POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Improving Poisoning Attacks against Banking Fraud Detection Systems

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Andrea Ventura**

Student ID: 953132
Advisor: Prof. Michele Carminati
Co-advisors: Prof. Stefano Zanero
Academic Year: 2021-22

Abstract

To counter the constantly increasing number of banking frauds, banks and financial institutions develop Data-Driven Fraud Detection Systems, which are advanced protection systems based on Machine Learning (ML) algorithms. Although automated Fraud Detection Systems have demonstrated excellent results, it has been proven that they can be deceived and corrupted through the use of Adversarial Machine Learning (AML) techniques, that aim to trick Artificial Intelligence (AI) models by providing deceptive and corrosive inputs. In particular, previous works have shown the FDSs vulnerabilities against evasion attacks, which interact with the test set of the Machine Learning model, and poisoning attacks, that manipulate the training set of the algorithm.

In this work, we extend and improve the application of poisoning attacks applied to the banking fraud domain. We present a novel approach to generating fraudulent samples based on the statistical analysis of past victims' transactions and we introduce ensembling techniques to create a reliable Oracle, i.e., a Machine Learning tool which validates the adversary's frauds. According to specific metrics, we evaluate the impact of poisoning attacks on eight models, i.e., Random Forest, XGBoost, Light Gradient Boosting, CatBoost, Support Vector Machine, Artificial Neural Networks, Logistic Regression, and Active Learning. We conduct our experiments in three different scenarios, that identify the attacker's knowledge about the target FDS: White Box (perfect knowledge), Grey Box (partial knowledge), and Black Box (no knowledge). The attacker can mount poisoning attacks by following three distinct strategies: poisoning the amount, poisoning the count, i.e., the number of transactions per iteration, or poisoning both. Each strategy presents a conservative and a greedy version, and it is evaluated for both weekly and bi-weekly update policy, i.e., how often the detectors are retrained in order to include new samples. Moreover, we provide a deep analysis of the feature regeneration process, that allows the adversary to change the features of the transactions during an attack.

Our experiments prove that our Oracle is extremely reliable, even in the Black Box scenario, where it is trained with just 50 features. Our Oracle allows the adversary to mount poisoning attacks without being noticed in different cases. In particular, we are able to keep the attack detection rate very low, sometimes zero, even with foreign frauds, which

have a higher suspicion level. Moreover, we show that poisoning only the amount is beneficial, especially against foreign users and detectors trained according a bi-weekly update policy. On the other hand, we point out how poisoning the count is more complicated and less cautious. In conclusion, our approach allows the attacker to steal a considerable amount of money even when he or she has no knowledge about the target system.

Keywords: Fraud Detection System, Poisoning Attacks, Adversarial Machine Learning

Abstract in lingua italiana

Per contrastare il costante incremento del numero di frodi bancarie, le banche e le istituzioni finanziarie sviluppano avanzati sistemi di rilevamento delle frodi, basati sul algoritmi di Machine Learning (ML). Sebbene i sistemi automatici di rilevamento di frodi abbiano ottenuto risultati eccellenti, è stato dimostrato che possono essere raggirati e corrotti tramite tecniche di Adversarial Machine Learning (ADL), che mirano a ingannare i modelli di intelligenza artificiale fornendo particolari input corrosivi. In particolare, precedenti lavori hanno dimostrato le vulnerabilità dei sistemi di rilevamento delle frodi bancarie contro attacchi di evasione e attacchi che mirano alla corruzione del sistema (i.e., poisoning attacks).

In questa tesi, estendiamo e ottimiziamo l'applicazione degli attacchi di poisoning applicati nel campo delle frodi bancarie. Presentiamo un nuovo approccio per generare transazioni fraudolente, basato su l'analisi statistica delle passate transazioni della vittima, e introduciamo tecniche di ensembling per creare un Oracolo affidabile (i.e., un sistema di Machine Learning che filtra le frodi di un attaccante). Secondo metriche specifiche, valutiamo l'impatto degli attacchi di poisoning su otto modelli, i.e., Random Forest, XGBoost, Light Gradient Boosting, CatBoost, Support Vector Machine, Artificial Neural Networks, Logistic Regression, and Active Learning. Conduciamo gli esperimenti in tre diversi scenari, che identificano la conoscenza dell'avversario riguardo al modello da attaccare: White Box (conoscenza perfetta), Grey Box (conoscenza parziale), e Black Box (conoscenza nulla). L'attaccante monta attacchi di poisoning seguendo tre distinte strategie: corruzione dell'importo, corruzione del numero di transazioni o corruzione di entrambi. Ogni strategia presenta una versione conservative e una piu' aggressiva, ed è valutata secondo le due diverse policy di aggiornamento, settimanale e bi-settimanale, che permettono ai sistemi antifrode di includere nei loro dataset nuove transazioni. Inoltre, forniamo un'analisi approfondita del processo di rigenerazione delle frodi, che permette all'avversario di cambiare gli attributi delle transazioni durante l'attacco.

I nostri esperimenti dimostrano che il nostro Oracolo è estremamente affidabile, anche in uno scenario Black Box, dove l'Oracolo è addestrato con solamente 50 attributi. Il nostro Oracolo permette all'avversario di creare attacchi di poisoning senza essere notato in di-

versi casi. In particolare, l'attaccante è in grado di mantenere tasso di rilevamento degli attacchi molto basso, talvolta zero, anche con frodi straniere, che hanno un piu' alto livello di sospetto. Inoltre, mostriamo che corrompere solamente l'importo delle transazioni è piu' conveniente, specialmente contro vittime straniere e sistemi di rilevamento addestrati secondo una policy di aggiornamento bi-settimanale. Contrariamente, sottolineamo come corrompere il numero di transazioni per iterazione è piu' complicato e meno cauto. In conclusione, il nostro approccio permette all'attaccante di rubare un importo considerevolmente alto anche quando non ha conoscenza riguardo al sistema bancario.

Parole chiave: Sistemi Antifrode, Attacchi di Poisoning, Adversarial Machine Learning

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 Background and Related Works **7**
1.1 Banking Fraud Detection Systems 7
1.2 Adversarial Machine Learning 9
1.3 Poisoning attacks 9
1.4 Goal and Motivation 13

2 Threat Model **15**
2.1 Adversary's goal 15
2.2 Adversary's knowledge 16
2.3 Adversary's capabilities 17

3 Dataset Analysis and Engineering **19**
3.1 Dataset Analysis 19
3.2 Fraud Generation Process 23
3.2.1 Customers Categorization 24
3.2.2 Synthetic Fraud Generation 25

4 Fraud Detection Systems: Tuning, Training, and Evaluation **29**
4.1 Selected Fraud Detection Systems 29
4.2 Feature Aggregation 31
4.3 Proportional Accuracy 33
4.4 Feature Selection 34

4.5 Hyperparameter Tuning . . . 36
4.6 Concept Drift and Update Policy . . . 37
4.7 Model Evaluation . . . 40

5 Poisoning Attack **45**
5.1 Assumptions . . . 45
5.2 Attack Approach Overview . . . 46
5.3 Scenario and Strategy . . . 48
5.3.1 Scenario . . . 48
5.3.2 Strategy . . . 48
5.4 Victim Selection . . . 50
5.5 Retrieval and Crafting . . . 51
5.5.1 Retrieval . . . 51
5.5.2 Crafting . . . 51
5.5.3 Timestamp Selection . . . 52
5.5.4 Amount Selection . . . 52
5.5.5 Count Selection . . . 55
5.5.6 Other Features Selection . . . 55
5.6 Oracle . . . 57
5.7 Regeneration Process . . . 57

6 Implementation Details **59**
6.1 System Architecture . . . 59
6.2 Run-Time Architecture . . . 60
6.3 Hardware and Software Architecture . . . 62

7 Experimental Evaulation **65**
7.1 Metrics . . . 65
7.2 Oracle Results . . . 66
7.3 Poisoning Process Results . . . 72
7.3.1 White Box . . . 79
7.3.2 Grey Box . . . 80
7.3.3 Black Box . . . 83
7.4 Regeneration Process Results . . . 85
7.4.1 White Box . . . 88
7.4.2 Grey Box . . . 89
7.4.3 Black Box . . . 91

8 Limitations and Future Works **93**
8.1 Limitations 93
8.2 Future works 94

9 Conclusions **95**

Bibliography **97**

List of Figures **101**

List of Tables **103**

List of Algorithms **105**

Introduction

The progressive digitalization of banking and financial institutions has led to the phenomenon of Home Banking, also known as Online Banking. It consists of an electronic payment system by which the customers can perform banking operations such as wire transfers and online payments. Although Italy is among the countries with the lowest adoption of e-banking, in the last years there was a consistent approach in that direction [1].

Consequently, banking and financial institutions must deal more and more with an increasing number of frauds; in addition, due to the wake of COVID-19, cybercriminals took advantage of the opportunity to profit from our dependence on technology. These criminals use phishing, spoofing, extortion, and various types of Internet-enabled fraud to target the most vulnerable part of society. In particular, the Internet Crime Complaint Center (IC3) has published a report which reveals that banking frauds are one of the most popular crime types and has shown that the trend is upward [2]. In the UK, the annual value of online banking fraud losses reaches a value of approximately 159.7 million British pounds in 2020, while in Italy the Financial Information Unit received 113.187 suspicious transaction reports, 7.9 percent more than in the previous year [3, 4].

In order to defend against fraudsters, banks and financial institutions had to develop advanced protection systems: in particular, Data-Driven Fraud Detection Systems have been preferred in opposition to Expert-based ones, thanks to their increasing detection power and cost efficiency. Most organizations still use rule-based systems as their primary tool to detect frauds, which are powerful against known patterns. However, rules are expensive to build, require continuous management, and can be easily bypassed by smart attackers who constantly update their strategies. This is why fraud analytics, based on machine learning, becomes necessary for fraud prevention and detection.

Although automated Fraud Detection Systems have demonstrated excellent results, it has been proven that machine learning algorithms can be deceived and corrupted. In order to compromise the correct functioning of these systems, Adversarial Machine Learning (**AML**) techniques are effectively used. Three different attacks can be performed against

Fraud Detection Systems. The first ones are evasion attacks, that violate the model integrity by crafting adversarial samples to deceive the target algorithm [5]. Then, we have poisoning attacks, that violate the correct functioning of the model and the goal is injecting malicious samples to corrupt the training set of the algorithm [6]. Finally, there are model extraction attacks, that violate confidentiality and aim to replicate the model without the need for the training set [7].

Banking detectors are periodically trained according to a specific update policy. After a certain amount of time, their training sets are updated in order to include more recent examples. In this way, they can continuously learn and then detect new patterns. FDSs which are constantly re-trained involve a consistent computational effort, while rare updates lead to underperforming. However, adversaries can exploit the re-training process in order to perform poisoning attacks. They craft fraudulent transactions which, if considered legitimate, are included in the training set that will be used for the learning task. Hence, an attacker can modify a user's spending pattern and mislead detectors into believing in a behavior change from the customer. It's an iterative process: the fraudster tries to increase at each iteration, based on the update policy, the amount, and the number of transactions, by crafting deceptive frauds. If they are evaluated as legit by the detector, they are included in the training set, and then, when re-training occurs, these malicious transactions are included in the detector's model. The attacker's final goal is to steal as much money as possible. Moreover, he or she tries to do that by adopting different strategies. The adversary can poison the transactions' amount, stealing money in a shorter time window and without the worry to be detected; alternatively, he or she can focus on the number of transactions performed at each iteration, being more cautious and decreasing the probability of being noted. Finally, he or she can adopt a standard approach, less stealthy but more effective, which consists of poisoning both amount and count. In order to mount poisoning attacks, the fraudster, based on his or her knowledge, studies the victim's spending profile, and tries to mimic it, creating misleading transactions that could have been executed by the customer himself or herself. Moreover, the adversary can control almost all the features that characterize a transaction, so that he or she can well replicate the user's spending behavior. In fact, the attacker makes use of phishing websites, in order to steal victims' sensitive information, or exploits Trojan Horses, which are malware that infect web browsers by changing web page contents without being noticed.

In this work, we focus on poisoning attacks against detection systems in the electronic banking fraud domain. In particular, we improve the results obtained by Monti [27], which is the first work in the context of poisoning attacks applied to FDSs. Our approach considers different degrees of knowledge about the target system. For instance, he or

she can ignore the algorithm, the update policy, and the training set. We present three scenarios, depending on the level of knowledge: White Box (perfect knowledge), Grey Box (partial knowledge), and Black Box (no knowledge). If the attacker doesn't know the target algorithm, he or she has to create an Oracle, a surrogate model that tries to reliably replicate the FDS and which validates crafted transactions. The Oracle states if a transaction could be submitted to the FDS or needs to be regenerated. The fraud regeneration process consists of changing some features' value, such as the amount or the IP address, until the fraud is accepted by the Oracle. If this happens, the crafted transactions are then subjected to the target FDS and, if it considers them legitimate, they are included in its updated training set. We present poisoning attacks against the most spread state-of-art banking fraud detection solutions: Random Forest, XGBoost, LightGB, CatBoost, Support Vector Machine, Artificial Neural Networks, Logistic Regression, and Active Learning. These detectors are trained according to two different update policies: weekly and biweekly, which suit our purposes and our datasets' length. We evaluate each model according to several performance indices, including a custom one that gives more importance to false negative examples than false positive ones. We impersonate an attacker, mount poisoning attacks against all the proposed systems, for each scenario, strategy, and update policy, and we study the results according to specific rates. Each attack is executed against 15 pseudo-random victims, which satisfy certain criteria, and it lasts up to eight weeks. Users are divided into two categories: national, which perform transactions towards Italy, and foreign, which execute more suspicious wire transfers to a foreign country.

Depending on the update policy, the attacker's knowledge, and the strategy, we achieve very heterogeneous results, which are evaluated according to specific metrics that assess the effectiveness of the attacks. In the White Box scenario, by exploiting a conservative strategy, the adversary can steal from about 975,000€ to more than 2,500,000€ with national frauds, while between 70,000€ and 175,000€ circa with foreign ones, against target machines with a bi-weekly policy update. In the same attack setting, Monti's frauds [27], targeting 30 victims, were able to steal up to 750,000€ from national users and up to 220,000€ from foreign ones. In the Grey Box scenario, where the attacker has partial knowledge about the banking FDS, the attacker is able to steal a great amount of money and keep at the same time the detection rate between 0% and 50% for both national and foreign frauds. It is an important improvement if we consider that in this scenario, in Monti's work [27], the frauds present a detection rate which is always higher than 58%. In the Black Box scenario, the fraudster achieves even better results, because he or she builds an Oracle according to a weekly update policy, to speed up the poisoning

process. Even if the attacks are less cautious, the adversary is able to mask himself or herself for more than 30 days (which is half of the attack time) for national frauds and between 3 and 30 days for foreign ones, since the detectors show a very heterogeneous behavior. Monti [27] decided to be more careful and trained the Oracle with a bi-weekly update policy, but the attacks were still detected in a very short time window, up to 30 days in the best case.

We summarize our contributions:

- We evaluate eight different fraud detection models and we assess their performances;
- We present a novel method for crafting fraudulent transactions, which is able to control a larger number of features with respect to [27];
- We show a novel approach to building a reliable Oracle, by combining multiple learners with an ensemble method. We study different ensembling solutions, we evaluate them and we explain which is the most suitable and why.
- We use a new transaction process, by which the adversary can generate several features during the attack and we deeply analyze which features are convenient to modify at runtime.
- We mount poisoning attacks against all the proposed models, for each scenario, strategy, and update policy. Then, we study models' behavior and how they react through specific rates.

We organize this work as follows:

- In Chapter 2 we explore Fraud Detection Systems applied to the banking fraud domain in previous works; then, we analyze generic Adversarial Machine Learning methods. Finally, we focus on the formal definition of poisoning attacks and we explain state-of-art poisoning approaches.
- In Chapter 3 we identify the threat model. We explain which are the adversary's goal, knowledge, and capabilities.
- In Chapter 4 we analyze our datasets, their relevant characteristics, and features. Then, we concentrate on the fraud generation process.
- In Chapter 5 we present the Fraud Detection Systems considered in this work and how they have been trained according to the concept drift and update policy.
- In Chapter 6 we give an explanation of the attack approach, by deeply inspecting all the steps composing the poisoning process.

- In Chapter 7, first we provide the implementation details of our work, showing the system and the run-time architecture. Then, we list all the tools, libraries, and programming languages used, and we explain the hardware setting where the experiments have been conducted.
- In Chapter 8 we list our final results and we accurately discuss them, according to specific metrics and considerations.
- In Chapter 9 we argue the limitations of our work and the possible focus of future works.
- In Chapter 10 we give the conclusions, by summarizing and analyzing the end point of our work.

1 | Background and Related Works

In this chapter, we provide an overview of the main concepts of Fraud Detection Systems and Adversarial Machine Learning. In particular, we focus on poisoning attacks: we explain how they work and their application in previous works.

1.1. Banking Fraud Detection Systems

In order to counter the constant increase of banking frauds, cybersecurity experts have developed Fraud Detection Systems based on Machine Learning algorithms. The purpose of these systems is to recognize which transactions are fraudulent, separating them from the legitimate ones. The interaction between a customer and a banking Fraud Detection System works as follows: a user performs a transaction, which is first aggregated and then standardized by the bank system. At this point, it is submitted to the FDS and, based on the outcome, the transaction is either executed or rejected. Figure 1.1 shows the typical flow of the transaction within a banking FDS.

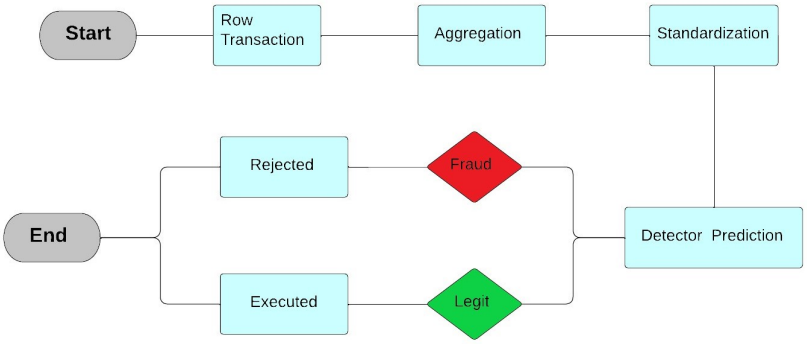

Figure 1.1: Transaction Flow

We can distinguish different approaches to face this challenge: unsupervised, supervised, and semi-supervised learning.

Descriptive model analytics (or unsupervised) aims at finding out frauds as elements that deviate from the normal behavior built on the historical transactions and they don't need label datasets.

David Weston et al [8] have proposed an unsupervised technique based on peer group analysis to detect banking transactions that deviate strongly from the norm.

Bolton et al [9] have shown how break point analysis works, an intra-account fraud detection method that compares with a t-test the spending pattern of a single user before and after the break point.

Sanchez et al [10] exploited association rule analysis intending to identify frequent relationships between transactions in order to define the norm of the data.

Vaishali [11] and Olszewski [12] focused on hierarchical clustering approaches applied to credit card fraud detection, respectively k-means, which deal with the distance between points and centroids, and self-organizing maps, which allow to visualize and automatically cluster high dimensional data on a low-dimensional space.

On the other hand, several supervised algorithms have been used in the banking fraud detection context: Random Forest [13], eXtreme Gradient Boosting [14, 15], Support Vector Machine [16, 17], Artificial Neural Network [17, 20] Catboost [18], Light Gradient Boosting [19] and Logistic Regression [20].

Finally, accredited works dealing with semi-supervised learning have achieved excellent results in the considered domain.

Amaretto [21] implements an active learning system combining both supervised and unsupervised learning since descriptive analytics allows to detect unknown patterns, while predictive analytics can automatically detect those patterns in the future.

FraudBuster [22] is an effective tool to detect frauds based on accurate modeling of the user's temporal profile: first, it builds the profiles, and then it computes the deviation of new transactions from the learned model.

Banksealer [23] ranks new transactions that deviate from the learned profiles, with an output that has an easily understandable and immediate statistical meaning.

FUZZGY [24] is a system that outputs an anomaly degree that explains how the new transaction is abnormal in comparison with the historical transactions pattern.

Jain et al [25] proposed a hybrid solution that consists of a preprocessing part using Rough sets theory in order to reduce data complexity, and then a J48 decision tree is trained for the classification task.

1.2. Adversarial Machine Learning

Adversarial Machine Learning (AML) is a machine learning method that aims to trick AI models by providing deceptive and corrosive input. Since Machine Learning have been widely explored in many areas, the spread and the study of Adversarial Machine Learning techniques have become really intense. In particular, more in-depth studies have been conducted in the image recognition area, where the goal is to modify the image in order to cause mispredictions of the classifier. However, it's known that Adversarial Machine Learning can be applied also in the banking fraud detection context [27–29]. In the literature, you can find popular adversarial attack methods, whose goal is to address all the weakness of Machine Learning models, trying to undermine and corrupt them.

Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is an algorithm discovered by Nocedal [30] for finding local extrema of functions, which are based on Newton's method of finding stationary points of functions. It uses less memory in comparison to the traditional BFGS and it was used in the Adversarial Machine Learning domain in order to attack power systems [31].

FastGradient Sign method (FGSM) is a gradient-based method that aims to generate adversarial examples. Introduced by Goodfellow et al [32], it exploits the gradients of a neural network to build an adversarial image.

Jacobian-based Saliency Map Attack (JSMA) tries to uses feature selection to minimize the number of features modified while causing miss-classification. It is computationally less convenient with respect to FGSM but it affects only a restricted number of features. Due to it's semplicity, it's really popular and it was used in different areas [31, 34, 35].

1.3. Poisoning attacks

Poisoning attacks aim to pollute the model's training data. Compared to evasion attacks, which interact only with the test set, poisoning ones try to tamper with the training set in order to decrease the accuracy of the target model. By slowly introducing malicious examples, they can impact the model's capacity to output correct predictions.

We can formally define poisoning attacks as a simple optimization problem [71]:

$$\underset{D_p}{\text{argmax}}(D, \Theta_p) \text{ s.t. } \Theta_p \in \underset{\Theta}{\text{argmin}}\ L(D \cup D_p, \Theta)$$

The attacker can add poisoned points to the training set and his or her objective is to maximize the training loss. On the other hand, the corrupted model Θ_p is learned by minimizing the loss function over both training and poisoned data. In order words, the adversary has to find the best set of malicious examples that, added to the training set, corrupt as much as possible the correct functioning of the model (i.e., maximizing the training loss). This definition suits the case in which the attacker has full knowledge of the training data D. In a more realistic scenario, this doesn't happen and D is approximated with a surrogate dataset D'. Therefore, the overall poisoning process is a challenging problem because the adversary has control over the objective which is implicit, and it is a bi-level optimization process, NP-hard in the general case.

