153	0.9858	0.6343

Table 5.2: Metrics after poisoning one client at a time (t = 1).

5.2.2 Varied Client Poisoning

Table 5.3 displays a few different tests, with many clients suffering the poisoning in order to evaluate how the model reacts to that. As seen on the first row, having half of the clients poisoned is enough to drag the average and the performance of the model in relation to the recall to 0, since even though it's only half of the clients, that means classifying everything as 0 is being done more than any other distribution of classifications. This way of analyzing is proven right by even with only three of the clients being poisoned, the recall struggles to grow even, staying very close to 0 even after the last round.

Clients id	Round	Accuracy	Loss	NegRecall	Recall
0-5	First	0.952	0.211	0.1	0
	Last	0.952	0.2293	0.1	0
2,4,6	First	0.952	0.2134	1	0
	Last	0.9581	0.1300	0.9974	0.1795

Table 5.3: Metrics for attempting poisoning with a varied set of clients.

5.2.3 Already Implemented Attacks

Some attacks were already implemented in this environment for testing a different version of Data Poisoning attacks against it, but they also fit our security experiments descriptions, so they were also run. In table 5.4 we have the results of running all already implemented attacks, with the outlier rejection function enabled, as it was from the start. Poison model 7 is not included because it does not work with the current version of the implementation.

5.2.4 Result Analysis

Utilizing the results obtained and displayed by table 5.1 as a baseline for comparison, it is easy to conclude that every result obtained by poisoning only one client at a time, shown in table 5.2, is very impacted by such attack. The average recall obtained was 0.642, which represents a drop of 0.24, signalling a decrease of more than 27%, even though only 10% of the total data was corrupted (1 client in 10). This goes to show how impactful these Data Poisoning attacks can be, even with a small portion of the data being corrupted, being enough to invalidate the model usage in most cases. The varied set of clients table 5.3 proves how impactful increasing the

Poison Model	Round	Accuracy	Loss	NegRecall	Recall
1	First	0.9586	0.2448	0.9981	0.1745
	Last	0.9656	0.1239	0.9802	0.6765
2	First	0.9538	0.2255	0.9933	0.1712
	Last	0.9625	0.1258	0.9761	0.6925
3	First	0.9541	0.2456	0.9935	0.1734
	Last	0.9647	0.1214	0.9791	0.6787
4	First	0.9519	0.2709	0.9935	0.1734
	Last	0.9664	0.1183	0.9791	0.6787
5	First	0.9546	0.2573	0.9932	0.1900
	Last	0.9674	0.1212	0.9822	0.6726
6	First	0.9535	0.2503	0.9930	0.1717
	Last	0.9659	0.1205	0.9799	0.6892
8	First	0.9556	0.2436	0.9869	0.3341
	Last	0.9637	0.1229	0.9772	0.6964
9	First	0.9551	0.2534	0.9867	0.3274
	Last	0.9679	0.1177	0.9825	0.6781
10	First	0.9435	0.2429	0.9837	0.1463
	Last	0.9668	0.1212	0.9811	0.6831

Table 5.4: Metrics obtained by testing every already implemented attack.

number of clients taken by attackers can be when talking about a Federated Learning Machine Learning scenario, and how much they can skew the results. It is interesting also that by utilizing around three or more attacking clients, out of ten, even the outlier rejection function could be bypassed, and actually utilized for the attackers' benefit, as the attacking clients' attributes would create a strong average, and the function could consider honest client attributes as outliers, and reject them, creating an impact even bigger than if it wasn't there.

The results obtained from the already implemented attacks 5.4 demonstrate a similar impact on the recall metric, but it is worth mentioning that in this case, even though only one client is an attacker, this is an extra client, implying a 1 in 11 attackers, instead of a 1 in 10. Due to this, it being a bit worse of a result seems comparable to the ones discussed before. Another important factor is that these attacks were put against the `OutlierRejectFactory`, also implying a more robust attack, that can avoid such mechanisms of defence. All in all, these

already implemented attacks proved to be the most efficient attacks tested, but since the results itself is not that far from the one client-poisoned ones, this demonstrates how both methods of Data Poisoning, poisoning the training Dataset itself, or poisoning the aggregation attribute sent by a client after a local training round, are valid and effective vectors of performing such attacks.

Fitting to Our Game Description

The performed attack described in the sections above can easily be fitted into our Fix-FL2 game, described in chapter 4. D_T is represented by the entries marked for validation, L is the attribute c2, K is twenty, the number of rounds per attack test, and c is 0, the targeted label to be fixed. By following the description, for each round r, the function Round() would then always return 9 or 10 for n, depending on the attack, as described before, and 1 for t. D_1 to D_{n+t} are performed by the code separation of entries into parts, one for each client, with all training datasets being provided by the generate_train_datasets() function, with this one doubling up as part of the attacker function $\mathcal{A}()$, poisoning the training dataset of the adversary controlled client, for the attacks implemented for this work, while for the already implemented attacks, the manipulation of v_i is the only step performed by the attacker. The last step with the counting of entries classified as the target label, is converted to be about the recall metric, but it is the same logic. The main difference in the representation is the inability to train both the honest model and the poisoned model at the same time, but this can be solved by doing a non-poisoned evaluation, before or after the poisoned one, and comparing both. The fact that the recall can be visualized for each round, serves as the δ vector, where all the results of each round are stored and can be retrieved.