The optimization-based approach is a popular solution to this optimization task. The idea is to start with a set of initial points and then run gradient descent on these points, which is just a greedy optimization approach consisting in iteratively updating each point in the direction that most improves the attacker objective [71].

It's important to consider a graphical evaluation of the poisoning process. In Figure 1.2 and Figure 1.3, we explain how a poison sample can compromise the output prediction of a machine learning model which deals with classification. In particular, we train a Support Vector Machine on a very small dataset with just two features.

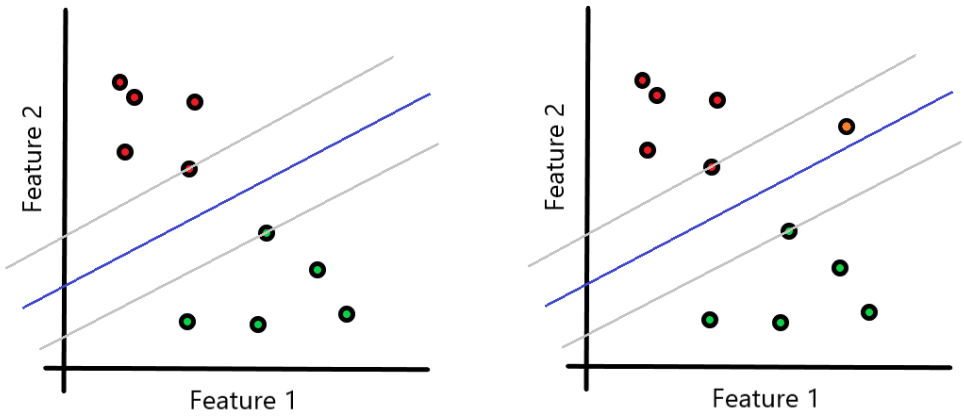

Figure 1.2: Trained SVM for a Classification Problem

In Figure 1.2, you see a trained SVM in which we specify the decision boundary between red and green classes and the margins. The model works well and it has not been com-

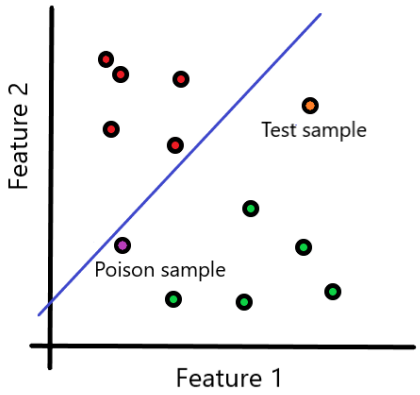

Figure 1.3: Compromised SVM after Injecting a Poison Sample

promised. If we add a test sample (in orange), it is correctly classified as a red point. In Figure 1.3, instead, you can notice that the introduction of a poison sample (in purple) shifts the decision boundaries, corrupting the functioning of the model which will predict the test point as a green one.

However, these concepts cannot really be applied to the fraud detection domain. The reason is that the adversary, in realistic scenarios, has not the full knowledge of the target system. He or she doesn't know either the algorithm or the training set. The interaction is indirect and the attacker can control only a reduced set of features. Hence, the fraudster needs to rely on a diverse approach, which consists of building an Oracle that tries to replicate the target model. It specifies if a crafted example can be submitted to the FDS or needs to be regenerated because too suspicious. The literature proposes different and effective solutions which deal with poisoning processes applied to different contexts.

Biggio et al [6] exploited a gradient ascent strategy to craft malicious examples and predict the change of the Support Vector Machine decision function. More specifically, the attacker aims at maximizing the test error by finding the optimal point $(x_c; y_c)$:

$$\max_{x_c} L(x_c) = \sum_{k=1}^{m} (1 - y_k * f_{xc}(x_k))$$

$L(x_c)$ is a non-convex objective function, therefore it's possible to optimize it thanks to a gradient ascent approach. Starting from an initial point $x_c^{(0)}$, it is then updated as $x_c^{(p)} = x_c^{(p-1)} * tu$, where p is the current iteration, t the step size and u the attack direction. By solving this optimization process, they were able to reduce SVM accuracy.

While [6] was about a specific machine learning model, Mozaffari-Kermani et al [72] propose a generic and algorithm-independent attack scheme. They add N' malicious instances to the original training set to create a manipulated dataset D' composed of $N + N'$ examples. In order to select the poisoned instances, they make use of an auxiliary algorithm that computes and studies statistics for each feature over the training set. This approach is very similar to the one that we propose in this work, because it's completely generic and works against any target machine learning model.

Suciu et al [39] introduced "StingRay", an attack that achieves poisoning but preserves overall classification performances. It consists of the following steps:

- The adversary selects a benign sample as the base instance;
- StingRay alters a not too suspicious subset of features, so that the model will classify it as legitimate;
- It filters crafted instances based on their negative impact, ensuring that their individual effect on the target prediction is negligible;
- The procedure is repeated until a specific test example is misclassified.

This approach achieves excellent results in four different classification tasks: image recognition, malware detection, Twitter-based exploit prediction, and data breach prediction. Moreover, three specific models were used: Convolutional Neural Network, linear Support Vector Machine, and Random Forest.

Chen et al [73] proposed "KuafuDet", an approach that tries to undermine poisoning attacks in the mobile malware context. First,they show how traditional machine learning models are weak against them, then they suggest a novel solution, which is divided into two learning phases. The first includes an offline training step that selects and extracts features from the training set, while the second one, the online detection phase, classifies large sets of online applications into two different categories, benign and malicious, with the help of three different models: Random Forest, Support Vector Machine and K-Nearest Neighbours. Finally, they make use of a Self-adaptive Learning scheme that discovers new information from both the identified malware and the filtered suspicious false negatives detected by a detector called "Camouflage Detector".

Monti's work [27] is the first one that really deals with poisoning attacks in the banking fraud detection domain. The author faced poisoning attacks with a novel approach to crafting malicious transactions and suggested the help of an Oracle during the process. He evaluated several ML-based fraud detection systems' reactions against this type of attack, for different strategies and scenarios.

1.4. Goal and Motivation

In this work, we aim to improve the results obtained by Monti [27], which is a strong reference point in the considered domain. More specifically, we enhance the approach to the Oracle, by involving simultaneously different models through ensembling methods. In addition, we evaluate more powerful poisoning attacks, because in our procedure the adversary has the possibility to manipulate a larger set of features. We also deepen the regeneration process performed by the Oracle, because we analyze for each model which are the attributes that the attacker needs to change more frequently in order to decrease the suspicion level. We propose different poisoning strategies, which separately focus on amount and count, the two features that the fraudster wants to poison most to steal more money. We analyze the differences and their impact on both national and foreign frauds, which show very diverse behaviors.

2 | Threat Model

In order to have a comprehensive understanding of this work, it's necessary to precisely define the threat model. This is why we rely on the attack taxonomy proposed in the literature [36–38].

2.1. Adversary's goal

The adversary has several goals depending on what he or she is looking for. Since we are in a poisoning scenario, the main objective is to corrupt the target algorithm. By injecting malicious samples into the training set, the attacker wants to modify the functioning of the model and, consequently, steal money from the users. However, the attack can go in two different directions: the adversary may want to steal much money as possible in a short window of time, without worrying about the model detection, or focus on crafting frauds in a way that the attack lasts as much as you can. In the first case, the fraudster tries to poison the amount of the victims' transactions while in the second he or she concentrates on poisoning the count of victims' spending pattern (i.e., the number of transactions per week). Finally, we propose a third strategy in which the adversary combines the previous directions, worrying only about maximizing the profit.

From a point of view of security properties, the attacker violates integrity, because he or she has to deceive the target model, and availability, as the correct functioning of the algorithm is compromised. In this work, we present a generic attack, because the victims are selected randomly among those that have specific requirements (i.e., a certain number of transactions). However, in a realistic scenario, an attacker can damage specific users or steal the information needed in a generic way. Finally, since we are in a fraud detection scenario, the attacker's aim is to poison the target model in order to consider fraudulent transactions legitimate.

2.2. Adversary's knowledge

The adversary's knowledge is a crucial point in banking fraud detection since in a realistic scenario the attacker can have different degrees of knowledge and this strongly impacts his or her final results. In order to deal with the adversary's knowledge, we rely on [27, 28, 39], which propose a setting that really suits our scope, by slightly modifying the representation of [37]. We list all the terms that represent the attacker's knowledge:

- Training Data Δ: training data on which the target model is trained;
- Features Set Φ: the set of features used to build the target algorithm;
- Target Algorithm A: the algorithm used to create the fraud detection system;
- Hyper-Parameters P: hyper-parameters used to train the machine learning model;
- Past User Transactions T: past users transactions to identify the user spending pattern;
- Update Policy Π: update policy of the target model (i.e., weekly or biweekly)

In conclusion, the adversary knowledge can be easily modeled with the following tuple: $\Theta = (\Delta, \Phi, A, P, T, \Pi)$.

Given these premises, we can identify three possible scenarios in which the attacks will be performed:

- White Box: the adversary has full knowledge about the detector and the victim's past transactions. Although it is the least realistic situation, it is introduced for two reasons: the attacks can be done by a bank's employee, which has the possibility to access the more reserved information and, moreover, it's interesting to study the poisoning process in the best case scenario for the attacker.

  $\Theta_{wb} = (\Delta, \Phi, A, P, T, \Pi)$

- Grey Box: the adversary has partial knowledge about the detector and the victim's past transactions. He or she knows only the features set and the update policy and uses a surrogate dataset to build the oracle; furthermore, the attacker has all the victim's transactions executed in the month before the attack.

  $\Theta_{gb} = (\delta, \Phi, \alpha, \rho, \tau, \Pi)$

- Black Box: the adversary doesn't know anything about the detector and has partial knowledge of the victim's past transactions. He or she ignores the features set,

the update policy, and the target algorithm and relies on one month user's past transactions history.

$\Theta_{bb} = (\delta, \phi, \alpha, \rho, \tau, \pi)$

2.3. Adversary's capabilities

In [28] the authors focus on evasion or exploratory attacks which craft adversarial samples in order to avoid the detection of the target model. In that scenario, the aim of the attacker is to manipulate the test set executing transactions on behalf of the user. In this work, we study causative or poisoning attacks, where the attacker tries to manipulate both the training and the test set. In fact, he or she wants to inject malicious examples into the training set so that the target model will be trained on a corrupted set of samples. In the beginning, the attacker, relying upon his or her knowledge, tries to mimic the spending pattern of the victim. Then, he or she poisons the amount and/or the count of the victim by incrementally raising the value and the number of transactions per week. In this way, the machine learning model, which is periodically updated, will assimilate the new fake spending pattern forged by the attacker, by considering the incoming transactions as legitimate. In order to access customers' sensitive data, an adversary can install malware on employees' devices or exploit phishing techniques in order to extract information, such as credentials and One Time Password (OTP), from the victims' bank accounts. In this way, an attacker can easily perform transactions on behalf of the user. Then, the features of the transaction are aggregated and standardized. In this work, the fraudster can manipulate only a set of features, that is larger than the one used in previous work [27, 28]. Obviously, the attacker can partially control the aggregation of the features, which also depend on past users' transactions.

3 | Dataset Analysis and Engineering

This chapter aims to analyze our datasets. In particular, we present the features that are relevant in the banking fraud detection context and we study the trend of the count and the amount during different time windows. Finally, we introduce our fraud generation process, which is necessary since our proposed detectors rely on supervised learning algorithms, that require labeled data.

3.1. Dataset Analysis

Thanks to a collaboration with a major bank, we have the possibility to work on two datasets composed of real European executed transactions. The first dataset presents transactions from 2012 to 2013, and the second one is composed of banking operations from 2014 to 2015. Both datasets contain many features that are preventable, this is why we select a subset that really interests us, so that we can work on a light and concise amount of data. Moreover, for privacy reasons and to protect personal users' information, some features are hashed: obviously, this compromises in no way our study. We list the considered relevant features:

- IP: hashed IP address of the connection associated with the transaction;
- IDSession: hashed unique value associated with a single session;
- Timestamp: date and time in which the transaction is executed;
- Amount: amount related to the transaction (€);
- UserID: hashed unique value identifying the user;
- IBAN: hashed value of the destination bank account of the transaction;
- ConfirmSMS: binary value which specifies if the transaction has been executed with a confirmation SMS or not;

- IBAN_CC: the Country Code of the beneficiary IBAN;
- CC_ASN: The Country Code and the Autonomous System Number from which the connection comes.

Once identified the fundamental features, we can reduce and clean the datasets. We have eliminated the duplicate transactions and we have kept all the transactions with invalid CC_ASN (i.e., 'n.d, n.d'). This is because CC_ASN is not a core feature as the amount or the timestamp, and it's an alphanumerical string whose value is not important, and an invalid CC_ASN is considered equal for each user. In Table 3.1, we present a general overview of the two datasets.

Dataset	Time Window	Users	Transactions	Mean (€)	Max-Min(€)
2012-13	01/12/12-10/09/13	53764	567550	1786.38	0.01-50000
2014-15	22/10/14-23/02/15	58507	471766	1778.99	0.01-50000

Table 3.1: General Information about the Datasets

The two datasets are really similar. The older one covers a larger time window and it has more transactions, but it has a smaller number of users. The amount mean is almost equal while the possible maximum and minimum amount is between 0.01€ and 50000€ for both datasets.

We show some plots in order to explain how the amount and the transactions are distributed over time periods. In order to understand our datasets, it's fundamental to clearly visualize the trends of the amount and the count with respect to the timestamp. We refer to the 2012-13 dataset, which is really similar to the more recent one, but it is composed of a greater number of transactions. In Figure 3.1 and Figure 3.2, we report the course of the count and the average transaction amount, grouped by days.

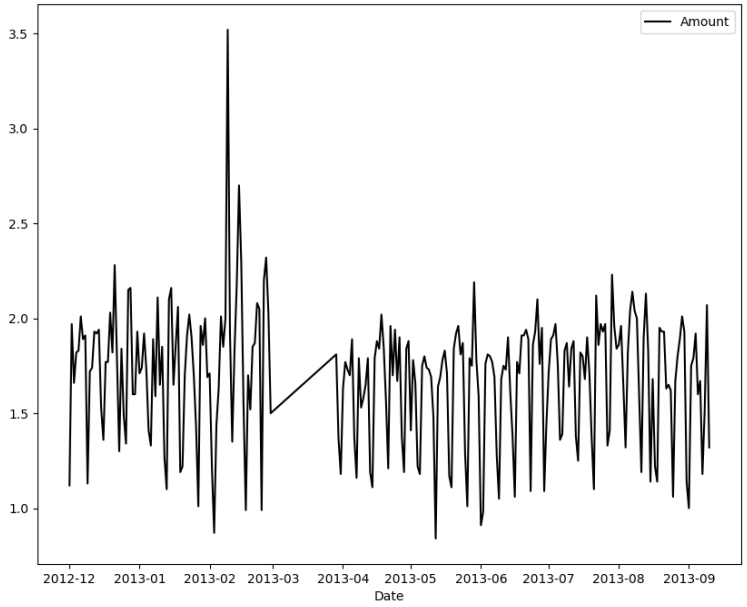

Figure 3.1: Mean Transaction Amount Per Day

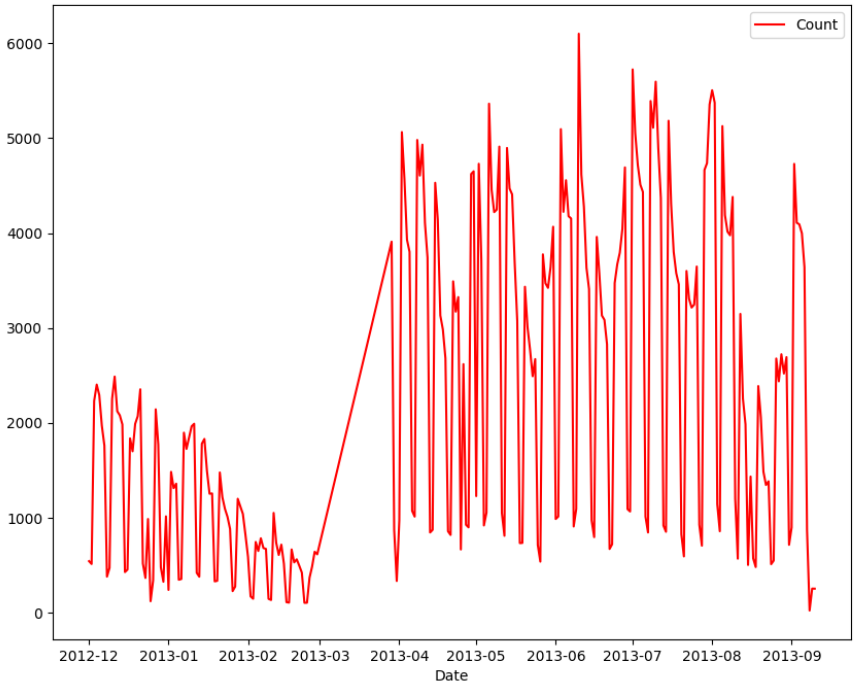

Figure 3.2: Count Per Day

In Figure 3.1, we notice that the average amount of the transactions is really unstable, but it presents the same pattern for all the dataset length. However, we have a peak in correspondence to February month. On the other hand, in Figure 3.2, the count plot allows us to understand that most of the transactions are executed during the summer and in February, where we register an average amount increase, we have instead the minimum number of operations.

In Figure 3.3 and Figure 3.4, we plot the users spending pattern during the weekdays.

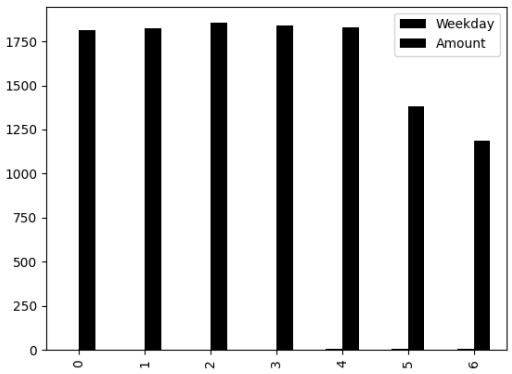

Figure 3.3: Mean Amount per Weekday

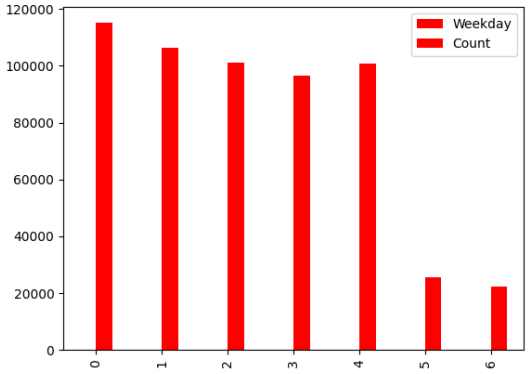

Figure 3.4: Count per Weekday

Figure 3.3 and Figure 3.4 state that users perform more transactions with a higher amount during the week, from Monday to Friday. On the weekend, we have a decrease in both count and amount. This aspect is very important for an attacker, because transfers executed Saturday or Sunday have a larger suspicious level and when crafting frauds, he or she will take into account it.

Finally, in Figure 3.5 and Figure 3.6, we provide histograms concerning the timestamp hours in which transactions are executed.

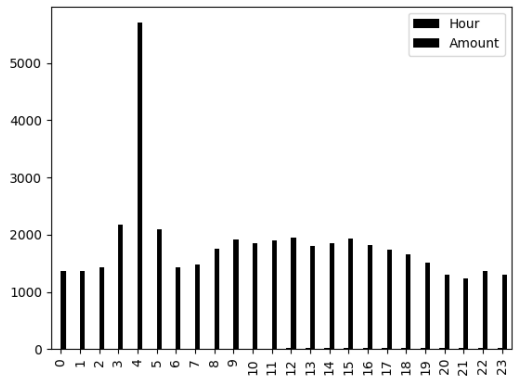

Figure 3.5: Mean Amount per Hour

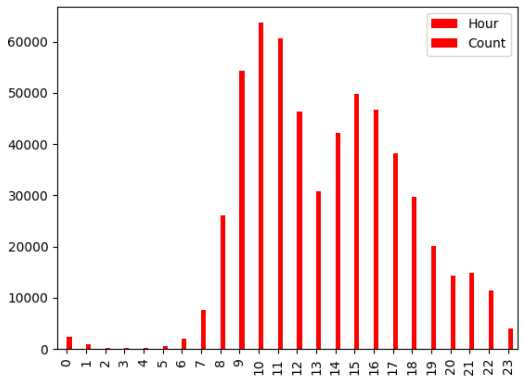

Figure 3.6: Count per Hour

Figure 3.6 shows that most of the transactions are executed during the day hours, from 7 am to 7 pm. When the attacker needs to choose an hour for creating a fraud, he or she selects this range, in order to deceive the detectors. Figure 3.5 illustrates that the average amount per transaction is constant during the day and it is about 2000€. However, we can observe an interesting peak registered at 4 am, where the average amounts to almost 6000€. On the other hand, from Figure 3.6, we understand that those transactions are very few. These examples could be considered outliers because they behave very differently from the norm. Usually, fraudulent transactions are executed during the night and they have a significant amount. These transactions follow exactly a fraudulent pattern, but they are legitimate, they are valid observations. These extreme but legitimate values that are very diverse from the rest of the population make the fraud detection task even more difficult, because effective detectors need to distinguish between particular but legitimate behaviors and very similar fraudulent ones.

3.2. Fraud Generation Process

The dataset 2012-13 has been completely cleaned from frauds, while the banking group made available a list of fraud reports concerning the dataset 2014-15. In particular, we have 606 frauds, corresponding to 0.128% of the entire dataset, the amount mean is 21,320€ and the victims' number is 96. Since we are in a supervised learning setting, in order to avoid bias toward the legitimate class, we need to craft fraud samples to effectively face the classification task for both datasets. We replicate fraudulent patterns and malicious behaviors and we need to rely on the available reports and previous works. More specifically, we can distinguish two fraudulent schemes: information stealing and transaction hijacking [23, 28]. In the information stealing scheme, the attacker, either

from data breaches or directly from the cardholders (e.g., via phishing websites or scam text messages) [40], is able to steal confidential data such as credentials and one time passwords (OTP) related to the targeted bank account. In this way, he or she has the necessary information to execute transactions on behalf of the victim. In the transaction hijacking pattern, which is less spread, adversaries install Trojans (e.g., ZeuS or Citadel) on bank customers' devices that are able to divert the transactions executed by the victim to a recipient IBAN at will [24].

In addition, in the banking fraud detection task, we need to deal with imbalanced datasets. The number of legitimate transactions is much greater that the number of fraudulent ones. According to the literature [43, 44], frauds usually constitute between 0.1% and 1% of the total transactions. In particular, we generate a frauds percentage which is close to 1% for both datasets, as Monti [27] and Carminati *et al.* [28] did. In order to synthesize fraudulent wire transfers, we exploit the same strategy used by Monti [27], with minor modifications.

3.2.1. Customers Categorization

The first phase in generating fraudulent transactions is to identify the victims. We divide the users into nine categories, more than Monti's work [27] because we want to involve all different types of users. A customer category is given by the transactions' amount and the number of transactions during the time window considered by the dataset.

Category	Amount Mean (€)	Count	Percentage
1	> 1500	$4 < \cdot < 20$	64%
2	$1500 < \cdot < 3000$	$4 < \cdot < 20$	9.97%
3	> 3000	$4 < \cdot < 20$	7.72%
4	> 1500	$20 < \cdot < 51$	7.96%
5	$1500 < \cdot < 3000$	$20 < \cdot < 51$	3.54%
6	> 3000	$20 < \cdot < 51$	1.9%
7	> 1500	> 51	2.19%
8	$1500 < \cdot < 3000$	> 51	1.6%
9	> 3000	> 51	1.12%

Table 3.2: Users' Category for 2014-15 Dataset

With this subdivision, we are able to cover all the possible spending patterns. In fact, in the real world, all people can be targeted by attackers, regardless of their spending behavior. The most consistent category is the one with the smallest amount mean and few transactions. On the other hand, users which perform many heavy wire transfers

represent a small set. In this work, we select around 1000 random victims for both datasets, based on the percentage shown in Table 4.2. Even if we have 606 reports for the more recent one, we apply the same generation strategy for both datasets. In addition, 82% victims are scammed according to information stealing pattern, while the remaining 18% refers to transaction hijacking scheme.

3.2.2. Synthetic Fraud Generation

In the banking fraud domain, information stealing is the most common scenario and the majority of victims are hit under this pattern. The adversary, with a phishing campaign or social engineering, retrieves the necessary information to impersonate the customer and execute transactions at will. In this way, the transfers are authorized by the attacker's device. This is why all the information related to the connection from which the transaction is performed, is related to the attacker. On the other hand, in a transaction hijacking scenario, the attacker exploits specific malware in order to hijack the operations performed by the user. All the data related to the transfer are associated with the victim. Table 3.3 provides a summary of how the features are chosen in the two different scenarios.

Scheme	IP	SessionID	IBAN	ConfirmSMS	IBAN_CC	CC_ASN
Information Stealing	Random	Random	Random Distribution	Dataset Distribution	Real Distribution	Dataset
Transaction Hijacking	Victim	Victim	Random	Victim	Real Distribution	Victim

Table 3.3: Features Values for Frauds Crafting

In the information stealing scheme, the IP address, the ID session, and the IBAN are random hashed strings, that depend on the attacker. The confirmation SMS is chosen according to its distribution in the dataset (85% 1 and 15% 0), as the CC_ASN, which the attacker can easily spoof using a VPN. For what concerns the IBAN Country Code, the Italian Ministry of Economy and Finance [45] stated that in 2020, the intensity of the foreign fraud phenomenon focused primarily on the United Kingdom, France, Spain, Germany, and Romania. Ergo, we assign the IBAN_CC with 40% of probability to these countries, the 40% to Italy, and the remaining 20% to all the other European countries present in our datasets. This probability distribution is very different from dataset one, and this is why foreign frauds represent a real challenge for fraudsters in the banking fraud detection world. In the hijacking transaction scheme, the only features that are changed by the attacker are the IBAN and the IBAN_CC, because the transaction itself (i.e., IP, SessionID, ConfirmSMS, CC_ASN) is executed by the victim.

Once we have introduced the general setting in which attackers usually act, we can address the different strategies that adversaries adopt in order to commit frauds. An attacker may want to execute a few fraudulent transactions, in a limited time window, but with a very high amount. On the contrary, he or she makes the attack last as long as possible, by stealing a small or medium amount of money. Alternatively, the adversary can observe a victim's behavior and study his or her spending pattern in order to deceive the fraud detection system. Finally, one can commit single frauds, performed just once against one specific user, with a high amount. However, the goal is always the same: stealing as much money as possible and remaining undetected. Therefore, we replicate these four strategies, for both information stealing and transaction hijacking scheme. For this purpose, we consider three variables that the attacker can directly control: number of frauds, amount of the transactions, and total duration of the attack (i.e., time window from the first to the last fraud), in which the frauds are equally distributed. Table 3.4 explains for each scenario the strategies and the fraudulent transactions implemented by potential adversaries.

Scheme	Typology	Percentage	Count	Amount (K €)	Duration (days)	Distribution
Information Stealing	Short-lived	10	1	35-50	-	Uniform
	Short-lived	8	1-5	10-25	14	Uniform
	Short-lived	10	1-5	10-25	7	Uniform
	Short-lived	2	24	1-10	1	Gaussian
	Short-lived	2	20-35	1-3	7	Uniform
	One-fraud	15	1	2-10	-	-
	Custom	15	1	$\max(5*\mu, \mu+2*\sigma)$ - $\max(7.5*\mu, \mu+2.5*\sigma)$	7	Uniform
	Custom	3	$5*count$	$\mu-$ $\max(1.5*\mu, \mu+0.5*\sigma)$	7	Custom
	Custom	3	$5*count$	$\mu-$ $\max(1.5*\mu, \mu+0.5*\sigma)$	14	Custom
	Custom	6	$3*count$	$\max(2*\mu, \mu+\sigma)$ - $\max(4*\mu, \mu+2*\sigma)$	14	Custom
	Long-lived	1	5-30	0.05-0.1	30	Uniform
	Long-lived	2	5-30	0.1-0.2	30	Uniform
	Long-lived	5	5-30	0.2-0.5	30	Uniform
Transaction Hijacking	Short-lived	4	1	35-50	-	Uniform
	Short-lived	3	1-5	10-20	-	Uniform
	Short-lived	1	15-25	2-5	-	Uniform
	Short-lived	2	-	2-5	14	Uniform
	One-fraud	5	1	1-10	-	-
	Long-lived	2	10-20	0.2-0.5	-	Uniform
	Long-lived	1	-	0.2-0.5	30	Uniform

Table 3.4: Generated Frauds

We have the four typologies introduced before: one-fraud, short-lived, long-lived, and custom. "Percentage" means the portion of the victims scammed by that typology. Under the heading "Count" and "Amount", we find the possible intervals in which the values are chosen. The duration, in days, indicates how long the attacks take. In particular, we name "short-lived" the frauds that last less than 30 days, while we use "long-lived" for attacks that last 30 days (i.e., the maximum possible time window). On the contrary, "one-fraud" stands for attacks that present just one fraud. Instead, "Distribution" suggests the statistical distribution according to which the amount between the intervals is selected. Notice that in one case, we have a Gaussian, hence the attacker performs a series of transactions following a normal distribution with the final goal to steal a certain amount of money. Regarding the custom attacks, it's important to underline that the count and the amount depend on the user's spending pattern and some parameters, different for each attack. Finally, in the transaction hijacking scheme, you see that an attacker can divert a certain number of transactions or hijack the transactions executed by the victim within a specific time window. Table 3.5 summarizes the results of the fraud generation process.

Dataset	IS frauds	TH frauds	Reported frauds	Total frauds	Frauds percentage
2012-13	3982	808	0	4790	0.85%
2014-25	4534	759	606	4899	1.15%

Table 3.5: Generation Frauds Results

4 | Fraud Detection Systems: Tuning, Training, and Evaluation

In this chapter, we prepare our data by applying all the common machine learning techniques that need to be faced before performing training and assessing the detectors' performance. After introducing the algorithms used to train our detectors, we focus on features aggregation, the procedure which allows us to contextualize each row transaction by adding information about historical information. Then, we apply hyperparameter optimization to find a set of hyperparameters that maximize the performance of our models and we perform the feature selection task to reduce the set of features that characterize each transaction. Moreover, we explain important concepts about how models are evaluated and about concept drift and update policy. Finally, we proceed with the periodical training of our proposed models and their evaluation according to a specific set of metrics.

4.1. Selected Fraud Detection Systems

In our work, we present 8 different models, all of which have been used in the banking fraud detection context.

The first algorithm for random forests (**RF**) was created in 1995 by Tin Kam Ho [46], but then it was extended in different versions [47]: it is an ensemble learning method for classification or regression, which creates a forest of decision trees. In classification tasks, the output is the one stated by most trees. Regarding the fraud detection domain, it was used by Xuan et al [13], which obtained important results by comparing two different random forest versions on a real dataset coming from an e-commerce company based in China.

EXtreme Gradient Boosting (**XGB**), designed by Chen [48], is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm,

which gives a prediction model in the form of an ensemble of weak prediction models, which are decision trees. Meng et al and Zhang et al proposed a fraud detection system based on XGBoost and evaluated carefully its performance on very imbalanced datasets [14, 15].

Light Gradient Boosting machine (**LGB**) is a gradient boosting machine learning framework that has been developed by Microsoft [49]. It is based on decision trees and its strong points are its speed in fitting, its low memory usage, performance, and scalability. This is why it has been applied to credit card detection systems by Taha et al [19], which demonstrated how LightGBM outperforms the other approaches in terms of accuracy.

Catboost (**CB**) is an efficient gradient boosting algorithm based on decision trees [51], proposed by Yandex experts, in order to deal with recommendation systems, self-driving cars, and weather predictions. Introducing a novel gradient boosting scheme, it is characterized by fast training and prediction, categorical features support, and overfitting reduction [52]. Yunlong [18] et al studied the Catboost model in the transaction fraud detection context and showed that their solution achieved an excellent ROC score.

Support Vector Machines (**SVM**) are one of the most popular and old supervised learning models, applied to both regression and classification. Introduced by Vapnik in 1995 [53], SVM tries to map each training example to one class, maximizing the gap between the two categories. Exploiting the well-known kernel trick, SVMs can efficiently perform a non-linear classification even in multidimensional feature spaces. Gyamfi et al [16] used the SVM model with Spark (SVM-S) in order to build normal and abnormal customer behaviors and to test the validity of incoming transactions, while Asha et al [17] deepened the study of SVM by comparing its performance to K-Nearest Neighbour (**KNN**) and Artificial Neural Networks in the occurrence of credit card frauds.

Artificial Neural Networks (**ANN**) are black-box methodologies that allow to model complex patterns and decision boundaries in your data. McCulloch et al [54] opened the subject by creating a computational model for neural networks but the first functional networks with many layers were introduced by Ivakhnenko et al [55]. ANNs are based on the concept of neurons: they take in input the data and multiply it by a weight, then they put it into a nonlinear transformation function (logistic regression). They are just a generalization of existing mathematical and statistical approaches. This machine learning technique was used by Asha et al [17] and Sahin et al [20] in order to develop effective anomaly detection systems.

Logistic regression (**LR**) is a statistical model which has been used since 1970 in many different research areas, such as medical and social contexts. It consists of an approach

that exploits the sigmoidal function in order to model the conditional probability of a class given one input. In [20], logistic regression was applied to a real banking dataset, but it has been outperformed by Artificial Neural Networks, in terms of frauds detected.

An active learning (**AL**) system combines both unsupervised and supervised techniques. First, a traditional anomaly detection model is trained in order to output the examples with higher anomaly scores. Then, these transactions are used to train a supervised model; in this way, we minimize the manual investigation process and we train the supervised model only on relevant examples, with the aim to minimize the false positives. Carminati et al [21] implemented an active learning system for Anti-Money Laundering (AML) in order to extract the strengths of unsupervised and supervised learning and proved that the proposed hybrid method worked better than other state-of-art solutions. In particular, to develop an effective active learning system, we make use of Isolation Forest (**IF**) as an anomaly detection model and CatBoost as the supervised one.

4.2. Feature Aggregation

Our datasets contain transactions described by precise characteristics, i.e., features. The attributes that we discussed in Section 3.1 are called direct because they individually contextualize the single-row transaction. In order to create a powerful model, we need to train the machine learning algorithms on a dataset that collects as much relevant information as possible. This is why direct features are not enough: we need to aggregate them in order to capture the user's spending pattern and his or her behavior in a certain time period. Aggregating the features allows us to summarize them into logical groups using statistical methods and help to define in a precise way user's profile since his or her past transactions history is analyzed. Moreover, among our features, we have categorical features (i.e IP, IBAN, IBAN_CC, CC_ASN) that cannot be aggregated. We need to move to the frequency domain: in practice, we transform categorical variables into "bins of a histogram". For instance, you keep track of how many times, in a certain amount of time, a user commits transactions from that specific IP address, and, in this way, the feature has become a number. In order to carry out the aggregation task, we rely on [28].

The directly derivable features are:

- Amount: the value of the transaction in euro (€), without any type of transformation;
- Time_x, Time_y: these two features are obtained directly from the Timestamp. Since time is cyclical and we have to deal with time differences, we need to encode

and transform it into a variable of two dimensions, using sine e cosine. t = $ts_h * 3600 + ts_{min} * 60 + ts_{sec}$, Time_x = $\cos \frac{t*2\pi}{86400}$, Time_y = $\sin \frac{t*2\pi}{86400}$;

- isInternationalTx: binary value which specifies if the transaction is international (i.e., if the Country Code is different from the IBAN_CC)
- isNationalIban: binary feature which states if the recipient IBAN is national (i.e., if IBAN_CC = 'IT');
- isNationalASN: binary value that indicates if the connection of the incoming transactions comes from a national IP address (i.e., CC_ASN contains 'IT');
- confirmSMS: binary value which means if the One Time Password was needed for the transaction;

The aggregated features are:

- group_function_time: an aggregation function is computed certain grouping user's transaction by group over a certain period of time.
  - The groups are: IP, IBAN, IBAN_CC, IDSession, CC_ASN, or Amount, if we want to consider all the transactions performed by a user;
  - The functions are:
    1. Sum: it computes the sum of the transactions amounts in the considered time window;
    2. Count: It counts the number of transactions in the considered time window;
    3. Mean: it computes the mean of the transactions amounts in the considered time window;
    4. Std: it computes the standard deviation of the transactions' amounts in the considered time window.
  - The time windows are: 1 hour, 1 day, 7 days, 14 days, 30 days, and 365 days if we want to consider all the transactions present in the dataset performed by that user.
- time_since_last_group: it specifies the time elapsed, in hours, since the last transaction among those obtained by grouping the transactions by user and by group;
- distance_from_mean_group_time: it represents the distance between the current transaction and the amount mean obtained grouping the user's past transactions by

group over a certain time period;

- is_new_IP: binary value which states if the IP address related to the transaction is used for the first time by the user
- is_new_IBAN: binary value which indicates if it's the first time that the user sends money to that IBAN;
- is_new_IBAN_CC: binary value which specifies if it's the first transaction performed by the user towards that IBAN Country Code;
- is_new_CC_ASN: binary value which says if it's the first transaction performed by the user from that country.

Thanks to this aggregation strategy, we are able to extract 196 numerical features, which capture the users' spending profile and allow the model to contextualize every single transaction.

It's essential to underline that now the features' values need to be standardized. In fact, the difference between two points that are characterized by not standardized dimensions will be biased by the feature with a higher value. This is a mandatory requirement in order to build effective machine learning models.

4.3. Proportional Accuracy

Banking datasets are known to be heavily imbalanced. The percentage of frauds with respect to all the transactions is usually from 0.1 to 1%. With our generation process, we have generated about 5000 malicious transfers for every dataset: still, the two target classes are consistently unbalanced. You may notice that in this scenario, we get poor results regarding the prediction of new observations, especially for the small class. Usually, experts which deal with this challenge make use of some helpful techniques, such as oversampling and undersampling. The former consists of the generation of additional cases, and copies of the minority class to increase their effect on the classifier while the latter is about decreasing the presence of samples belonging to the majority class. However, these strategies can lead to overfitting our training data, and that is something you need to avoid in machine learning contexts. On the other hand, we could use the Synthetic Minority Oversampling Technique (SMOTE), which creates new elements of the minority class by generating convex combinations of neighboring instances. However, SMOTE has different limitations: sample overlapping, noise interference, and blindness of neighbor selection [62]. In addition, we don't want to create unrealistic samples which

corrupt our data. For these reasons, we adopt another strategy in order to overcome this limitation, and we rely on a custom score, proposed by Monti [27], which assigns different weights to false positives and false negatives and which is able to avoid the drawbacks of common performance metrics, such as the accuracy. In fact, a poor model could classify every transaction as legitimate, by achieving an accuracy of about 99%. In financial institutions, false negatives have a greater cost with respect to false positives. However, false positives make the customer lose confidence in the banking institution; an article by Forbes [63], states that 40% of consumers in Europe left their banking institution which declined a transaction even if it was legitimate. In addition to the loss of customers, the reputation is irreparably damaged. On the contrary, false negatives translate directly into loss of money: a fraudulent transaction ranked as legitimate involves a waste of finances and a bank image ruining. Hence, we decided to give more importance to false negatives and train our models in a way that the number of these mispredictions is minimized. This is why we introduce a custom miss-classification cost metric:

$$miss\_prediction\_cost = FP + k * FN$$

With this equation, we are stating that a false negative weighs k times more than a false positive. However, the challenge is estimating the value of k: it depends on how much the two miss-predictions can cause damage to the bank. It should be really specific to each bank. We try to be much general as possible, by evaluating k according to our datasets. In particular, we model k as the ratio between the total legit transactions and the fraudulent ones, obtaining k equal to about 100. In conclusion, we can write the final equation of our proportional accuracy:

$$proportional\_accuracy = 1 - \frac{miss\_prediction\_cost}{FP+TN+k*(TP+FN)}$$

The proposed metric is considered our yardstick. Models with high proportional accuracy have a medium amount of false positives but a few one of false negatives. We compare machine learning algorithms with each other based on this metric and we consider better the models which have higher proportional accuracy.

4.4. Feature Selection

At this point, it's necessary to introduce an essential step in the machine learning domain: feature selection. This task consists of extracting from the entire set of features, those which best fit each specific model. In particular, feature selection brings different benefits [64]:

1. Complexity reduction: making our model less complex implies a probability decrease

of overfitting our data;

2. Computational advantages: having fewer features means shorter training times;
3. Model simplification: simpler models are easier to be interpreted and studied by experts and researchers.

A key point to well understanding feature selection is that there are features that are irrelevant or redundant. More specifically, we can have features that are strongly correlated to one another and can be removed [65]. There are different approaches to face the feature selection task: embedded, filter, and wrapper methods.

Embedded approaches are represented by algorithms that automatically perform feature selection when trained (i.e., Lasso Regression).

Filter methods consist of looking at one feature per time, trying to assess how much that feature can be predictive and can be correlated with respect to your target.

Wrapper solutions are more general: instead of applying an exhaustive search, comparing all the possible feature combinations, that would be the ideal solution, these methods try to reduce the complexity of the search, by using simple greedy algorithms, backward stepwise and forward stepwise selection.

In our work, we deal with a dataset made up of many transactions, each one characterized by a lot of features. Wrapper methods would be too computationally expensive. On the contrary, we can exploit filter solutions, by inspecting how much every feature impacts the true label. Since we are studying each feature individually, we are not able to capture the correlation between them, but at least we are sure to include features that are relevant to the target. In order to implement our strategy, we divide our dataset into training and test set. We train the model on the first one using one feature at a time and we analyze the performance on the test set, focusing on our proportional score. Then, we remove the features whose score is less than the average and we keep all the others. We repeat this process for every model so that each one has its own specific set of features. With this approach, we are able to decrease the feature number from 196 to about 80 for each algorithm, apart from Artificial Neural Networks, where we can exclude only 36 features. If we inspect the performances, we notice that on average we lose 0.05% of our proportional accuracy, an acceptable percentage. Moreover, we realize that by reducing the feature set, the models tend to increase the number of miss-predictions in terms of false positives, not of false negatives. This is an important observation, which suits our work because despite having a certain cost, false positives make the attacks still more difficult. In Table 4.1 we provide first the set of the features which are shared by all the

models, then we list the features which are specific to each one.