Figure 5.1 shows our game description with the variables substituted by concrete values related to the performed attack, when only one client was poisoned. It was considered that having the δ of the last round correspond to at least 10% of the number of total entries, was the victory condition. The abstract attacker function was substituted by the poisoning of the client's local dataset, where $D'_{i,r}$ is the same as $D_{i,r}$, but all labels c2 are set to 0.

Fix-FL2-Example($D_T, L = c2, K = 20, c = 0$):

for r in [1..20]:

$(n = 9, t = 1, D_1..D_{10}) \leftarrow Round()$

for i in [1..9]:

$v_{Hi} \leftarrow \Pi(m_{Hr-1}, D_{i,r})$

$v_i \leftarrow \Pi(m_{r-1}, D_{i,r})$

for i in [10]:

$D'_{i,r} \leftarrow\$ \mathcal{A}(m_{r-1}, D_{i,r})$

$v_i \leftarrow\$ \Pi(m_{r-1}, D'_{i,r})$

$m_{Hr} \leftarrow \mathsf{f}(v_{H1}..v_{H9}); m_r \leftarrow \mathsf{f}(v_1..v_{10})$

$res \leftarrow \mathsf{I}(m_{Hr}, D_T, c2); res' \leftarrow \mathsf{I}(m_r, D_T, c2)$

$s \leftarrow 0; s' \leftarrow 0$

for i in $[D_T]$:

$s \mathrel{+}= (res[i] == 0)$

$s' \mathrel{+}= (res'[i] == 0)$

$\delta[r] = s' - s$

$m_{Hr-1} \leftarrow m_{Hr}; m_{r-1} \leftarrow m_r$

Return $(\frac{\delta[20]}{|D_T|}) \geq 10\%)$

Figure 5.1: Fix attack performed, given the target classification 0 to be fixed, and over the span of 20 rounds. The δ evaluated at the final line only considers the last round results. The attack is considered a success if the increase of entries classified as 0 is at least 10%.

Chapter 6

Conclusion

Before discussing the conclusions of the work presented, it is worth mentioning that this research area is very vast, which in turn makes it harder to deeply analyze such topics in an MSc dissertation. This work tackles a definitional task that does not exist in a systemic format, and certainly much more work has to be done so final definitions that satisfy all dimensions of this theme can be drawn.

With those limitations in mind, the mapping of attacks presented in chapter 3 and the work of mapping Data Poisoning attacks to security games, presented in chapter 4 demonstrate a method for progressing. They serve as an example, that if we are willing to focus on a very concrete context, we are able to reach the same level of definitions as pseudo-code. Those definitions can then be instantiated, with the work provided, and **hopefully**, future work.

Lastly, the testing aspect of this work left much to be desired, as the time limitations and lack of access to more environments, to present a more varied approach of testing attacks had a great impact on this step. This is not to say that the environment utilized and the results presented are useless though, as they easily demonstrate, in the most simple way, the impact that Data Poisoning attacks can have in Machine Learning environments, even when done in the most crude and simplistic way possible, justifying worrying about this vector of attacks, and motivating development of methods of preventing against such attacks.

6.1 Future Work

Figuring out the future work to be done on this topic is very simple when utilizing the structure provided. Depending on whether the objective is to do a widening research on the topic or more of a deepening one, two different approaches can be had.

When talking about a broadening approach to expanding this work, one could apply the same logic shown in chapter 4 for each attack category and subcategory, describing general and more specialized security games, that fit into most existing real-world attacks. Now, if a future researcher takes upon the task of deepening this work, he could then set many different environments to test multiple existing attacks described in the literature and take the already described Data Poisoning games and those attacks to fit these games, perfecting the descriptions of the games in the process, adjusting them to be more realistic if necessary.

With the possibility of broadening and deepening this research, the amount of work available to be done, already mapped by the steps taken in this paper, is considerably big, could span many years and will never be finished completely, as other research in the area also expands the work to be done in parallel, complementing the work described.

Bibliography

[1] Hojjat Aghakhani, Lea Schönherr, Thorsten Eisenhofer, Dorothea Kolossa, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. Venomave: Targeted poisoning against speech recognition, 2023.

[2] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016. doi:10.1609/aaai.v30i1.10237.

[3] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Cryptology ePrint Archive, Paper 2000/025, 2000. https://eprint.iacr.org/2000/025.

[4] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 26–45, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN: 978-3-540-68462-6.

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines, 2013.

[6] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2018.

[7] Cláudia Brito, Pedro Ferreira, Bernardo Portela, Rui Oliveira, and João Paulo. Soteria: Preserving privacy in distributed machine learning. Cryptology ePrint Archive, Paper 2021/966, 2021. https://eprint.iacr.org/2021/966. doi:10.1145/3555776.3578591.

[8] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017.