Model	Features
Shared	IBAN_std_14d, Amount, distance_from_mean_IBAN_30d, IBAN_sum_365d, IP_sum_7d, isInternationalTX, CC_ASN_mean_30d, IBAN_mean_1h, IBAN_CC_mean_30d, IDSession_mean_1h, CC_ASN_sum_7d, IP_mean_30d, IBAN_CC_mean_365d, IBAN_mean_14d, IBAN_std_30d, IBAN_CC_mean_1h, IP_mean_1d, IBAN_CC_count_7d, IBAN_CC_mean_7d, IBAN_mean_1d, IBAN_count_1d, Amount_count_14d, CC_ASN_count_7d, time_since_last_tx_global, IBAN_count_30d, CC_ASN_mean_7d, IBAN_CC_mean_1d, time_since_last_IBAN_CC, IBAN_CC_sum_1d, IBAN_sum_1d, Amount_mean_7d, IBAN_sum_1h, CC_ASN_mean_1d, IBAN_mean_30d, CC_ASN_mean_1h, Amount_sum_7d, Amount_mean_1d, IBAN_mean_365d, IBAN_std_1d, IBAN_CC_sum_7d, IBAN_CC_mean_14d, Amount_count_7d, Amount_sum_1d, IBAN_count_14d, IP_sum_1d, IBAN_count_7d, IP_mean_14d, IP_mean_7d, IBAN_count_365d, CC_ASN_mean_14d, IBAN_mean_7d, IBAN_sum_30d, time_since_last_CC_ASN, IP_mean_365d, IBAN_std_7d, IBAN_sum_14d, Amount_mean_14d, IDSessione_sum_1h, IBAN_sum_7d, IP_mean_1h, CC_ASN_sum_1d
Light Gradient Boosting	Amount_mean_1h, Amount_std_7d, IP_std_7d, IP_std_14d, IP_std_30d, IP_sum_14d, IP_count_7d, IP_count_365d, IBAN_std_365d, IBAN_CC_std_7d, IBAN_CC_std_14d, CC_ASN_mean_365d, CC_ASN_std_7d, CC_ASN_count_14d, distance_from_mean_Amount_7d, distance_from_mean_Amount_14d, distance_from_mean_Amount_30d, distance_from_mean_IP_7d, distance_from_mean_IBAN_1d, distance_from_mean_IBAN_7d, distance_from_mean_IBAN_14d, distance_from_mean_IBAN_365d, distance_from_mean_IBAN_CC_7d, distance_from_mean_CC_ASN_7d, time_since_last_IP, time_since_last_IBAN, Time_x, isNationalIban
CatBoost	Amount_mean_1h, Amount_std_7d, Amount_sum_14d, IP_std_7d, IP_std_14d, IP_sum_14d, IP_count_7d, IBAN_std_365d, IBAN_CC_std_7d, IBAN_CC_std_14d, CC_ASN_std_7d, CC_ASN_std_14d, distance_from_mean_Amount_7d, distance_from_mean_Amount_14d, distance_from_mean_Amount_30d, distance_from_mean_IP_7d, distance_from_mean_IBAN_1d, distance_from_mean_IBAN_7d, distance_from_mean_IBAN_14d, distance_from_mean_IBAN_365d, distance_from_mean_CC_ASN_7d, time_since_last_IP, time_since_last_IBAN, Time_x, isNationalIban
EXtreme Gradient Boosting	Amount_mean_1h, Amount_std_7d, IP_std_7d, IP_std_14d, IP_sum_14d, IP_sum_365d, IP_count_7d, IBAN_std_365d, IBAN_CC_std_7d, CC_ASN_std_7d, CC_ASN_std_14d, distance_from_mean_Amount_7d, distance_from_mean_Amount_14d, distance_from_mean_IP_7d, distance_from_mean_IP_14d, distance_from_mean_IBAN_1d, distance_from_mean_IBAN_7d, distance_from_mean_IBAN_14d, distance_from_mean_IBAN_365d, distance_from_mean_IBAN_CC_7d, distance_from_mean_CC_ASN_7d, time_since_last_IP, time_since_last_IBAN, Time_x, isNationalIban
Logistic Regression	Amount_mean_30d, Amount_sum_1h, Amount_sum_14d, Amount_sum_30d, IP_sum_1h, IP_sum_14d, IP_count_7d, IP_count_14d, IBAN_std_1h, IBAN_count_1h, IBAN_CC_sum_1h, IBAN_CC_sum_14d, IBAN_CC_count_14d, IBAN_CC_count_365d, CC_ASN_mean_365d, CC_ASN_sum_1h, CC_ASN_sum_14d, CC_ASN_sum_30d, CC_ASN_count_14d, CC_ASN_count_365d, distance_from_mean_Amount_30d, distance_from_mean_Amount_365d, distance_from_mean_IBAN_365d, time_since_last_IDSessione, Time_x, isNationalIban
Random Forest	Amount_mean_1h, Amount_std_7d, Amount_sum_14d, IP_std_7d, IP_std_14d, IP_sum_14d, IP_sum_365d, IP_count_7d, IP_count_14d, IP_count_365d, IBAN_std_365d, IBAN_CC_std_7d, IBAN_CC_count_14d, CC_ASN_std_7d, CC_ASN_count_14d, distance_from_mean_Amount_7d, distance_from_mean_IP_7d, distance_from_mean_IBAN_1d, distance_from_mean_IBAN_7d, distance_from_mean_IBAN_14d, distance_from_mean_IBAN_365d, distance_from_mean_CC_ASN_7d, time_since_last_IP, time_since_last_IBAN, isNationalIban
Support Vector Machine	Amount_mean_1h, Amount_mean_30d, Amount_mean_365d, Amount_sum_1h, Amount_sum_14d, Amount_count_365d, IP_sum_1h, IP_sum_14d, IP_count_365d, IBAN_std_1h, IBAN_count_1h, IBAN_CC_sum_1h, IBAN_CC_sum_14d, IBAN_CC_count_14d, IBAN_CC_count_365d, CC_ASN_mean_365d, CC_ASN_sum_1h, CC_ASN_sum_14d, CC_ASN_count_14d, CC_ASN_count_365d, distance_from_mean_Amount_365d, distance_from_mean_IBAN_7d, distance_from_mean_IBAN_14d, distance_from_mean_IBAN_CC_365d, distance_from_mean_CC_ASN_365d, time_since_last_IP, Time_x
Artificial Neural Networks	Amount_mean_1h, Amount_mean_30d, Amount_mean_365d, Amount_std_1h, Amount_std_1d, Amount_std_7d, Amount_std_14d, Amount_std_30d, Amount_std_365d, Amount_sum_1h, Amount_sum_14d, Amount_sum_30d, Amount_sum_365d, Amount_count_1h, Amount_count_1d, Amount_count_30d, Amount_count_365d, IP_std_1h, IP_std_1d, IP_std_7d, IP_std_14d, IP_std_30d, IP_std_365d, IP_sum_1h, IP_sum_30d, IP_sum_365d, IP_count_1h, IP_count_1d, IP_count_7d, IP_count_14d, IP_count_30d, IP_count_365d, IDSession_std_1h, IDSession_count_1h, IBAN_std_1h, IBAN_std_365d, IBAN_CC_std_1h, IBAN_CC_std_1d, IBAN_CC_std_7d, IBAN_CC_std_14d, IBAN_CC_std_30d, IBAN_CC_std_365d, IBAN_CC_sum_1h, IBAN_CC_sum_14d, IBAN_CC_sum_30d, IBAN_CC_sum_365d, IBAN_CC_count_1h, IBAN_CC_count_1d, IBAN_CC_count_14d, IBAN_CC_count_30d, IBAN_CC_count_365d, CC_ASN_mean_365d, CC_ASN_std_1h, CC_ASN_std_1d, CC_ASN_std_7d, CC_ASN_std_14d, CC_ASN_std_30d, CC_ASN_std_365d, CC_ASN_sum_1h, CC_ASN_sum_14d, CC_ASN_sum_30d, CC_ASN_sum_365d, CC_ASN_count_1h, CC_ASN_count_1d, CC_ASN_count_14d, CC_ASN_count_30d, CC_ASN_count_365d, distance_from_mean_Amount_7d, distance_from_mean_Amount_14d, distance_from_mean_Amount_30d, distance_from_mean_Amount_365d, distance_from_mean_IP_7d, distance_from_mean_IP_14d, distance_from_mean_IP_365d, distance_from_mean_IBAN_1h, distance_from_mean_IBAN_1d, distance_from_mean_IBAN_7d, distance_from_mean_IBAN_14d, distance_from_mean_IBAN_365d, distance_from_mean_IBAN_CC_1d, distance_from_mean_IBAN_CC_7d, distance_from_mean_IBAN_CC_14d, distance_from_mean_IBAN_CC_30d, distance_from_mean_IBAN_CC_365d, distance_from_mean_CC_ASN_1d, distance_from_mean_CC_ASN_14d, time_since_last_IP, time_since_last_IDSessione, time_since_last_IBAN, Time_x, Time_y, isNationalIban, isNationalASN, is_new_CC_ASN, is_new_IBAN, is_new_IBAN_CC, is_new_IP, ConfirmSMS

Table 4.1: Features Selected

4.5. Hyperparameter Tuning

In the machine learning domain, hyperparameter optimization, or tuning, is a key task whose purpose is to find the best set of hyperparameters for the current algorithm. They have a very strong impact on the learning process and its performance. While parameters

are intrinsic to the model, hyperparameters are external and cannot be estimated from the trained data. The goal of hyperparameter tuning is to find a tuple that entails the model to reduce the loss function on the available data. The literature proposes several solutions to face this task, which is crucial due to the impossibility of estimating the best set of hyperparameters in advance. The most popular approach is called Grid Search, which is an exhaustive search through a manually specified subset of the hyperparameter space. The cross-validation technique is used in order to assess the analysis. On the other hand, we can have a Random Grid Search, which avoids inspecting every possible hyperparameter set by studying only a specified number of randomly selected combinations [57]. Moreover, in the hyperparameter tuning field, it's has been introduced the Bayesian optimization, which is a global optimization method for black-box algorithms based on a probabilistic model that incrementally evaluates and updates the hyperparameter configuration. This solution works well only in a few cases with specific learning algorithms [58]. Finally, you may exploit the gradient descent in order to optimize the hyperparameters setting. However, also this approach can be applied to a reduced number of models, such as Neural Networks [59], Support Vector Machine [60] and Logistic Regression [61].

Since we present very different models, based on very different concepts, we use a Grid Search strategy for each one, in order to tune at best the hyperparameters. In particular, since we need to explore a hyperparameters space that in some cases is very large, we adopt a Random Grid Search solution, according to which 30 different combinations of hyperparameters are evaluated with a k-fold cross-validation strategy. Larger values for k results in higher variance and higher running time, as training folds will be closer to the total dataset. Since we deal with very large datasets and we look for a model which gives the most accurate predictions on incoming transactions (i.e., we want a test error as small as possible), we choose k equal to 3. During the hyperparameter tuning task, for each model, we keep the hyperparameter set that performs best on our datasets, according to a proportional accuracy that we introduced in Section 4.3. We execute this task twice: first, with all the 196 features, then with the features specific to each model. In fact, after having reduced the number of features by more than 50% on average, the optimal hyperparameter set could be different from the previous one.

4.6. Concept Drift and Update Policy

Before evaluating the final performance indices of our detectors, we need to introduce some important concepts related to the banking fraud detection world.

In predictive analytics, the statistical properties of the target variable can change over

Model	Hyperparameter	Value
Light Gradient Boosting	reg_lambda	200
	reg_alpha	1
	objective	binary
	num_leaves	10
	n_estimators	200
	max_depth	32
	learning_rate	0.05
	boosting_type	gbdt
CatBoost	learning_rate	0.1
	l2_leaf_reg	1
	iterations	250
	depth	4
EXtreme Gradient Boosting	reg_lambda	0.4
	reg_alpha	200
	n_estimators	300
	max_depth	12
	learning_rate	0.1
	gamma	100
	booster	gbtree
Logistic Regression	tol	0.0001
	solver	saga
	penalty	l2
	max_iter	2500
	class_weight	balanced
	C	1
Random Forest	n_estimators	800
	min_samples_split	2
	min_samples_leaf	4
	max_features	auto
	max_depth	10
	criterion	gini
	class_weight	balanced
	bootstrap	true
Support Vector Machine	tol	0.001
	penalty	l2
	max_iter	5000
	loss	hinge
	C	0.1
Artificial Neural Networks	sl_neurons	64
	fl_neurons	128
	epochs	80
	dropout_rate	0.1
	batch_size	1024
	activation_funct	relu

Table 4.2: Hyperparameters Selected

time, and models which assume a static relationship between input and output can result in poor performance. This concept is known as concept drift, and in fraud detection applications it's very critical. Many aspects that can influence the customers' spending pattern in the short or the long term, such as holiday periods, the purchase of a car, or the birth of a child. These are all details to which a powerful detector has to pay attention. To deal with concept drift, past works adopted some countermeasures [23]. In this thesis, during the training, we assign different weights to the examples based on when they happened: more recent transactions contribute more than old ones. More specifically, we make use of a decreasing exponential function, which assigns more weight to newer examples:

$$weights = e^{-\frac{t}{k}}$$

The term "t" specifies the time, in hours, between the training and the transactions, while the term "k" models how fast the weight decreases among them. With this solution, our detection systems give more importance to the latest spending behavior with respect to the older one. Given that our datasets last about half a year, we estimate the value of "k" at 4380, which represents the hours' number contained in six months.

The update policy is a key concept of our work. Bank institutions need to update their detectors to assimilate new users spending behaviors. In this way, they are always updated and capable to learn new patterns. Detection systems incrementally increase their knowledge and their skill to predict incoming transactions. On the other hand, systems updates are very crucial because allow attackers to perform poisoning attacks: by injecting malicious transactions, fraudsters corrupt training processes and systems knowledge. Since the training phase is computationally demanding and the time period covered by our datasets, we assume two possible update policies, weekly and biweekly.

In conclusion, as Figure 4.1 explains, we train our models in a periodical way. We split the dataset into two equal parts. Each model is trained on the first one, while the second represents the test set. However, the predictions of the test set are made incrementally according to the update policy. In other words, the trained models predict one-week transactions (or two weeks), then, after saving the results, they include in their training set these just predicted transactions and repeat the same process until the dataset ends. Finally, we combine all the obtained results.

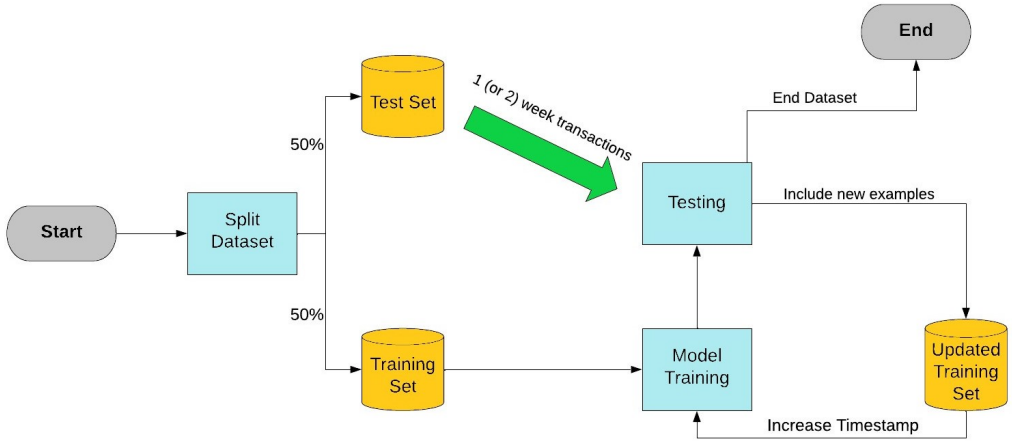

Figure 4.1: Training Process Flow

4.7. Model Evaluation

Once trained the models, we need to evaluate them. We exploit traditional performance metrics and, as explained in Section 4.3, a custom one, which suits our purposes. We report here the scores used:

- Precision (or positive predictive value): it is the fraction between the true positive examples among the positive ones.

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

- Recall (or sensitivity): it is the fraction between the true positive examples and the true ones.

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

- F1-Score: it is the weighted average of Precision and Recall.

$$\text{F1-Score} = 2 * \frac{2TP}{2TP+FP+FN}$$

- F2-Score: it is the weighted average of Precision and Recall, with more weight assigned to Recall.

$$\text{F2-Score} = \frac{TP}{TP+0.2*FP+0.8*FN}$$

- False Positive Rate (or fall-out): it is the probability of falsely rejecting the null hypothesis, i.e., the ratio between the false positive examples and the total number of actual negative ones.

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

- Matthews Correlation Coefficient (MCC): it is a quality measure of binary classifications. It takes into account true and false positives and negatives. It is a coefficient between -1 and +1, where +1 represents a perfect prediction, 0 an average random prediction, and -1 an inverse prediction. In this work, we use a weighted Matthews correlation coefficient since the strong unbalancing of our datasets.

$$\text{Weighted MCC} = \frac{w_1 * TP * w_0 TN - w_0 FP * w_1 FN}{\sqrt{(w_1 TP + w_0 FP) * w_1 (TP+FN) w_0 * (TN+FP) * (w_0 TN + w_1 FN)}},$$
$$w_0 = \frac{TP+FN}{TP+FN+FP+TN},\ w_1 = \frac{TN+FP}{TP+FN+FP+TN}$$

- Area Under the Curve of Receiver Operator Characteristic: it is a probability curve that plots the Recall against the False Positive Rate at various threshold values. If it's 1, then the classifier is able to perfectly distinguish between all the positive and the negative class examples, while if it's 0, the classifier would be predicting all negatives as positives and all positives as negatives.

$$\text{ROC-AUC} = \int_0^1 \frac{Recall}{FPR(x)} dx$$

- Area Under the Curve of Precision-Recall Curve: it is a probability curve that plots the Precision against the Recall at various threshold values.

$$\text{PRC-AUC} = \int_0^1 \frac{Precision}{Recall(x)} dx$$

In Table 4.3, we report performance metrics of detectors trained according to a weekly update. We omit results following a biweekly update policy because are very similar, but slightly poorer because the learning process is slower.

Model	P-acc	Precision	Recall	F1	F2	FPR	W-MCC	ROC-AUC	PRC-AUC
LightGB	97.99	23.96	97.89	38.50	60.53	1.91	95.98	99.84	60.32
CatBoost	98.78	32.36	98.84	48.75	70.05	1.27	97.56	99.92	64.99
XGBoost	98.52	28.31	98.58	43.99	65.88	1.54	97.04	99.90	62.84
AL	99.27	39.65	99.48	56.70	76.42	0.93	98.55	99.96	68.95
LR	95.97	21.42	94.06	34.89	56.04	2.13	91.99	99.22	57.14
RF	97.74	28.44	96.99	43.98	65.44	1.50	95.49	99.81	62.10
SVM	88.74	3.2	95.18	6.2	14.13	17.70	78.12	94.27	48.59
ANN	93.64	94.92	87.31	90.96	88.74	0.04	80.15	96.31	52.11

Table 4.3: Fraud Detection Systems Metrics, Weekly Update

The first consideration is that Active Learning performs better than the others, achieving 99.27% in proportional accuracy. Then, we have CatBoost, XGBoost, LightGB, Random

Forest, Logistic Regression, Artificial Neural Networks, and finally Support Vector Machine. In general, our custom metric is above 93.64%, except for SVM, which is the less powerful detector. In fact, SVMs results present a lot of false positives. This is why all the metrics are in general so low, in particular the Precision and the $F1$-Score. This is not an important issue in this work, because detectors with many false positives will hinder the poisoning attacks more effectively. On the other hand, ANN model achieves the opposite result, because it presents a balance between false positives and false negatives and hence it shows a good performance for each considered metric.

The precision is very low in general and the reason is that models tend to have a large number of false positives, trying to minimize the negative ones. They have been trained according to our Proportional Accuracy, which gives more importance to false negatives. Hence, it's obvious that the precisions are low.

The Recall is very high for all models. Since we try to avoid false negatives, we have that the percentage ratio is large. On average, the Recall is 95.67%, meaning that 95 fraudulent transactions out of 100 are detected.

$F1$-Score is a very important metric because it is really useful when you have an uneven class distribution and a different cost between false positives and false negatives. On the other hand, $F2$-Score works as $F1$, but it weights Recall higher than Precision and this makes it more suitable in certain applications where it's more important to classify correctly as many positive samples as possible, rather than maximizing the number of correct classifications. This is why $F2$ scores are higher than $F1$ ones.

The False Positive Rate represents the probability that false alerts will be raised; it is very low in general and it is higher for the models with more false positives, such as SVM.

The Matthews Correlation Coefficient (MCC) is a reliable statistical rate that takes into account all four confusion matrix categories. It has been shown that MCC is more informative and truthful than $F1 - Score$ and Accuracy in binary classification problems [77]. Since our datasets are very imbalanced, we use a weighted MCC. Its value is between 78.12% and 98.55%.

The Area Under the Curve of Receiver Operator Characteristic explains how much the model is capable to distinguish between classes. It is very high, at about 99%, for all the detectors.

The Area Under the Curve of Precision-Recall Curve combines Precision and Recall in a single visualization. On average, it is about 59.63%.

Now we provide a visual overview of the performance of the detector which performs best,

Active Learning.

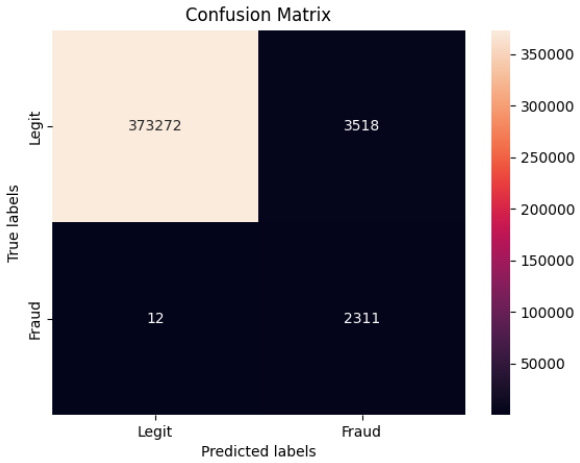

Figure 4.2: Confusion Matrix, Active Learning, Weekly Update

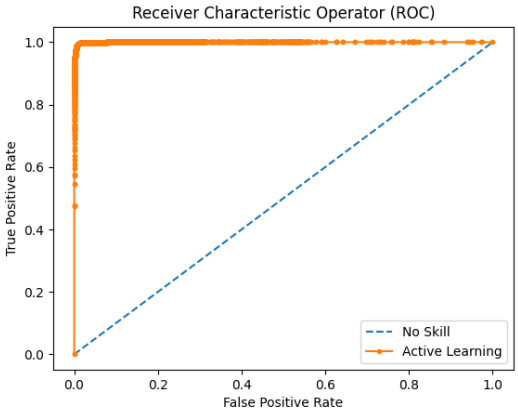

Figure 4.3: AUC-ROC, Active Learning, Weekly Update

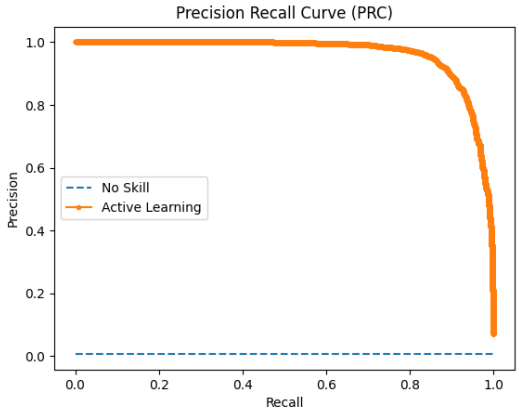

Figure 4.4: AUC-PRC, Active Learning, Weekly Update

In Figure 4.2, we can observe the confusion matrix, related to the weekly update. We test 379,113 transactions, 376,790 are legitimate and 2,323 fraudulent. The Active Learning detector is violated only by 12 frauds, while 2,311 are correctly detected. On the other hand, it raises 3,518 false alarms, which is acceptable in a banking context.

Figure 4.3 and Figure 4.4 shows the Receiver Operator Characteristic curve and the Precision- Recall curve. ROC allows us to understand the relation between TPR and FPR at different classification thresholds: the closer the graph is to the top and left-hand borders, the more accurate is the model. On the other hand, the closer the graph to the

diagonal, the less accurate is the model. A perfect curve would go straight from zero up to the top-left corner and then straight across the horizontal. As you may notice, our AUC is very similar to a perfect curve. PRC specifies the precision against the recall at various thresholds: the perfect curve will have a PRC that passes through the upper right corner (corresponding to 100% precision and 100% recall). In our case, also this curve is very similar to the best case possible, meaning that our detector is effective.

5 | Poisoning Attack

This chapter aims to deeply explain which is the attack setting, which is the attacker's approach and how the adversary carries on his or her strategy to mount poisoning attacks. We explore attacks against every detector, for each scenario, and for each update policy.

5.1. Assumptions

First of all, it's necessary to contextualize the setting and the main assumptions on which our poisoning attacks are based. As we already said, there are three possible scenarios: White Box, Grey Box, and Black Box. Every scenario specifies the attacker's degree of knowledge about the target systems. Moreover, they are independent of each other and they are studied separately. The main assumptions on which our work is based are:

- The attacker has at his or her disposal an older dataset of banking transactions that can exploit in order to train the oracle;
- The attacker can retrieve legit past victim's transactions executed in the month before the beginning of the attack;
- The target Fraud Detection Systems have an update policy (i.e., weekly or biweekly), according to which they update their training set;
- The attacker can perform transactions on behalf of the victim, according to the transaction hijacking pattern, or by his or her devices, after stealing the victim's credentials;
- The attacker can manipulate all attributes which characterize a single transaction, except the IBAN_CC;
- An attack can last at most 8 weeks, when the most recent dataset finishes. Moreover, if a transaction is labeled as fraudulent by the target machine, the attack against that victim ends immediately;
- The attacker can adopt different strategies depending on his or her goal and the

current scenario;

- The attacks are executed on the more recent dataset;
- The victims are chosen according to specific criteria which allow us to study poisoning attacks.

5.2. Attack Approach Overview

In this section, we want to clarify the approach that the attacker uses to execute poisoning attacks. The attack is divided into different phases, each of which has its characteristics and needs to be adequately explored.

The first step is understanding the scenario with which the adversary has to deal. The fraudster has to figure out what he or she knows about the target system and which instruments have at his or her disposal. In other words, he or she needs to understand which is the scenario among White Box, Grey Box, and Black Box. Then, the adversary selects the victims to attack. In more detail, he or she targets customers affected by previous scams, such that phishing campaigns or Trojan malware, which allow the attacker to steal the necessary information to carry out the attack. In this work, the fraudster selects 15 victims, an empirical number that allows attacking customers with different spending patterns and guarantees an acceptable computational effort. The next step consists of retrieving the past transactions executed by the chosen victims, with the goal to collect all the information necessary to build users spending profiles and, consequently, to craft evasive frauds which partially replicate victims' behaviors. At this point, the attacker creates fraudulent transactions trying to mimic the spending habits of the victim. More specifically, he or she selects through specific algorithms the features which can be controlled, based on the information previously retrieved. After crafting frauds, the adversary builds and trains the Oracle, i.e., the model which takes care of validating and regenerating the malicious transactions. It strictly depends on the scenario and the attacker's knowledge. Once the fraudster has aggregated and standardized the transactions and has extracted the features in the best possible way, he or she submits the crafted frauds to the Oracle. If the Oracle classifies them as legitimate, they are subjected to the target system; otherwise, they are regenerated (or deleted, in the worst case) and submitted again, until they overcome the Oracle checks. If the proposed transactions are considered legit also by the target detector, another attack, after some days, depending on the update policy, will be performed. At this point, the bank machine learning model is now trained on data that contain the transactions crafted by the attacker. Consequently, the FDS training set is corrupted and poisoned. This is why the next attacks against the same victim will consist

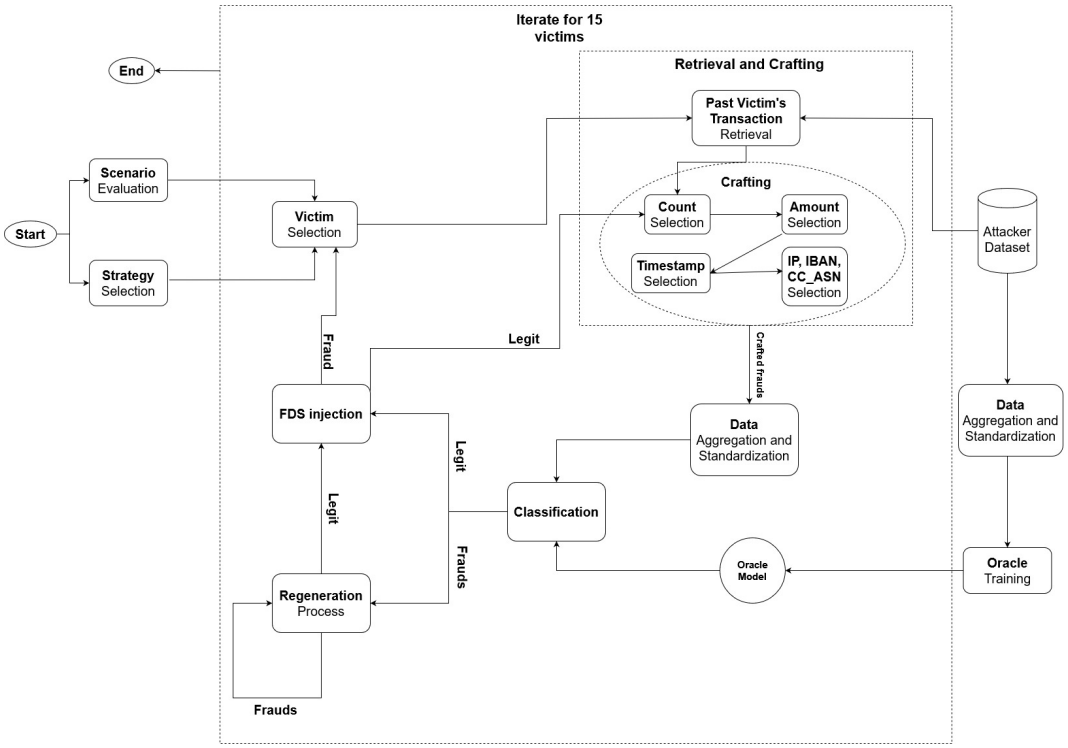

Figure 5.1: Attack Flow Graph

of more transactions with a higher amount, according to the attacker's goal and strategy. On the other hand, if the target system detects at least one fraud among those subjected, the attack against that victim ends and the adversary will affect another customer. The attack against one user lasts as long as the dataset ends (i.e., 8 weeks after the start of the attack) or when a fraud is detected. We repeat this process for 15 victims selected.

Figure 5.1 illustrates the flow graph from the attacker's point of view. Firstly, the attacker tries to capture the scenario where he or she is, and how he or she knows about the target system. Moreover, the adversary chooses a strategy, based on his or her goal. Then, he or she starts the attack against the 15 victims selected, one at a time. He or she retrieves the user's past transactions history, in order to craft the malicious transactions which are submitted to the Oracle, after having trained it according to the scenario. If the Oracle rejects some transactions, they are regenerated, modifying specific features, until they are accepted. The transactions deemed legitimate are subjected to the target FDS; if they are accepted, they are injected into the system training set and new frauds for the next week, or two weeks, depending on the update policy, are crafted, otherwise the attack against the victim stops and the adversary carries on with another victim. The process ends when the attacker has performed poisoning attacks against all 15 victims.

5.3. Scenario and Strategy

5.3.1. Scenario

When executing an attack, the fraudster can be in three different situations, that depend on the knowledge about the target Fraud Detection System.

In the White Box scenario, the attacker knows everything. He or she is able to replicate a perfect Oracle which is identical to the bank machine learning model. In particular, the adversary has the target training set, the system update policy, the algorithm, and its hyper-parameters. Moreover, he or she knows all the features used, both direct and aggregated, and has a perfect knowledge of the victim's past transactions. This scenario represents the best possible case: although it is an unrealistic situation, the attacker might be internal to the bank, in a way that he or she knows every aspect to perform an attack.

In the Grey Box scenario, the adversary only knows the features set and the update policy which are used in order to train the target model. In this way, the attacker cannot build an Oracle which is perfectly equivalent to the FDS: in fact, he or she doesn't know the algorithm, the hyper-parameters, and the training set. This is why he or she makes use of a surrogate dataset and a particular strategy to create the Oracle. This is an essential point: the Oracle must be reliable, it has to validate and regenerate the crafted frauds correctly, otherwise the FDS will detect them. As we already said, the attacker doesn't know the algorithm, so he or she has to rely on a particular approach based on ensembling methods.

In the Black Box scenario, the attacker doesn't even know the feature set: he or she tries to aggregate them in the best possible way, by selecting just 50 features according to a filter approach. This is a consistent improvement with respect to [27], in which the Oracle was built using 70 features. Moreover, as in Grey Box, the adversary has the victim's transactions executed in the month before the beginning of the attack, which will be used to create a reliable user spending profile.

5.3.2. Strategy

Another relevant aspect when performing an attack is to select a strategy that suits the adversary's goals. More specifically, we present three different directions: poisoning amount, poisoning count, and poisoning both. In the first one, the attacker tries to steal money in the smallest possible time window, without worrying about being detected. He or she focuses on poisoning the transactions' amount, increasing it consistently every iteration.

On the other hand, the number of transactions per week (or two weeks, depending on the update policy) is kept almost coherent with respect to that before the attack. Poisoning count is the opposite approach: the adversary tries to poison the count of transactions per week, crafting frauds that have an amount similar to the mean of legit transactions executed by the victim. According to Monti [27], increasing the count is more cautious than focusing on the amount and with this strategy, the adversary minimizes the attack detection. The last approach is a hybrid one that combines the first two: in particular, the attacker's goal is to steal as much money as possible, poisoning both count and amount, without caring about the detection. Every strategy presents a conservative and an aggressive version.

Strategy	Increase Count (%)	Increase Amount (%) Percentage Deletion (%)	Min-Max Increase (€) Confidence, Nat/For (%)	Max Std (%)
Cons. Amount	0	0.8 0.5	25-5000 0.900/0.675	0.75
Cons. Count	0.3	0.2 1	25-2500 0.950/0.750	0.75
Cons. Both	0.3	0.75 0.6	25-2500 0.925/0.850	0.75
Greedy Amount	0	1.2 0.4	25-5000 0.900/0.675	0.75
Greedy. Count	0.7	0.4 1	25-2500 0.950/0.750	0.75
Greedy. Both	0.7	0.9 0.6	25-5000 0.925/0.850	0.75

Table 5.1: Strategy Parameters

Table 5.1 specifies the values assigned to each parameter which composes the strategy:

- Increase Count: increment percentage of the transactions number with respect to the previous iteration one;
- Increase Amount: increment percentage of the transactions' amount mean with respect to the previous iteration one;
- Min-Max Increase: minimum and maximum possible amount increase between one iteration and the other;
- Max Std: percentage standard deviation which limits the amount increase;
- Percentage Deletion: parameters that state when a transaction has to be deleted (i.e., we delete transactions with an amount equal to "Percentage Deletion" * k, where k is the minimum amount selected in the previous iteration);

- Confidence: it specifies how we trust the Oracle (i.e., we accept frauds that are considered legitimate by the Oracle with a percentage higher than "Confidence"). We provide two possibilities for this parameter, one refers to national frauds and the other to foreign ones. The reason is that our Oracle is very strict against foreign frauds: hence, we cannot use high confidence, otherwise, our Oracle will reject each of them. Moreover, while poisoning the count, we are much more cautious than when poisoning the amount, because we want to minimize the attack detection.

5.4. Victim Selection

As we explained in section 3.2.1, we have divided the users into nine categories, with the aim to deal with all the possible spending patterns present in our dataset. Users with few transactions with small amount are more than users with many costly transactions. Moreover, customers with a little number of banking operations are more difficult to poison, because their incremental spending pattern change will be more consistent and then more suspicious. As Carminati et al [28] did, we consider victims which have at least four transactions in the month before the attack. In this way, we are able to collect 8710 suitable victims, which can be chosen by the adversary. In addition, we want to separate national customers from foreign ones, because international transactions are harder to poison. We consider 'foreign' users who, in the time window analyzed, have sent money more often to foreign countries (i.e., transactions with IBAN_CC different from 'IT'), while we define 'national' all the others. During a single attack, the fraudster selects 15 victims, of which 4 are foreign, trying to replicate the spending categories and the international customers' distribution of our dataset. Table 5.2 summarizes the information about the 15 victims selected.

Category	National	Foreign
1	3	1
2	1	1
3	1	1
4	1	1
5	1	0
6	1	0
7	1	0
8	1	0
9	1	0

Table 5.2: Victims Selection

5.5. Retrieval and Crafting

5.5.1. Retrieval

One important step during the attack is the user's past transactions retrieval. The adversary collects information about the operations history performed by the victim: then, he or she studies them. The goal of this step is trying to replicate the spending behavior of the victim, in order to deceive the FDS. Hence, the attacker can compute some important statistics, such as the amount mean, the average count per week, and more recent and more frequent IP addresses or CC_ASN. Finally, once he or she has explored each feature, the adversary can create transactions that could be performed by the victim itself. Moreover, in the White Box scenario, the attacker has all the previous transactions belonging to the victim, while in Grey Box and Black Box has partial knowledge (i.e., one month's transactions history). In the former, he or she is able to compute a perfect victim spending profile; in the latter case, the adversary tries to approximate it through the available information.

5.5.2. Crafting

At this point, the attacker has all the necessary instruments to craft misleading transactions. In this work, we assume that the attacker can directly control almost every feature to create raw transactions. In [27], the author considered as controllable features only the amount, the timestamp, and the count (i.e., the number of transactions per iteration). However, it is a very restricted set of features. We can assume that the attacker can also

manipulate the IP address, the CC_ASN identifier, the IBAN, and the confirmation SMS. This is a realistic assumption because the IP and the CC_ASN could be easily spoofed and replicated. In addition, we assume, as already said, that the adversary can execute transactions on behalf of the victim. On the other hand, the IBAN addresses, which is always different from the ones used by the victim, can be used multiple times and at will by the attacker. Also, the confirmation SMS is controlled by the attacker, who can arbitrarily choose if a specific transaction is associated with an OTP (i.e., One Time Password). Finally, we assume that the Country Code related to the IBAN is partially manipulated by the adversary. This could be a counter-intuitive assumption, but the reason is that foreign transactions have a higher level of suspicion with respect to national ones. All the fraud detection systems behave in two very different ways when facing transactions toward Italy and other European Countries; this is why it's really interesting to analyze the differences and compare the results.

5.5.3. Timestamp Selection

The timestamp associated with one transaction is a core feature in the fraud detection domain and it specifies precisely when the operation is executed. The attacker has to properly and carefully manage this characteristic because it is crucial for the FDS final output and it influences most of the aggregated features. In order to craft deceptive transactions, the adversary studies the spending profile previously created and try to mimic as best he or she can the victim's behavior. To do that, he or she looks at the most frequent weekdays and hours. For instance, if a victim usually performs many transactions during the night hours on the weekend, the attacker will try to replicate this pattern.

We design a novel algorithm that selects the most frequent weekday on which the user usually performs transactions. Then, the adversary selects that weekday as many times as the maximum number of operations executed in one day by the victim. Finally, he or she repeats the process with the second most frequent weekday, until the total number of frauds specified by the strategy is crafted. Regarding the hours' selection, the same approach is applied. We show the pseudo-code of the algorithm in Algorithm 5.1.

5.5.4. Amount Selection

The amount is the most important feature in this context: everything revolves around the transactions' amount since adversaries will try to steal as much money as possible. Wire transfers with higher amounts will attract more attention from the target models, which tend to consider consistent transactions as outliers in users' spending norm. On

Algorithm 5.1 Timestamp Selection Algorithm

Input: df_user: Dataframe containing past user's transactions, ts: Starting timestamp, n: Number of transactions to perform

Output: Attack timestamps list

SelectTimestamp (df_user, ts, n):

$timestamps \leftarrow listUserTimestamps(df\_user)$ Get past user's timestamps
$moreFrequentWeekdays \leftarrow getMoreFrequentWeekdays(timestamps)$

$moreFrequentHours \leftarrow getMoreFrequentHours(timestamps)$

$maxInDay \leftarrow getMaxInDay(timestamps)$

$maxInHour \leftarrow getMaxInHour(timestamps)$

$days \leftarrow []$
$i \leftarrow 0$
$j \leftarrow 0$

while $n > 0$ **do**
  while $i < 7$ **do**
    $weekday \leftarrow moreFrequentWeekdays[i]$
    while $j < maxInDay$ **do**
      $day \leftarrow ts + weekday$
      $days.append(day)$
      $j \leftarrow j + 1$
      $n \leftarrow n - 1$
    end while
    $i \leftarrow i + 1$
  end while
end while

$daysWithHours \leftarrow selectHours(days, moreFrequentHours, maxInHour)$
$sortedList \leftarrow sorted(daysWithHours)$

return *sortedList*

the other hand, the amount feature is targeted by the attacker, who wants to increase its value at each iteration, in order to carry on the poisoning task. Hence, on one hand, the adversary wants to craft frauds with a significant amount, on the other he or she has to be cautious in order to deceive the FDS.

We present an algorithm that consists of selecting an amount that depends on statistics concerning past user transactions and the attacker strategy. The key idea is to increase the amount value by adding to the user mean a certain parameter which depends on the standard deviation of the victim's transactions history and the increment percentage stated by the strategy. In this case, we intend a weighted mean and standard devia-

tion: in fact, we want that the attacker is cautious and give more importance to more recent transactions. Since we want to focus on poisoning attacks, which involve iterative approaches, the increment amount chosen will be never less than that of the previous iteration. Finally, we use the Gaussian function which generates a floating number according to a Normal distribution, in order to have some unpredictability. We present the pseudo-code of the algorithm in Algorithm 5.2

Algorithm 5.2 Amount Selection Algorithm

Input: df_user: Dataframe containing past user's transactions, strategy: Dictionary containing information strategy, ts: Actual Timestamp, previousMax: Max amount selected of previous iteration

Output: Candidate Amount

SelectAmount (df_user, strategy, ts, previousMax):

$minIncrease \leftarrow strategy['min\_increase']$
$maxIncrease \leftarrow strategy['max\_increase']$
$stdMax \leftarrow strategy['std\_max']$
$incrementPercentage \leftarrow strategy['increment\_percentage']$

$stastics \leftarrow computeStatistics(dfUser, ts)$
$mean \leftarrow stastics['mean']$
$std \leftarrow statistics['std']$

if $mean < previousMax$ **then**
  $mean \leftarrow previousMax$
end if
$increase \leftarrow incrementPercentage * mean$

if $increase < minIncrease$ **then**
  $increase \leftarrow minIncrease$
else
  if $increase > stdMax * std$ **then**
    $increase \leftarrow stdMax * std$
  end if
end if
if $increase > maxIncrease$ **then**
  $increase \leftarrow maxIncrease$
end if
$\mu \leftarrow mean + increase$
$\sigma \leftarrow 0.05 * \mu$
$finalAmount \leftarrow random.gauss(\mu, \sigma)$
return $round(finalAmount, 2)$

5.5.5. Count Selection

The count represents the number of transactions executed during an iteration. Together with the amount, it is another feature that the attacker may poison. However, the count affects only a restricted set of aggregated features. Poisoning this attribute is less risky with respect to the amount. From the attacker's point of view, increasing the number of victim's transactions is more stealthy. This is why we decided to focus on a particular attack dedicated to poisoning exclusively the count. More specifically, the direction of this strategy aims at increasing the number of transactions more aggressively with respect to the one which focuses on the amount. If we want to poison the amount, we always add just one to the legit count of the victim, otherwise, we increment it according to our strategy. The legit count of the victim is extracted by computing the average transactions' number considering the available time window before the attack.

The algorithm we propose is pretty linear, and it depends on the strategy selected and on the count of the previous iteration. We show the pseudo-code of the algorithm in Algorithm 5.3.

Algorithm 5.3 Count Selection Algorithm

Input: df_user: Dataframe containing past user's transactions, strategy: Dictionary containing information strategy, ts: Actual Timestamp, previousCount: Count used in the previous iteration

Output: Selected Count

SelectCount (df_user, strategy, ts, previousCount):

$increaseCount \leftarrow strategy['increase\_count']$
if $(strategy['objective'] ==' poison\_amount')$ **then**
  $prevCount \leftarrow selectFirstCount(df\_user, ts) + 1$
else
  $newCount \leftarrow int(prevCount * (1 + increaseCount))$
end if
if $newCount == prevCount$ **then**
  $newCount \leftarrow newCount + 1$
end if
return $newCount$

5.5.6. Other Features Selection

The attacker can control also the IP, the CC_ASN, the SMS confirmation and the IBAN. The IP and the CC_ASN are crafted by looking at the most recent and the most frequent used by the victim, exploiting an approach similar to that applied for the Timestamp. In addition, the IP and the CC_ASN can be changed at "runtime", thanks to the help of the

Oracle. I submit a specific fraud with a certain IP and CC_ASN and, for each feature, the Oracle told me if I should change or keep it (i.e., if the probability of classifying it as legitimate increases). Regarding the SMS confirmation, the attacker simply replicates the distribution occurred in the victim's past transactions. Finally, the IBAN and the IBAN_CC are tightly coupled. Obviously, the adversary uses specific personal IBAN. However, we assume that he or she cannot completely control the IBAN_CC, because for the purpose of our work, it's important to study separately foreign and national transactions. Hence, the IBAN_CC are selected according to the previous ones used by the victim: if a user is classified as foreign (i.e. he or she has more international transactions than national ones), the adversary selects European Country Codes (i.e., different from 'IT') which are equal to those used previously by the victim. For each different IBAN_CC, the attacker has a specific IBAN, that can exploit multiple times at will. Moreover, as the IP and the CC_ASN, he or she can regenerate it at runtime, if the Oracle suggests it.

We report the algorithm which selects the initial IP addresses, really similar to the one that deals with CC_ASN. First, it extracts the most recent IP addresses; then, it adds to the final list each of them according to its frequency. We provide the pseudo-code in Algorithm 5.4.

Algorithm 5.4 IPs Selection Algorithm

Input: df_user: Dataframe containing past user's transactions, n: Number of transactions to perform
Output: Selected IPs
SelectIPs (df_user, n):

```
moreFrequentIPs ← getMoreFrequentIPs(df_user)
moreRecentIPs ← getMoreRecentIPs(df_user)
IP_list ← []
k ← 0
i ← 0
while i > len(moreRecent) do
    frequence ← getFrequence(moreRecentIPs[k], moreFrequentIPs)
    while frequence > 0 do
        IP_list.append(moreRecentIPs[k]
        frequence ← frequence − 1
    end while
    k ← k + 1
end while
IP_list ← IP_list[: n]
return IP_list
```

5.6. Oracle

The Oracle is the machine learning model which is built by the attacker to have a reliable imitation of the target Fraud Detection System. It's a key point in Adversarial Machine Learning applied in the fraud detection domain because a poor Oracle leads to bad results from the adversary's point of view. The goal of the fraudster is to create a model, with the available instruments, which is close to the bank FDS. However, this is very hard in practice. In fact, the attacker may create the Oracle with different algorithms, trained on a different dataset, with different features. In [27] and [28], the authors propose to overcome this problem by using the best algorithm found, respectively XGBoost and Random Forest. However, trying to create a replica of an unknown model with just another model cannot really solve the problem. Instead, the scope of this work is to propose and show an alternative method, based on ensembling learning, which allows to create a very strong Oracle, that is reliable and closer to the target machine. After having explored and compared different ensembling solutions, we can conclude that the most powerful Oracle found is based on the Light Gradient Boosting algorithm, improved by Bagging with 20 bootstraps.

5.7. Regeneration Process

Features' regeneration is a key task during the poisoning process. The attacker, in each scenario, builds an Oracle which validates the fraudulent transactions. Based on the outcome of the Oracle, he or she either submits them to the target FDS or regenerates them by changing the value of a subset of features: IP address, IBAN, CC_ASN, and amount. As we stated in section 5.5, the attacker crafts frauds based on attributes related to the victim's previous transactions: hence, he or she uses IP address, IBAN and CC_ASN already adopted by the victim or by prior attacks. With regeneration, the fraudster substitutes these values with random ones. The adversary crafts the malicious transactions based on the information at your disposal, then filters each of them through the Oracle: if the Oracle rejects a transaction, the fraudster changes the features until the Oracle accepts it. First, he or she regenerates the IP, followed by the IBAN, CC_ASN, and finally the amount: the attacker wants to steal as much money as possible and this is why he or she first changes the categorical features and only at the end, if needed, he or she cares about the amount. In this work, we deeply study which features the adversary needs to change more frequently. The results are really interesting, especially in the White Box scenario, where the attacker has a perfect replica of the target machine. The regeneration process strictly depends on the features on which the models are trained. Concerning the

Grey and Black Box scenarios, we analyze which features our Oracle pushes the attacker to modify. In other words, in this task, the attacker's goal is to end up with an evasive transaction with the highest possible amount.