[9] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. Disease prediction by machine learning over big data from healthcare communities. *IEEE Access*, 5:8869–8879, 2017. doi:10.1109/ACCESS.2017.2694446.

[10] European Commission, Content Directorate-General for Communications Networks, and Technology. *Ethics guidelines for trustworthy AI*. Publications Office, 2019. doi:doi/10.2759/346720.

[11] Matt Fredrikson, Somesh Jha, and Thomas 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, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery. ISBN: 9781450338325. doi:10.1145/2810103.2813677.

[12] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An End-to-End case study of personalized warfarin dosing. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 17–32, San Diego, CA, August 2014. USENIX Association. ISBN: 978-1-931971-15-7.

[13] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Mądry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1563–1580, 2023. doi:10.1109/TPAMI.2022.3162397.

[14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[15] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[16] Alejandro Guerra-Manzanares, L. Julian Lechuga Lopez, Michail Maniatakos, and Farah E. Shamout. Privacy-preserving machine learning for healthcare: Open challenges and future perspectives. In Hao Chen and Luyang Luo, editors, *Trustworthy Machine Learning for Healthcare*, pages 25–40, Cham, 2023. Springer Nature Switzerland. ISBN: 978-3-031-39539-0.

[17] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), aug 2018. ISSN: 0360-0300. doi:10.1145/3236009.

[18] Inken Hagestedt, Yang Zhang, Mathias Humbert, Pascal Berrang, Tang Haixu, Wang XiaoFeng, and Michael Backes. Mbeacon: Privacy-preserving beacons for dna methylation data. In *Annual Network and Distributed System Security Symposium*, February 2019.

[19] Wim Hardyns and Anneleen Rummens. Predictive policing as a new tool for law enforcement? recent developments and challenges. *European Journal on Criminal Policy and Research*, 24, 09 2018. doi:10.1007/s10610-017-9361-2.

[20] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 148–162, New York, NY, USA, 2019. Association for Computing Machinery. ISBN: 9781450376280. doi:10.1145/3359789.3359824.

[21] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Comput. Surv.*, 54(11s), sep 2022. ISSN: 0360-0300. doi:10.1145/3523273.

[22] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894. PMLR, 06–11 Aug 2017.

[23] Pahulpreet Singh Kohli and Shriya Arora. Application of machine learning in disease prediction. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–4, 2018. doi:10.1109/CCAA.2018.8777449.

[24] Jing Lin, Long Dang, Mohamed Rahouti, and Kaiqi Xiong. ML attack models: Adversarial attacks and data poisoning attacks. *CoRR*, abs/2112.02797, 2021.

[25] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. When machine learning meets privacy: A survey and outlook. *ACM Comput. Surv.*, 54(2), mar 2021. ISSN: 0360-0300. doi:10.1145/3436755.

[26] Yunhui Long, Vincent Bindschaedler, and Carl A. Gunter. Towards measuring membership privacy, 2017.

[27] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen. Understanding membership inferences on well-generalized learning models, 2018.

[28] Octavio Loyola-González. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE Access*, 7:154096–154113, 2019. doi:10.1109/ACCESS.2019.2949286.

[29] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015. doi:10.1609/aaai.v29i1.9569.

[30] Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 634–646, New York, NY, USA, 2018. Association for Computing Machinery. ISBN: 9781450356930. doi:10.1145/3243734.3243855.

[31] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753, 2019. doi:10.1109/SP.2019.00065.

[32] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016. doi:10.1109/EuroSP.2016.36.

[33] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY, USA, 2017. Association for Computing Machinery. ISBN: 9781450349444. doi:10.1145/3052973.3053009.

[34] Svetlana Pavlitska, Nico Lambing, and J. Marius Zöllner. Adversarial attacks on traffic sign recognition: A survey, 2023.

[35] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models, 2018.

[36] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *Machine Learning Definition and Basics*, pages 1–17. Springer International Publishing, Cham, 2019. ISBN: 978-3-030-15729-6. doi:10.1007/978-3-030-15729-6_1.

[37] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning, 2021.

[38] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Stealthy poisoning attacks on pca-based anomaly detectors. *SIGMETRICS Perform. Eval. Rev.*, 37(2):73–74, oct 2009. ISSN: 0163-5999. doi:10.1145/1639562.1639592.

[39] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. Updates-Leak: Data set inference and reconstruction attacks in online learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1291–1308. USENIX Association, August 2020. ISBN: 978-1-939133-17-5.

[40] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017. doi:10.1109/SP.2017.41.

[41] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *cryptology eprint archive*, 2004.

[42] Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 196–206, New York, NY, USA, 2019. Association for Computing Machinery. ISBN: 9781450362016. doi:10.1145/3292500.3330885.

[43] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[44] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX, August 2016. USENIX Association. ISBN: 978-1-931971-32-4.

[45] Jean-Baptiste Truong, Pratyush Maini, Robert J. Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4771–4780, June 2021.

[46] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52, 2018. doi:10.1109/SP.2018.00038.

[47] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021. ISSN: 0950-7051. doi:https://doi.org/10.1016/j.knosys.2021.106775.