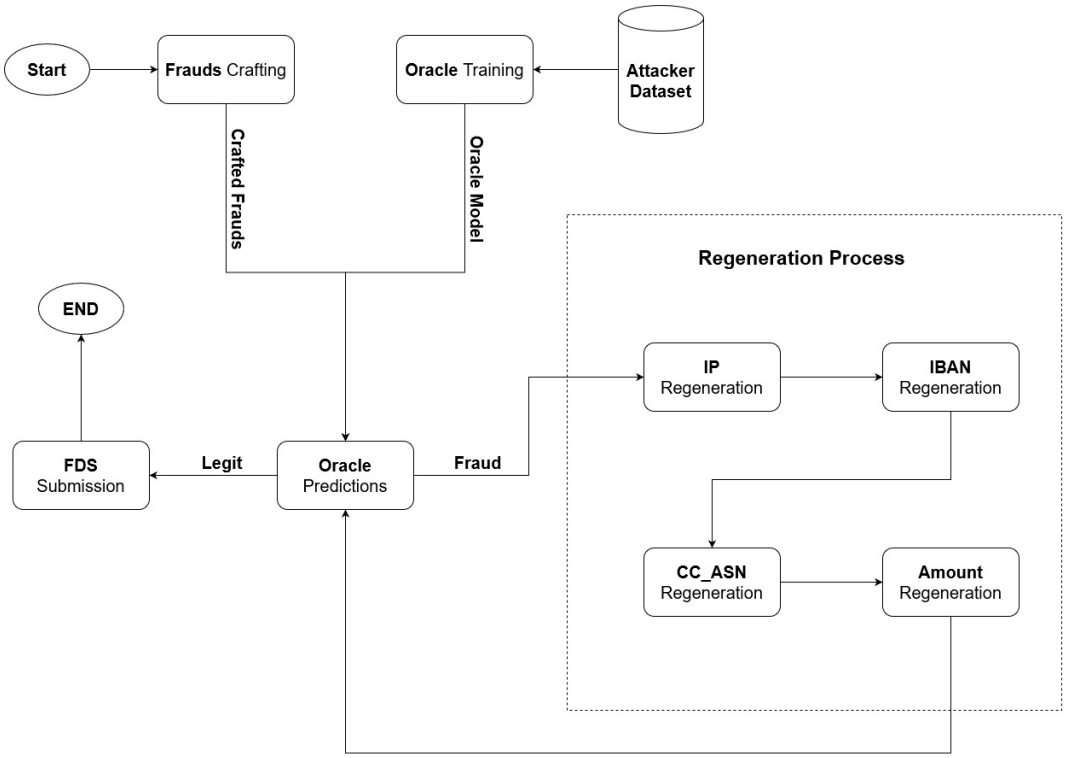

Figure 5.2: Regeneration Flow

In Figure 5.2, we provide a visual explanation of the regeneration process: the attacker submits the transaction to the Oracle, and if it's rejected, he or she modifies the IP address. Then, it submits again the regenerated transaction, and if it's refused by the Oracle, he or she changes the IBAN and validates the new transaction. The adversary performs the same task for the CC_ASN. At this point, if the Oracle rejects the transaction with regenerated IP, IBAN, and CC_ASN, the adversary needs to lower the amount, until the operation is accepted.

6 | Implementation Details

This chapter aims to provide a full explanation of the design of our system. We discuss its static view and its dynamic run-time view. Moreover, we present all the software tools, libraries, programming languages, and hardware settings that we exploited in order to conduct our experiments.

6.1. System Architecture

We explain our software architecture through a system UML diagram, presented in Figure 6.1, which gives an idea of the static view concerning the implementation of our work. Our approach is composed of seven modules, which interact with each other. The attack module is used to start the poisoning attack. In order to do that, it exploits the Victim block, which is in charge to select the victims. The Oracle module has the task to create and fit the model, which is different according to the scenario. The attack utils block is composed of auxiliary functions: for instance, it is used to aggregate and standardize the data. Moreover, it is exploited for crafting the frauds and this is why it uses the Feature module, which is in charge to select the features to compose each fraud. Finally, the attack evaluation block has to evaluate the frauds, compute the metrics introduced in Section 7.1, and save the results.

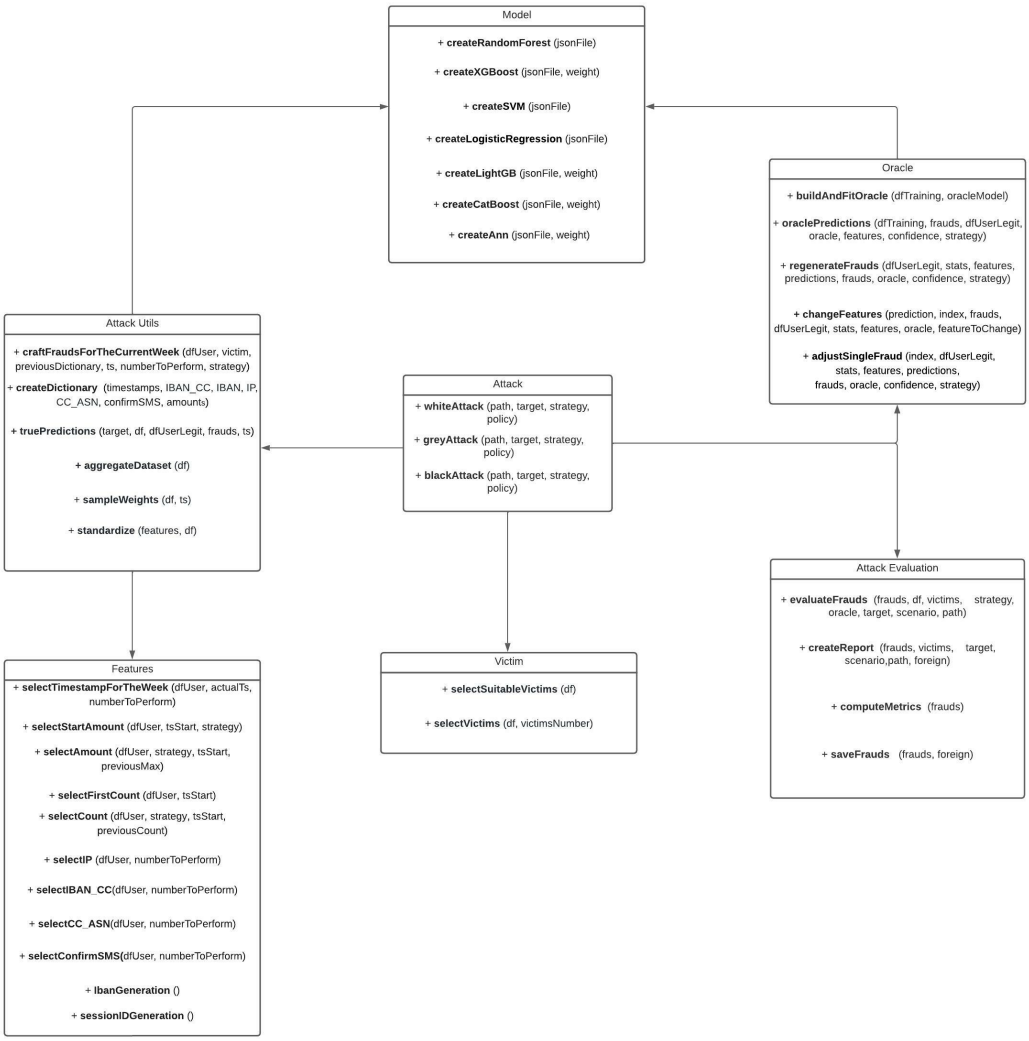

Figure 6.1: System UML Diagram

6.2. Run-Time Architecture

The goal of this section is to provide a dynamic run-time view of our work, in order to have a visual explanation of the attack process. In Figure 6.2, we present a UML sequence diagram that aims to give an idea of the functions and the components involved in a simulation.

The process starts by selecting a scenario, a target FDS, a strategy, and an update policy. Then, we select 15 victims from a pool that contains users with at least four transactions performed one month before the attack. After that, the adversary trains the Oracle using the old dataset, which will be used in order to filter the fraudulent transactions. At this point, for each victim selected, the adversary first retrieves the past user's transactions

and then, thanks to this information, decides an initial count and amount, which depends on the strategy. Subsequently, the attacker starts the poisoning process, which begins on 22/12/2014 and ends, in the best case, on 22/02/2015. Now, the adversary crafts the frauds, by relying on the spending pattern of the victim and on the strategy, and then he or she validates and regenerates them through the use of the Oracle previously trained. The Oracle, after the filtering, returns the new fraudulent transactions. These transactions are submitted to the target FDS, thanks to the truePredictions function, which is in charge to aggregate and standardize the transactions. Then, the FDS predicts the incoming samples and updates the banking dataset. If a fraud is detected, the poisoning process against the current victim is stopped and the adversary starts the same attack against another user. Otherwise, he or she increments the timestamp and creates a dictionary, which will be used in the next iteration in order to keep track of the previous transactions successfully injected. Once all 15 victims are defrauded, the frauds are saved and evaluated.

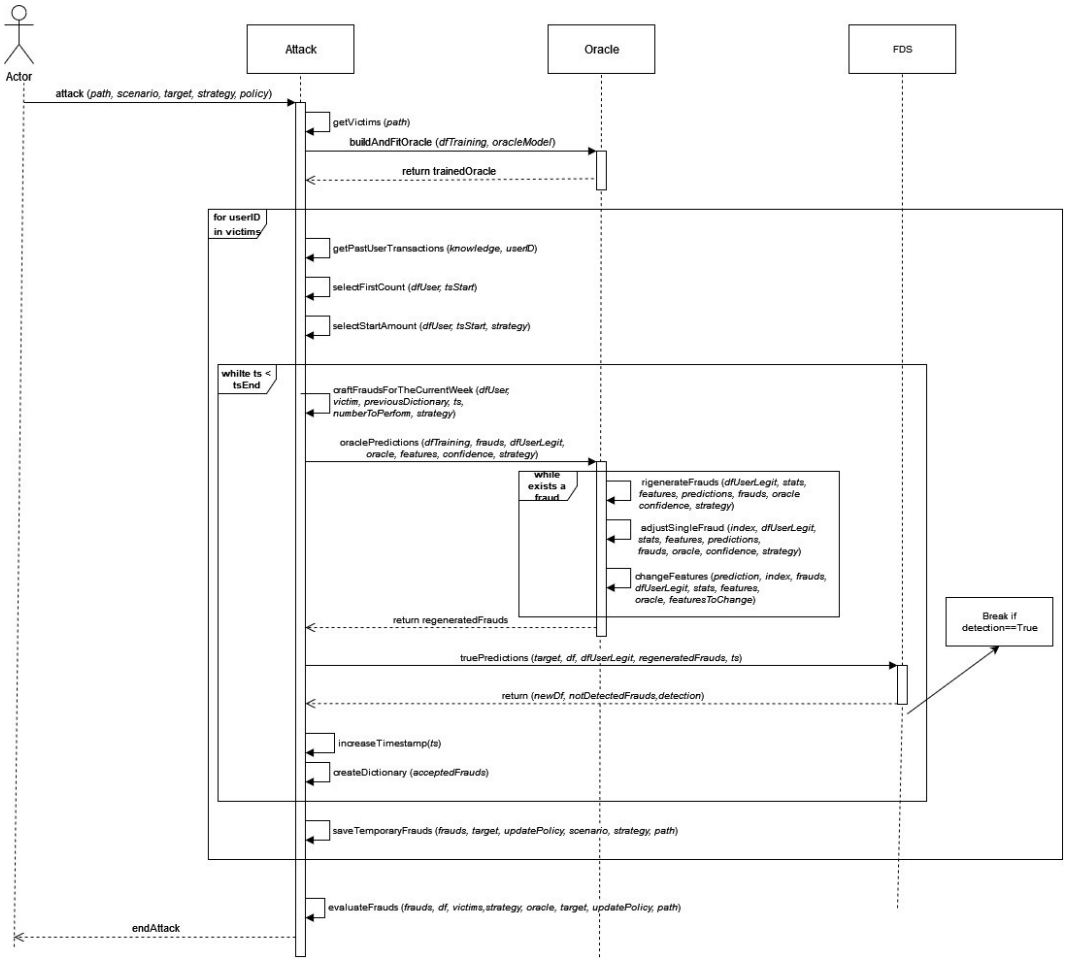

Figure 6.2: UML Sequence Diagram

6.3. Hardware and Software Architecture

In this section, we discuss the tools, libraries, and hardware settings that we have exploited in order to perform our experiments and achieve our results.

The main programming language used is Python 3.8.12, and we used PyCharm 2021.2.3 as IDE for the development. We store our data and our results into Comma Separated Values (CSV) files, while we use JSON files in order to save information such as features and hyperparameters.

We list the most important libraries exploited during our work:

- numpy 1.23.0 is a library that allows performing complex mathematical operations on very large data structures.
- pandas 1.3.4 is a fast and powerful data analysis and manipulation tool, that is useful to manage our large datasets composed of transactions.
- scikit-learn 1.0.1 is a software machine learning library for the Python programming language which features various classification algorithms including support vector machine, random forest, logistic regression, and isolation forest. Moreover, it gives the opportunity to implement ensembling techniques such as Bagging and Boosting.
- catboost 0.24.1 and lightgbm 3.3.2 allows us to implement respectively CatBoost and LightGB model.
- xgboost 1.6.1 is a library that is used in order to create and fit XGBoost models.
- tensorflow 2.9.0 and keras 2.9.0 are exploited to implement Artificial Neural Networks.
- matplotlib 3.5.0 is a software library that gives the possibility to plot data in order to have a graphical overview of our results.

In order to conduct our experiments, we used two different machines.

In particular, the frauds generation process, the aggregation, the feature selection, the hyperparameter tuning, and the training tasks were executed on a laptop with the following specifications:

- OS: Windows 10 Home
- Model: ASUS Zenbook 14 UX431F
- Processor: Intel® Core™ i7-8565U Processor 1.8 GHz

- Graphics: Intel® UHD Graphics 620, NVIDIA® GeForce® MX150, 2 GB GDDR5
- Memory: 16 GB LPDDR3
- Storage: 256 GB

On the other hand, the simulations concerning the poisoning attacks are performed on remote machines, which have the following specifications:

- OS: Ubuntu LTS 16.04
- Processor: Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz, 4 Cores, x86_64 architecture.
- Memory: 16 GB
- Storage: 250 GB

7 | Experimental Evaulation

In this chapter, we explain the metrics that we adopt in order to evaluate poisoning attacks. Then, we provide the experiments that allow us to select a reliable Oracle. We show our poisoning attacks' results, for each scenario, for each strategy, and update policy. Finally, we comment on the results and provide an overview of the regeneration process task.

7.1. Metrics

To evaluate and analyze the poisoning processes against the Fraud Detection Systems, we need to rely on specific metrics, which allow understanding the effectiveness of the attacks from different points of view. Before introducing them, we explain the meaning of the terms that we will use: V is the set of the victims, R is the set of regenerated frauds, L represents the frauds considered legitimate by the Oracle, F all the fraudulent transactions proposed by the attacker, A the frauds accepted by the target machine, D the detected ones, T_d is the difference between the attack start time and the detection time of the transaction, $S_{w,v}$ is the amount of money stolen from victim v in week w. We present five metrics:

- Injection Rate:

$$IR = \frac{|L|}{|F|}$$

  The injection rate (**IR**) is a metric that specifies how much an attacker needs to regenerate the submitted transactions. A high injection rate means that the adversary has to modify at least one feature among the IP address, IBAN, CC_ASN, and amount. On the contrary, we have a low injection rate when the Oracle considers most of our crafted frauds as legitimate.

- Evasion Rate:

$$ER = \frac{|A|}{|F|}$$

  The evasion rate (**ER**) represents the fraction between the number of frauds eval-

uated legitimate by the target machine and the number of transactions submitted to the FDS. It specifies the percentage of the crafted frauds which are accepted by the banking system. An attack with ER = 100% means that every fraud has been successfully submitted without being detected.

- Detection Rate:

$$DR = \frac{|D|}{|V|}$$

The detection rate (**DR**) plays a key role in this work: it represents the percentage of how many attacks against the total number of victims have been detected. An adversary mounts poisoning attacks affecting 15 victims: if the attacks against 4 victims are detected the DR is equal to $(4/15) * 100 = 26\%$

- Average Detection Time:

$$ADT = \frac{\sum_D T_d}{|D|}$$

The average detection time (**ADT**) shows after how many days, on average, an attack is detected: an attack that hasn't been noticed by the target machine is not considered for the computation of the metric. An ADT equal to 40 means that, on average, a FDS takes 40 days in order to detect a poisoning attack.

- Money Stolen:

$$MS = \sum_i amount(A_i)$$

The Money Stolen metric(**MS**) indicates the amount of money that the adversary is able to steal against a specific target system by considering all 15 victims.

- Average Weekly Increase:

$$AWI = \frac{1}{n\_weeks} * \sum_{w=0}^{n\_weeks-1} \frac{S_{w+1,v}}{S_{w,v}}$$

The Average Weekly Increase (**AWI**) allows to understand how fast is the poisoning process. It is the average increase of one iteration with respect to the previous one. An AWI equal to 150% means that, on average, the attacker is able to steal 150% more from the previous iteration.

7.2. Oracle Results

In Section 5.6 we explained how to improve the approach to the Oracle. We exploit ensembling learning techniques in order to create an Oracle as reliable as possible. In this

section, we explore different ensembling solutions, compare them, and we select the best one.

Ensembling methods exploit multiple learning algorithms and combine their predictions in order to classify new incoming examples [66]. The objective is trying to extract every single advantage of the considered models. This is why, theoretically, ensembling tends to perform best when there is a consistent difference among them [67]. In general, ensembling methods always outperform single models, but they require a more demanding computational effort. There are different approaches when dealing with ensembling, each of them has its peculiarity and is useful in a specific context. The most popular are bagging, boosting, stacking, and majority voting.

Bagging, or bootstrap aggregating, consists of dividing the entire dataset into n bootstrapped datasets. For every smaller dataset, you train one model, building a single classifier for each bootstrap. Finally, you average all the n models, obtaining a more reliable one. This process has different benefits: first of all, from a theoretical point of view, if we assume that datasets are independent, we get a final model which has a variance equal to $\frac{1}{n}$ of the total one. Hence, bagging helps to reduce the overfitting risk, by dealing with models with high variance. Moreover, bagging is really useful when working with unstable learners and when the data present a lot of noise.

Boosting, instead, aims at reducing the bias. Its goal is to transform a set of weak learners in a strong one [68]. It consists of assigning weights on data examples, starting from random uniform weights, and then you iteratively re-weight the data according to the classification error (miss-classified cases get higher weights). By doing this you focus on difficult observations because they will get higher importance. The final ensemble model is then a weighted combination of all the individual models. Although it's extremely easy to implement, this method tends to overfit, because you focus on outliers that are really specific to your dataset, and by assigning them more weight, you risk considering also the noise of the data.

Stacking, or stacked generalization, is a powerful ensembling method that has important differences with respect to both Bagging and Boosting. Although the goal is always the same (i.e., increasing the performance by involving the predictions of multiple machine learning models), its architecture is composed of two layers: a set of base or level-0 models, and a meta-model, or level-1 model. The base models are trained on the training set and their predictions are analyzed by the meta-model, which combines the output of the level-0 models. In this way, the meta-model is trained on the predictions of the base models, which are usually different and fitted on the same dataset. Moreover, the training set

used for the meta-model is prepared through k-fold cross-validation for the level-0 models [74].

Majority voting is the most simple and spread ensembling method. It consists of considering the predictions of the involved models and output the class with the most votes. However, in classification, there are two possibilities: hard and soft voting. Hard voting means that you predict the class with the largest sum of votes, while with soft voting you predict the class with the largest summed probability from models [76]. However, the majority voting approach has several drawbacks. There are situations where an individual model can outperform a group of models which can wrongly nullify the correct prediction since they represent the majority. Voting is beneficial only when the models perform in a similar way.

In the fraud detection world, all the presented approaches have been studied, demonstrating to achieve optimum performance.

Zareapoor et al [69] explained how applying bagging on the best fraud detection model found in their research, the decision tree, helps to improve the accuracy.

On the other hand, Randhawa et al [70] proposed a hybrid approach that consists of exploiting AdaBoost, one of the most common boosting methods, in combination with a majority voting between different models.

Finally, Soleymanzadeh et al [75] made use of an ensemble staking method to effectively detect credit card frauds and they tested the FDS on a real banking dataset.

Since different ensembling methods have obtained excellent results in the fraud detection domain, we decide to test every single approach on our dataset, in order to find a reliable and powerful Oracle from the attacker's point of view. In particular, we propose and compare three methods:

1. Bagging and Majority Voting: we select three different models. We perform bagging for each model by dividing our dataset into 10 or 20 equal parts. Finally, we execute a soft majority voting between the final models.

2. Boosting and Majority Voting: we select three different models. We exploit AdaBoost for each model with 10 or 20 estimators. Then, we execute a soft majority voting between the final models.

3. Stacking: we choose three models as base models and one as a meta-model.

It's difficult to estimate a priori which are the best models to choose for creating an effective ensemble: hence, we need to explore at least a portion of the entire set of com-

binations. We study the ensembling of the methods that perform best and, for computational reasons, we decide to conduct these experiments with one-quarter of our dataset, a consistent reduction but enough to understand which solution outperforms the others. Moreover, we choose the parameters, such as the bootstrap and estimators number, according to our dataset length and our computational requirements. We can evaluate each solution according to our custom score because the adversary may want to be cautious and an Oracle with a higher amount of false positives but a small one of false negatives. On the other hand, the attacker may want to avoid also false positives, in order to avoid regenerating unnecessarily a fraud. This is why we evaluate our results according to Recall, $F1$-Score, and FPR.

In Table 7.1, 7.2 and Table 7.3 we show our results concerning Bagging and Boosting used in conjunction with Majority Voting, and Stacking:

Oracle	Models	Bootstraps	Recall	F1	P-acc	FPR
Bagging + Majority Voting	XGBoost LightGB CatBoost	10	93.76%	59.1%	96.34%	1.09%
Bagging + Majority Voting	Random Forest XGBoost CatBoost	10	91.37%	64.49%	95.31%	0.73%
Bagging + Majority Voting	Random Forest LightGB CatBoost	10	91.13%	68.28%	95.23%	0.67%
Bagging + Majority Voting	Random Forest XGBoost Logistic Regression	10	92.57%	55.58%	95.66%	1.24%
Bagging + Majority Voting	Random Forest LightGB Logistic Regression	10	92.81%	59.36%	95.87%	1.06%
Bagging + Majority Voting	Random Forest CatBoost Logistic Regression	10	92.57%	69.86%	95.96%	0.64%
Bagging + Majority Voting	XGBoost LightGB Logistic Regression	10	94.00%	50.65%	96.22%	1.57%
Bagging + Majority Voting	XGBoost CatBoost Logistic Regression	10	94.24%	58.27%	96.55%	1.14%
Bagging + Majority Voting	Random Forest XGBoost LightXGBoost	10	93.29%	57.33%	96.06%	1.17%

Table 7.1: Bagging and Majority Voting Oracle Performance

Oracle	Models	Estimators	Recall	F1	P-acc	FPR
AdaBoost + Majority Voting	XGBoost LightGB CatBoost	10	79.6%	85.9%	89.78%	0.04%
AdaBoost + Majority Voting	Random Forest XGBoost CatBoost	10	82.09%	84.08%	90.99%	0.04%
AdaBoost + Majority Voting	Random Forest LightGB CatBoost	10	80.6%	85.26%	90.26%	0.07%
AdaBoost + Majority Voting	Random Forest XGBoost Logistic Regression	10	84.55%	86.13%	92.22%	0.09%
AdaBoost + Majority Voting	Random Forest LightGB Logistic Regression	10	72.25%	83.13%	86.12%	0.01%
AdaBoost + Majority Voting	Random Forest CatBoost Logistic Regression	10	84.82%	86.98%	92.37%	0.08%
AdaBoost + Majority Voting	XGBoost LightGB Logistic Regression	10	1.31%	2.36%	50.61%	0.08%

Table 7.2: AdaBoost and Majority Voting Oracle Performance

Oracle	Base Models	Meta Model	Recall	F1	P-acc	FPR
Stacking	XGBoost LightGB CatBoost	Logistic Regression	97.00%	23.49%	95.83%	5.33%
Stacking	XGBoost Logistic Regression CatBoost	LightGB	96.00%	31.54%	96.25%	3.5%
Stacking	Logitstic Regression LightGB CatBoost	XGBoost	96.00%	34.63%	96.48%	3.04%
Stacking	XGBoost Logistic Regression LightGB	CatBoost	94.00%	38.17%	95.73%	2.53%

Table 7.3: Stacking Oracle Performance

We notice that Bagging combined with majority voting results in high proportional accuracy, low False Positive Rate and $F1$-Score, and a medium Recall. This happens because this approach helps to keep small the number of false negatives, at the expense of false positives. On the contrary, with Boosting we achieve a very low percentage of false alerts and a higher F_1 Score. However, we have a lower Recall and Proportional Accuracy. This

aspect reflects what we said before: Boosting tends to overfit our data, it has a restricted training error but a more consistent test error, while Bagging in general tries to reduce the variance. Finally, the Stacking approach guarantees good Proportional Accuracy, and excellent Recall, but unacceptable F1 score and False Positive Rate. In conclusion, after analyzing every single oracle, we can state that the best approach is Bagging with majority voting between XGBoost, CatBoost and Logistic Regression.

At this point, we verify if using Majority Voting with Bagging is really beneficial. As mentioned before, Majority Voting could be useless or even counterproductive, when a larger group of models miss-predict a transaction correctly classified by another one. This is why we decided to perform one step more and inspect what happens if we exploit Bagging with just one model. Moreover, since we do not use Majority Voting which is computationally expensive, we can increase the number of bootstraps to 20. In Table 7.4 we report the results for all the best models.

Oracle	Model	Bootstraps	Recall	F1	P-acc	FPR
Bagging	CatBoost	20	93.36%	74.17%	96.44%	0.49%
Bagging	Active Learning	20	93.76%	59.10%	96.32%	1.09%
Bagging	LightGB	20	94.89%	53.71%	96.79%	1.31%
Bagging	XGBoost	20	93.87%	47.24%	96.10%	1.70%
Bagging	Random Forest	20	82.14%	64.08%	90.76%	0.62%
Bagging	Logistic Regression	20	92.60%	39.33%	95.14%	2.31%

Table 7.4: Bagging Oracle Performance

We can state that Bagging without Majority Voting works better. Another essential consideration is that Bagging influences in different ways the models. In Section 4.7 we have demonstrated that Active Learning (Isolation Forest and CatBoost) achieves the best results. On the other hand, if we observe the performance indexes in Table, 7.4, we notice that the models which gain more from Bagging are Light GrBoosting and CatBoost. In particular, CatBoost has a higher F1-Score and a lower FPR, whereas LightGB has higher proportional accuracy and a higher Recall. We have decided to use LightGB as our Oracle because we prefer a model which is more stealthy, that maybe arises false alarms, but minimizes the false negatives.

After all these considerations, in order to create our Oracle, we decided to adopt a Light Gradient Boosting model, which is improved by Bagging with 20 bootstraps.

7.3. Poisoning Process Results

In this section, we show our numerical results. Our poisoning attacks affect 15 victims, chosen according to their spending pattern and their nationality (national or foreign). We provide results for each scenario (White Box, Grey Box, Black Box), for each update policy (weekly or bi-weekly), and for each strategy (poisoning both, poisoning amount, and poisoning count). In order to decrease the bias associated with the victims, we performed each iteration twice, and then we averaged the results.

				White Box							
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative	Injection Rate (%)	Nat	42.65	31.73	53.3	42.76	26.51	45.51	74.9	64.08
			For	21.1	20.24	17.54	21.26	23.77	22.43	43.9	21.76
		Evasion Rate (%)	Nat	100	100	100	100	100	100	100	100
			For	100	100	100	100	100	100	100	100
		Detection Rate (%)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Detection Time (days)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Money Stolen (€)	Nat	8,733,248	6,333,868	8,891,680	8,718,549	5,554,781	8,124,589	11,258,042	10,550,761
			For	518,874	260,121	200,569	268,075	301,606	201,439	31,323	351,496
		Weekly Increase (%)	Nat	280	254	281	278	239	272	293	289
			For	321	283	272	285	290	273	101	295
	Greedy	Injection Rate (%)	Nat	28.51	22.48	38.45	36.61	28.74	35.56	65.31	45.48
			For	15.6	14.8	11.2	15.78	21.03	16.77	33.32	19.8
		Evasion Rate (%)	Nat	100	100	100	100	100	100	100	100
			For	100	100	100	100	100	100	100	100
		Detection Rate (%)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Detection Time (days)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Money Stolen (€)	Nat	35,420,881	29,201,213	33,455,720	38,101,708	22,588,411	31,681,320	45,465,321	38,602,552
			For	1,908,249	979,405	801,789	1,122,012	888,772	787,239	65,588	1,230,801
		Weekly Increase (%)	Nat	421	399	418	427	382	415	472	429
			For	475	411	402	464	407	399	109	466
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative	Injection Rate (%)	Nat	38.59	34.89	51.61	42.76	37.36	50.76	80.06	51.77
			For	26.43	25.71	23.44	21.26	31.82	25.78	40.0	26.43
		Evasion Rate (%)	Nat	100	100	100	100	100	100	100	100
			For	100	100	100	100	100	100	100	100
		Detection Rate (%)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Detection Time (days)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Money Stolen (€)	Nat	2,073,919	1,675,701	2,191,912	1,987,482	974,924	2,089,803	2,674,183	2,143,119
			For	175,240	103,973	95,677	84,006	120,677	97,764	69,246	93,702
		Weekly Increase (%)	Nat	223	216	225	220	209	224	232	226
			For	272	258	255	249	265	257	230	253
	Greedy	Injection Rate (%)	Nat	42.51	29.63	55.43	47.81	33.92	53.29	78.92	54.84
			For	21.08	22.59	16.67	18.25	26.51	18.72	32.51	18.71
		Evasion Rate (%)	Nat	100	100	100	100	100	100	100	100
			For	100	100	100	100	100	100	100	100
		Detection Rate (%)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Detection Time (days)	Nat	-	-	-	-	-	-	-	-
			For	-	-	-	-	-	-	-	-
		Money Stolen (€)	Nat	9,780,892	5,701,339	7,640,858	11,547,893	1,660,824	9,541,390	13,434,301	12,450,311
			For	190,541	89,853	75,515	150,777	158,571	91,512	65,223	121,565
		Weekly Increase (%)	Nat	277	249	272	287	217	275	295	291
			For	281	247	248	269	272	250	226	262

Table 7.5: White Box Attacks

				Grey Box							
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative	Injection Rate (%)	Nat	6.55	7.24	9	5.52	7.71	6.01	6.78	6.8
			For	3.29	3.67	4.55	1.83	2.29	5.12	5.81	3.21
		Evasion Rate (%)	Nat	100	99.59	98.75	99.64	98.11	98.70	99.63	99.19
			For	100	100	93.94	100	98.71	100	97.1	100
		Detection Rate (%)	Nat	-	27.27	63.64	27.27	72.72	63.64	27.27	54.55
			For	-	-	50	-	50	-	50	-
		Detection Time (days)	Nat	-	47	43	46.47	38.5	52	45	51.5
			For	-	-	15.5	-	16	-	33.5	-
		Money Stolen (€)	Nat	2,178,902	1,935,610	1,219,945	1,981,969	968,439	1,311,237	2,286,037	1,952,087
			For	41,572	30,218	23,290	38,560	35,915	27,450	29,779	33,756
		Weekly Increase (%)	Nat	161	152	152	153	142	155	156	149
			For	112	116	92	121	107	95	90	140
	Greedy	Injection Rate (%)	Nat	1.45	1.37	1.57	2.15	1.79	1.54	1.22	1.23
			For	2.29	2.41	2.12	2.95	2.12	2.01	2.83	2.94
		Evasion Rate (%)	Nat	99.47	99.72	98.54	98.89	99.35	99.62	98.52	99.71
			For	99.28	99.57	98.11	99.54	98.54	99.83	98.58	99.55
		Detection Rate (%)	Nat	18.18	63.64	72.73	36.36	100	72.73	36.36	72.73
			For	25	50	75	25	75	50	100	25
		Detection Time (days)	Nat	52	51	45.52	50	32.63	48.5	39.52	47.12
			For	18	57.5	23.48	47.5	29.95	52	12.58	49
		Money Stolen (€)	Nat	5,933,201	4,509,971	3,235,211	6,762,501	1,279,401	3,510,475	8,784,333	4,051,305
			For	622,391	606,454	470,349	650,671	201,711	451,532	59,421	180,184
		Weekly Increase (%)	Nat	186	182	175	189	147	177	195	179
			For	189	186	180	191	159	178	102	161
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative	Injection Rate (%)	Nat	16.22	17.53	17.81	17.36	17.01	17.12	17.27	16.45
			For	10.91	11.43	8.51	9.29	9.92	8.02	13.95	8.57
		Evasion Rate (%)	Nat	99.85	99.82	98.99	99.82	98.75	99.95	100	100
			For	100	100	96.88	100	99.75	100	95.96	100
		Detection Rate (%)	Nat	9.09	9.09	45.45	9.09	54.54	9.09	-	-
			For	-	-	25	-	25	-	50	-
		Detection Time (days)	Nat	60	16	50	58	46	59	-	-
			For	-	-	56	-	50	-	30.5	-
		Money Stolen (€)	Nat	1,525,357	1,016,85	972,268	1,316,138	996,755	1,002,137	1,420,298	1,386,852
			For	49,571	45,036	40,676	44,993	47,392	46,543	31,491	47,843
		Weekly Increase (%)	Nat	141	138	136	141	138	139	136	136
			For	149	115	98	111	114	115	133	113
	Greedy	Injection Rate (%)	Nat	9.54	9.86	9.72	10.42	9.32	9.33	9.02	9.14
			For	10	9.5	10.31	10	10.22	10.42	9.95	10.5
		Evasion Rate (%)	Nat	99.26	99.46	99.31	99.6	98.68	99.54	100	100
			For	100	100	99.45	100	100	100	99.33	100
		Detection Rate (%)	Nat	54.55	36.36	54.55	27.27	63.64	27.27	-	-
			For	-	-	25	-	-	-	75	-
		Detection Time (days)	Nat	53.83	42.25	47.65	39.67	38.5	42.58	-	-
			For	-	-	50	-	-	-	22	-
		Money Stolen (€)	Nat	1,642,965	1,722,892	1,121,364	1,695,882	1,012,141	1,598,774	2,015,773	1,913,449
			For	37,778	36,547	35,554	36,870	51,290	35,421	29,452	40,396
		Weekly Increase (%)	Nat	148	135	141	145	144	132	142	133
			For	111	132	107	144	152	110	115	153

Table 7.6: Grey Box Attacks: Poisoning Both

				Grey Box							
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative Amount	Injection Rate (%)	Nat	22.64	24.19	23.68	22.03	24.54	22.22	23.46	22.31
			For	10.23	15.12	9.52	10.23	14.77	11.73	13.85	9.2
		Evasion Rate (%)	Nat	99.75	98.72	97.88	99.75	97.25	98.81	100	99.15
			For	100	98.53	100	100	100	98.51	97.87	100
		Detection Rate (%)	Nat	9.09	45.45	72.73	9.09	63.64	27.27	-	27.27
			For	-	-	50	-	-	25	25	-
		Detection Time (days)	Nat	57	43.4	57.5	59	33.71	42.5	-	39
			For	-	-	33.5	-	-	7	10	-
		Money Stolen (€)	Nat	1,923,720	1,584,052	1,788,769	1,938,419	960,885	1,114,234	2,014,528	1,612,753
			For	43,628	43,745	39,175	48,188	59,955	42,178	35,903	32,113
		Weekly Increase (%)	Nat	147	140	147	147	117	140	148	136
			For	114	142	120	115	123	112	108	107
	Greedy Amount	Injection Rate (%)	Nat	22.41	23.39	22.45	21.55	24.9	23.25	21.47	23.52
			For	10.23	9.25	9.72	9.43	9.38	9.56	9.23	9.81
		Evasion Rate (%)	Nat	100	98.32	96.28	98.52	96.6	97.79	100	98.5
			For	100	98.75	98.43	100	96.77	98.82	97.12	100
		Detection Rate (%)	Nat	-	54.54	90.9	18.18	72.73	36.36	-	36.36
			For	-	25	75	-	25	50	75	-
		Detection Time (days)	Nat	-	45.6	42	52.5	31.38	40	-	43.5
			For	-	-	21.5	-	10	22.5	8.5	-
		Money Stolen (€)	Nat	2,300,756	1,890,820	1,472,882	2,170,199	880,418	1,230,331	2,983,612	1,992,422
			For	46,009	39,442	31,527	59,671	43,726	45,521	29,492	41,348
		Weekly Increase (%)	Nat	152	149	141	150	104	111	153	141
			For	115	135	113	127	121	125	102	118
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative Amount	Injection Rate (%)	Nat	28.76	28.77	28.92	27.41	27.49	27.78	27.66	27.91
			For	27.91	20.73	18	26.14	25	25.51	21.95	15.85
		Evasion Rate (%)	Nat	100	100	99.39	100	99.7	100	100	99.74
			For	100	100	96.67	100	95	100	100	100
		Detection Rate (%)	Nat	-	-	18.18	-	9.09	-	-	9.09
			For	-	-	25	-	50	-	-	-
		Detection Time (days)	Nat	-	-	45.45	-	3	-	-	60
			For	-	-	44	-	30	-	-	-
		Money Stolen (€)	Nat	1,119,353	1,025,160	864,941	1,174,870	910,211	1,023,341	1,188,937	1,147,223
			For	45,535	53,909	41,117	40,368	40,181	49,512	52,872	49,021
		Weekly Increase (%)	Nat	129	128	117	130	116	120	131	130
			For	117	118	112	116	105	116	113	115
	Greedy Amount	Injection Rate (%)	Nat	28.12	27.44	28.11	29.12	26.19	29.15	27.41	26.54
			For	27.27	22.89	23.91	25.45	20.73	24.72	24.39	23.21
		Evasion Rate (%)	Nat	100	99.58	98.37	100	99.4	99.37	100	99.31
			For	100	100	96.78	100	100	100	100	100
		Detection Rate (%)	Nat	-	9.09	27.27	-	18.18	9.09	-	27.27
			For	-	-	25	-	-	-	-	-
		Detection Time (days)	Nat	-	51	49	-	31	45	-	35.5
			For	-	-	56	-	-	-	-	-
		Money Stolen (€)	Nat	1,266,827	1,226,634	1,013,129	1,543,260	1,100,688	1,264,692	1,346,367	1,401,320
			For	57,949	60,041	46,595	47,887	54,070	54,771	58,924	55,541
		Weekly Increase (%)	Nat	132	131	119	133	115	125	134	136
			For	122	125	114	119	114	131	119	121

Table 7.7: Grey Box attacks: Poisoning Amount

				Grey Box							
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative Count	Injection Rate (%)	Nat	6.12	5.83	5.55	5.85	7.72	5.98	6.81	6.19
			For	10.76	11.47	9.83	7.34	15.62	10.66	11.9	8.26
		Evasion Rate (%)	Nat	100	99.77	99.14	99.53	98.68	99.69	99.63	99.89
			For	100	100	98.26	100	88	97.5	96.23	100
		Detection Rate (%)	Nat	-	18.18	54.54	36.36	72.73	36.36	27.27	9.09
			For	-	-	50	-	75	25	50	-
		Detection Time (days)	Nat	-	25.5	45.25	47.75	42.25	52.22	43.33	49
			For	-	-	38	-	18.67	21.5	42.5	-
		Money Stolen (€)	Nat	1,250,611	1,172,173	1,150,799	1,231,771	851,342	921,433	1,170,488	1,186,580
			For	20,322	17,114	11,506	14,571	9,556	10,782	14,016	13,464
		Weekly Increase (%)	Nat	165	160	149	155	136	138	172	171
			For	115	109	85	96	73	82	89	112
	Greedy Count	Injection Rate (%)	Nat	4.21	4.75	5.41	3.92	4.14	4.56	3.99	5.25
			For	9.72	8.59	9.32	9.44	9.73	8.78	8.89	9.11
		Evasion Rate (%)	Nat	99.13	99.11	98.72	99.23	96.55	98.76	99.65	99.45
			For	99.15	98.98	98.72	99.13	95.18	98.52	98.29	99.22
		Detection Rate (%)	Nat	18.18	54.54	72.73	54.54	54.54	72.73	54.54	45.45
			For	25	25	75	25	100	75	100	25
		Detection Time (days)	Nat	42	36.6	39.5	52.25	35.45	56	48.42	51.5
			For	35	41	29.5	45	13	25.75	19.28	32
		Money Stolen (€)	Nat	1,092,782	1,000,252	997,343	1,356,420	672,311	1,058,879	1,207,546	1,181,620
			For	16,721	11,221	7,213	12,998	3,222	6,402	4,103	8,158
		Weekly Increase (%)	Nat	150	147	132	161	115	141	165	168
			For	111	95	89	103	75	86	82	92
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative Count	Injection Rate (%)	Nat	15.94	16.84	16.37	16.28	15.61	16.34	16.67	16.28
			For	18.51	17.86	21.45	19.29	30.56	20.15	20.18	20
		Evasion Rate (%)	Nat	100	99.83	99.45	100	99.42	100	99.83	100
			For	100	100	99.32	100	87.88	100	98.75	100
		Detection Rate (%)	Nat	-	9.09	9-09	-	27.27	-	9.09	-
			For	-	-	25	-	100	-	25	-
		Detection Time (days)	Nat	-	48	43	-	21.33	-	60	-
			For	-	-	12	-	26.5	-	30	-
		Money Stolen (€)	Nat	728,472	752,212	709,332	713,060	714,761	752,890	765,419	755,390
			For	25,498	24,372	21,107	26,482	14,122	16,433	22,069	22,698
		Weekly Increase (%)	Nat	144	131	129	143	96	129	132	152
			For	136	129	123	137	89	101	122	141
	Greedy Count	Injection Rate (%)	Nat	8.07	7.91	7.82	8.92	7.98	8.52	8.22	8.13
			For	18.73	21.5	19.89	19.1	19.26	18.88	18.67	19.13
		Evasion Rate (%)	Nat	100	99.9	99.38	100	98.31	98.88	99.93	99.87
			For	100	100	99.63	100	100	100	100	100
		Detection Rate (%)	Nat	-	9.09	36.36		90.9	27.27	18.18	9.09
			For	-	-	50	-	100	50	100	-
		Detection Time (days)	Nat	-	43	47.5	-	31.5	50.75	57.5	41
			For	-	-	28	-	14.25	22.5	27.5	-
		Money Stolen (€)	Nat	1,102,997	1,096,883	687,302	998,730	482,773	885,421	995,708	1,045,779
			For	31,792	29,373	16,551	32,452	6,520	13,702	7,158	26,891
		Weekly Increase (%)	Nat	163	152	113	156	102	133	150	151
			For	119	117	107	119	81	95	84	115

Table 7.8: Grey Box Attacks: Poisoning Count

Black Box

		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative	Injection Rate (%)	Nat	13.1	15.54	14.16	13.55	13.48	14.33	14.01	14.32
			For	7.12	6.9	5.88	9.59	0	7.21	0	5.91
		Evasion Rate (%)	Nat	100	99.12	98.32	99	98.16	98.42	99.27	98.64
			For	100	90	88.24	100	89.5	100	0	98.25
		Detection Rate (%)	Nat	-	27.27	45.45	36.36	36.36	36.36	27.28	54.55
			For	-	50	50	-	100	50	100	25
		Detection Time (days)	Nat	-	30.67	42.4	49	13.75	52.24	35.33	57.17
			For	-	10	16.5	-	0	25	0	31
		Money Stolen (€)	Nat	1,658,038	1,022,444	849,368	1,428,393	491,297	1,005,478	1,592,290	1,526,330
			For	51,326	29,419	26,435	56,737	0	27.720	0	21,505
		Weekly Increase (%)	Nat	163	156	141	162	89	145	172	163
			For	156	143	132	158	0	133	0	129
	Greedy	Injection Rate (%)	Nat	13.75	12.45	12.54	19.02	15.71	14.44	11.84	18.37
			For	5.7	7.14	8.51	6.54	0	6.76	0	6.33
		Evasion Rate (%)	Nat	99.27	98.39	98.17	99.3	97.6	98.42	98.92	99.62
			For	100	80	91.3	99.03	0	92.57	0	100
		Detection Rate (%)	Nat	36.36	63.64	54.55	27.27	72.72	54.55	45.45	18.18
			For	-	75	50	25	100	50	100	-
		Detection Time (days)	Nat	49.25	40.14	31.83	35	18.75	34.48	43.8	49.5
			For	-	12	16.5	59	0	21	0	-
		Money Stolen (€)	Nat	3,260,425	1,595,950	914,065	3,180,277	661,509	1,111,708	1,978,910	4,054,445
			For	97,670	21,575	30,332	79,798	0	30,998	0	101.055
		Weekly Increase (%)	Nat	191	150	145	190	125	147	182	199
			For	208	137	138	266	0	139	0	212
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative	Injection Rate (%)	Nat	13.63	15	19.05	14.35	14.27	16.39	13.29	13.38
			For	11.86	8.93	11.11	8.64	0	9.51	0.1	5.77
		Evasion Rate (%)	Nat	99.32	98.66	96.93	99.05	98.21	98.58	99.54	99.1
			For	100	90	92	96.67	0	92.57	20	98.55
		Detection Rate (%)	Nat	27.27	45.45	81.82	36.36	54.55	36.36	18.18	36.36
			For	-	50	50	50	100	50	100	25
		Detection Time (days)	Nat	51	33.8	37.22	41.5	25.5	54.65	23.5	52.5
			For	-	6.5	16	26	0	31	3.5	44
		Money Stolen (€)	Nat	1,530,935	1,212,172	733,666	1,497,813	505,888	1,170,102	1,658,829	1,530,502
			For	50,763	54,100	50,150	31,821	0	30,402	201	23,472
		Weekly Increase (%)	Nat	152	151	150	167	95	147	141	159
			For	155	202	310	257	0	135	93	231
	Greedy	Injection Rate (%)	Nat	18.22	10.54	13.68	15.04	10.97	11.15	10.31	13.16
			For	3.8	7.35	9.52	4.43	0	5.41	6	5.38
		Evasion Rate (%)	Nat	99.21	97.53	96.44	98	96.82	97.62	99.06	98.99
			For	100	90.48	87.5	100	0	87.8	20	98.81
		Detection Rate (%)	Nat	45.45	90.91	90.91	90.91	90.91	90.91	45.45	54.55
			For	-	50	75	-	100	75	100	25
		Detection Time (days)	Nat	49.6	36.4	32	45.1	31.3	38.5	38	47.33
			For	-	10	25.33	-	-	27.5	3.5	42
		Money Stolen (€)	Nat	4,810,805	1,321,238	552,270	2,616,346	828,549	1,451,413	2,138,938	3,293,494
			For	43,663	70,406	50,955	68,894	0	52,671	1,115	60,797
		Weekly Increase (%)	Nat	189	146	164	180	121	148	154	182
			For	248	222	182	350	0	183	26	209

Table 7.9: Black Box Attacks: Poisoning Both

Black Box

		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative Amount	Injection Rate (%)	Nat	22.16	24.44	24.89	24.91	23,95	24.56	22.21	24.17
			For	10.26	15.62	14.29	11.54	12.44	13.71	12.19	11.62
		Evasion Rate (%)	Nat	99.37	97.39	96.1	98.21	97.17	98.12	98.72	98.81
			For	100	85	91.3	100	100	98.67	98.54	98.71
		Detection Rate (%)	Nat	18.18	63.64	81.82	45.45	72.73	36.36	18.18	36.36
			For	-	75	50	-	-	25	25	25
		Detection Time (days)	Nat	60	46.29	40.44	60	30.31	38.82	21	37.5
			For	-	16.67	23.5	-	-	51.5	15	50
		Money Stolen (€)	Nat	1,544,023	1,061,736	769,669	1,566,890	781,309	997,552	1,762,124	1,409,019
			For	65,805	53,278	42,939	82,181	62,507	48,920	32,331	40,915
		Weekly Increase (%)	Nat	202	170	161	210	142	168	209	200
			For	285	186	142	289	283	178	139	151
	Greedy Amount	Injection Rate (%)	Nat	21.88	23.75	27.35	22.65	23.43	25.62	23.57	25.81
			For	11.54	19.44	7.69	13.85	10.92	11.44	0	13.33
		Evasion Rate (%)	Nat	98.12	97.87	97.72	98.36	100	100	98.18	99.34
			For	100	88	95.83	98.18	98.12	98.52	0	96.43
		Detection Rate (%)	Nat	54.55	45.45	45.45	45.45	100	36.36	45.45	18.18
			For	-	75	25	25	50	25	100	25
		Detection Time (days)	Nat	57.33	36.4	37	48	41.55	35.5	39.2	45.5
			For	-	21.33	17	16	17.5	45.5	0	3
		Money Stolen (€)	Nat	1,991,366	950,310	854,180	1,919,513	701,434	1,102,598	1,484,658	1,914,712
			For	81,520	53,529	72,848	41,822	48,872	52,533	0	86,116
		Weekly Increase (%)	Nat	179	162	173	221	122	169	217	231
			For	290	191	160	180	195	171	0	137
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative Amount	Injection Rate (%)	Nat	22.5	23.83	26.2	22.76	21.74	25.74	23.36	21.91
			For	11.54	13.89	16.67	11.76	0	12.93	0	13.79
		Evasion Rate (%)	Nat	97.76	97.6	96.04	98.68	96.41	97.12	98.65	98.59
			For	100	88.89	93.75	97.83	0	98.17	0	97.5
		Detection Rate (%)	Nat	63.64	54.55	81.82	36.36	72.73	54.55	36.36	36.36
			For	-	50	50	25	100	25	100	25
		Detection Time (days)	Nat	56.71	36	38.89	38.25	33.62	40.5	37.5	34.75
			For	-	3	24	38	0	58	0	24
		Money Stolen (€)	Nat	1,586,556	1,007,285	739,457	1,767,987	880,111	1,110,561	1,511,972	1,568,341
			For	67,588	77,720	80,946	114,075	0	69,744	0	92,788
		Weekly Increase (%)	Nat	166	138	137	147	113	144	149	142
			For	201	159	212	239	0	195	0	221
	Greedy Amount	Injection Rate (%)	Nat	24.52	26.34	25.42	23.4	23.26	22.45	25.85	22.61
			For	14.1	16.67	17.5	18.37	0	17.49	0	15.77
		Evasion Rate (%)	Nat	97.4	96.86	96.14	98.02	96.63	100	98.62	98.38
			For	100	83.33	85.71	95	0	100	0	97.56
		Detection Rate (%)	Nat	72.73	63.64	81.82	54.55	63.64	63.64	36.36	45.45
			For	-	75	100	50	100	50	100	25
		Detection Time (days)	Nat	55.62	32.29	41.42	48	29	33.57	36.5	49.8
			For	-	14.33	33.75	16.5	0	27.5	0	3
		Money Stolen (€)	Nat	1,776,255	788,202	832,362	1,839,186	829,795	966,204	1,785,961	1,948,416
			For	101,213	54,426	84,465	40,134	0	74,721	0	82,625
		Weekly Increase (%)	Nat	173	144	148	163	122	134	154	173
			For	272	157	180	184	0	174	0	258

Table 7.10: Black Box Attacks: Poisoning Amount

				Black Box							
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative Count	Injection Rate (%)	Nat	14.53	15.54	17.05	16.27	16.12	17.31	14.92	15.84
			For	16.1	20	13.79	18.42	21.15	18.62	21.12	17.29
		Evasion Rate (%)	Nat	100	99.5	98.73	99.29	98.12	98.85	99.94	99.32
			For	100	100	86.21	99.12	86.11	97.65	82	99.16
		Detection Rate (%)	Nat	-	18.18	45.45	27.27	72.73	45.45	36.36	27.27
			For	-	-	100	25	100	50	100	25
		Detection Time (days)	Nat	-	31.5	41.4	43.33	35.78	37.15	56.42	51.25
			For	-	-	22.75	57	11.75	14	6.75	56.55
		Money Stolen (€)	Nat	967,459	778,842	751,995	881,094	501,391	772,422	1,011,842	927,458
			For	12,673	6,660	9,053	12,860	2,511	8,972	1,512	12,681
		Weekly Increase (%)	Nat	111	120	112	131	99	116	145	138
			For	87	70	90	94	39	91	21	96
	Greedy Count	Injection Rate (%)	Nat	9.97	9.47	11.99	10.96	9.56	10.13	11.94	10.86
			For	15.51	18.75	12.5	8.96	0	13.59	0	8.05
		Evasion Rate (%)	Nat	99.68	98.74	97.95	98.9	97.11	98.66	98.83	99.39
			For	99.31	81.25	83.33	97.01	0	82.29	0	96.55
		Detection Rate (%)	Nat	9.09	54.55	63.64	45.45	100	54.55	45.45	27.27
			For	25	75	100	50	100	75	100	75
		Detection Time (days)	Nat	57	46.67	31.14	49.8	35.51	41.19	37	54.67
			For	42	12	19.75	43.5	0	14.75	-	42.33
		Money Stolen (€)	Nat	1,211,532	1,048,271	602,258	1,024,305	412,814	815,992	1,081,316	1,139,537
			For	14,678	6,939	6,258	10,989	0	6,351	0	11,219
		Weekly Increase (%)	Nat	125	122	101	140	98	113	133	130
			For	101	75	82	90	0	78	0	99
		Metric	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative Count	Injection Rate (%)	Nat	14.29	15.75	17.91	16.3	16.97	16.83	15.19	16.19
			For	16.1	18.75	16.67	16.1	0	17.92	0	17.33
		Evasion Rate (%)	Nat	99.57	98.16	98.21	99.03	97.83	98.82	99.3	98.92
			For	100	75.0	86.67	100	0	97.44	0	98.91
		Detection Rate (%)	Nat	18.18	63.64	54.55	36.36	54.55	45.45	27.27	45.45
			For	-	100	100	-	100	50	100	50
		Detection Time (days)	Nat	57	40.14	33.5	38.75	16.17	33.8	28.67	44.35
			For	-	12.75	24.5	-	0	13.5	0	54.5
		Money Stolen (€)	Nat	967,569	793,717	495,650	841,898	469,120	712,847	963,389	709,443
			For	12,402	7,845	8,986	12,097	0	8,873	0	10,782
		Weekly Increase (%)	Nat	132	124	107	141	104	112	129	110
			For	112	67	109	112	0	90	0	111
	Greedy Count	Injection Rate (%)	Nat	12.45	10.48	13.01	9.62	11.54	11.03	11.23	10.14
			For	16.34	18.75	17.24	6.54	0	15.24	60	10.26
		Evasion Rate (%)	Nat	98.76	98.19	97.29	98	97.03	98.12	98.27	98.44
			For	100	75.0	86.21	97.2	0	95.41	20	96.15
		Detection Rate (%)	Nat	54.54	81.82	90.91	90.91	90.91	81.82	72.73	72.73
			For	50	100	100	75	100	100	100	75
		Detection Time (days)	Nat	41.66	42.56	38.7	44.2	29.1	35.44	37.88	45.62
			For	37.54	12.75	22.75	47.33	-	25.25	3.5	38.33
		Money Stolen (€)	Nat	1,271,530	1,130,181	612,214	1,260,724	482,488	750,666	1,077,267	1,298,233
			For	15,978	7,322	8,673	11,818	0	8,502	986	10,478
		Weekly Increase (%)	Nat	193	193	198	191	106	188	139	167
			For	189	100	149	182	0	151	24	183

Table 7.11: Black Box Attacks: Poisoning Count

7.3.1. White Box

In this scenario, the adversary has complete knowledge of the target system. Even if these kinds of attacks are really rare, they are helpful in understanding and analyzing the best case possible. White Box results will be the reference point for Grey and Black scenarios. The goal of the attacker is, despite his or her partial or no knowledge, to get as close as possible to the best possible situation. Since the adversary has complete knowledge, we list only the results related to the strategy in which he or she poisons both amount and count. In general, weekly attacks perform better than bi-weekly ones, because the poisoning process is faster. In the conservative strategy with the weekly update, the attacker can steal up to 10,550,761€ against an Active Learning detector. As we stated in Section 4.7, Active Learning is the most powerful model among those proposed. It means that there are no direct consequences between the accuracy of the FDSs and their reaction to poisoning attacks. XGBoost is the detector that counters best the national frauds, while Light Gradient Boosting works well against foreign malicious transactions. However, you can notice that for the bi-weekly update we get different results. In fact, in the conservative strategy, the best model against national fraud is still XGBoost, but Logistic Regression, which is the worst against them, outperforms other models regarding foreign ones. The amount of money stolen is higher in weekly update cases. Since the adversary is able to build a perfect replica of the target system, the evasion rate is always 100% while the detection rate is 0% for all models. The injection rates are always between 31.73%, achieved by XGBoost, and 80%, from Logistic Regression. It means that XGBoost pushes the attacker to regenerate the proposed frauds while Logistic Regression is weaker and doesn't detect them.

In Figure 7.1 and Figure 7.2, we provide a graphical overview of the trend related to the average amount of malicious transactions during the attack of a CatBoost system.

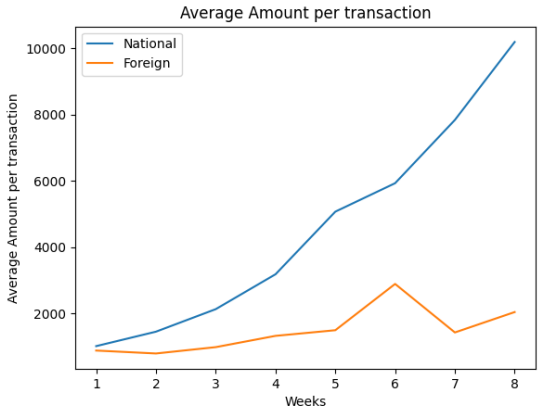

Figure 7.1: CatBoost, Weekly Policy, Conservative Strategy

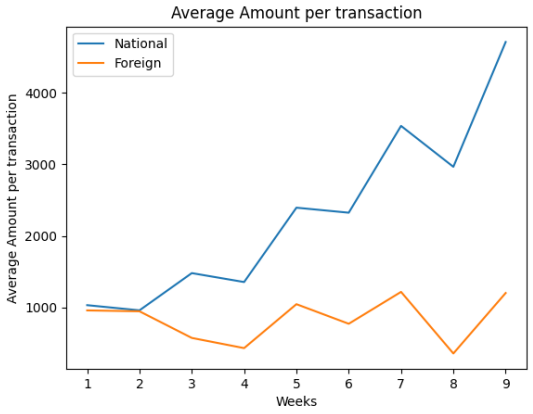

Figure 7.2: CatBoost, Bi-weekly Policy, Conservative Strategy

Figure 7.1 and Figure 7.2 show the poisoning process through the average amount per transaction. You may notice that the process is faster for a weekly update policy while it is more irregular when adopting a bi-weekly one. At the end of a weekly poisoning attack, the adversary can steal on average 10,000€ per transaction, just 5,000€ in a bi-weekly scenario. In addition, it's evident how national and foreign attacks behave very differently. It's easier to incrementally raise the average amount of national transactions, whereas it's more problematic for foreign ones. As we expected, for both national and foreign transactions, in the biweekly policy the attacker is able to steal on average 50% of what he or she can steal adopting a weekly policy.

7.3.2. Grey Box

Grey Box attacks are mounted by an adversary which has partial knowledge of the target system. For what concerns the standard (i.e., which poisons both count and amount) strategy against a machine with a weekly update, we obtain the following results: the detection rates are between 27% and 63% for national users and between 0% and 50% for foreign ones, the detection time is reasonably high (from 43 to 51.5 days) and the amount of stolen money is almost 0.25 with respect to the White Box scenario. The very important result is about the bi-weekly policy: in the conservative version of the strategy, the results of national users are very similar to those related to the White Box scenario. For instance, against the CatBoost target system, an adversary, with the conservative strategy, is able to steal 1,361,132€ over 2,071,482€ which are stolen in the White Box scenario. This means that our Oracle is extremely reliable. The greedy approach, if we look at the national frauds, allows the adversary to steal more money at the expense of a

higher detection rate. This result is even more evident against weekly updated detectors. On the other hand, an aggressive strategy is beneficial against foreign users in the weekly update, while it's counterproductive with bi-weekly updated systems. This is why foreign users are more difficult to attack and they are affected by a faster poisoning process. Again, we can state that Logistic Regression is the less powerful with national frauds, but the best against foreign ones. In addition, CatBoost and Active Learning are the detectors that perform worst with foreign malicious transactions. Another important aspect is that our Oracle is more restrictive about foreign transactions and, at least in conservative strategies, allows the attacker to be undetected in some cases, such as XGBoost and CatBoost. In general, the injection rates are really low (between 1.83% and 17.61%), because the Oracle pushes the adversary to regenerate the features very frequently.

Moreover, we analyze strategies that allow us to study the poisoning process related to the count and to the amount independently. If we focus on the conservative strategy which poisons only the amount feature, we notice that the attacker is able to steal a similar amount of money, slightly lower with respect to a standard strategy, but he or she is capable to decrease the attack detection rate in a consistent way. For example, against a Logistic Regression detector with a weekly policy, the fraudster steals 2,315,816€ with a conservative standard strategy while 2,050,431€ with a conservative amount one; however, in the first case 5 over the 15 attacks are detected, whereas in the second one only one is noticed. Poisoning just one feature makes the attacks more evasive and effective. In addition, we found out that for foreign transactions, this strategy is much more powerful, because you are able to increase the amount stolen and decrease the detection rate. This is true for every target system, for each strategy, and for each update policy. In the bi-weekly update, this is more evident: an attacker is able to steal 53,909€ from foreign users against XGBoost, which is more than the standard conservative strategy against XGBoost trained according to a weekly policy (30,218€). In this sense, bi-weekly attacks outperform weekly ones.

We also provide a strategy that consists of poisoning only the count, that is the number of transactions executed by a customer in one iteration. In this context, the adversary poisons only this aspect: he or she increments the number of banking operations performed by the victim during one or two weeks. The average amount of these transactions is replicated according to the victim's spending pattern. By looking at the results, we notice that this strategy doesn't bring any advantages to the fraudster. In fact, the amount of money stolen is always less with respect to the two previous strategies, especially for the weekly update and foreign transactions. However, this type of attack allows the attacker to decrease the detection rate in some cases: for example, considering a conservative

strategy against XGBoost trained according to a weekly policy, the adversary decreases the detection rate to 18.18%, which is less than 45.45% registered with the conservative amount strategy and 27.27%, associated to the standard strategy. On the contrary, that's not true for CatBoost, which presents an attack detection rate that is higher, 36.36%. This happens because the detectors are trained on different features, according to the feature selection task performed in Section 4.4. Figure 7.3, Figure 7.4 and Figure 7.5 present some plots about the mean amount stolen at each iteration, referring to the XGBoost detector trained according a bi-weekly policy.

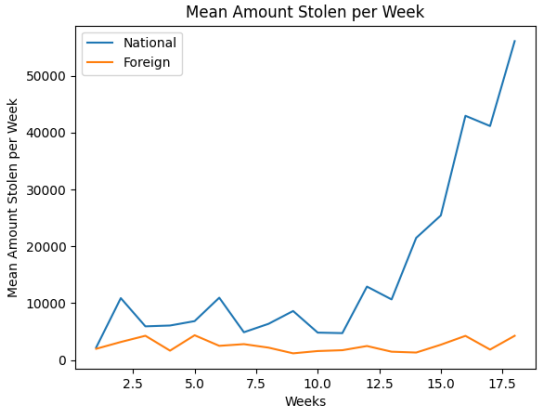

Figure 7.3: XGBoost, Bi-weekly Policy, Greedy Strategy

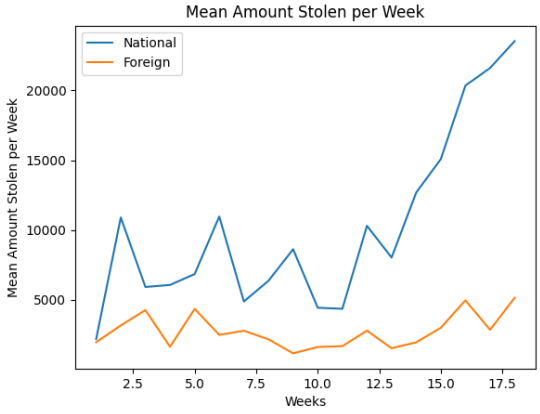

Figure 7.4: XGBoost, Bi-weekly Policy, Greedy Strategy Amount

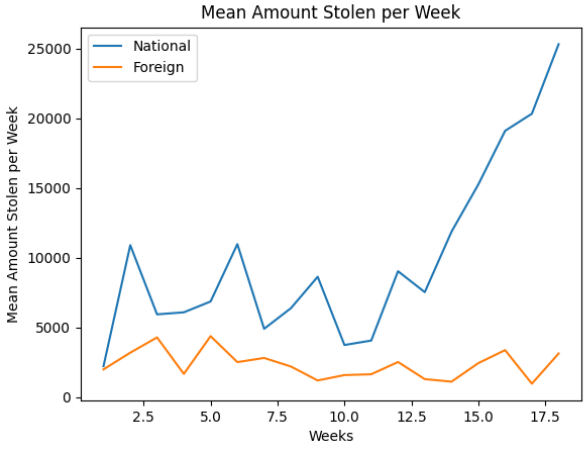

Figure 7.5: XGBoost, Bi-weekly Policy, Greedy Strategy Count

The adversary, exploiting a standard strategy, is able to steal, in a single iteration, more than 50,000€ from national users. However, a strategy that focuses on the amount, is

able to increase the stolen mean in a fast way. As we stated before, the count strategy is not helpful from this point of view. The presented plots show the trend of the mean before the attack and after the attack, which starts at week 10.

7.3.3. Black Box

In the Black Box scenario, the adversary doesn't know anything about the target systems. He or she trains an Oracle with a surrogate dataset in order to validate the frauds. Since he or she doesn't know the feature set, a subset of features is used. Moreover, differently from Grey Box, the attacker doesn't know the update policy of the banking detector. This is why the adversary chooses a weekly policy to update the Oracle, in order to make the poisoning process faster and steal as much money as possible.

Concerning the standard strategy and detectors with a weekly update policy, we can state that, in general, the results are worse than those of the Grey Box. This is why the attacker has a weaker Oracle and he or she adopts a weekly policy that makes him or her more suspicious. In fact, the attack detection rates are higher. However, the update policy used by the adversary is beneficial for foreign frauds crafted against some detectors: for example, considering a CatBoost detector, a standard conservative strategy allows to steal 56,737€ from foreign users in the Black Box scenario, while only 38,560€ in the Grey Box one. Notice that this doesn't apply to standard greedy strategies, which are much more effective in the Grey Box. We can observe that Support Vector Machine and Logistic Regression are completely resistant to foreign frauds. The attacker is not able to poison these two detectors, which don't accept even one foreign fraud. This result confirms what we stated in Section 7.3.2: SVM and LR are the most powerful models against not national frauds. Moreover, Support Vector Machine is the model from which the adversary steals the minimum amount of money. This happens because SVM presents a lot of false positives and this aspect makes it the most difficult detector to poison. We obtain better results with models trained with a bi-weekly update policy since the learning process is slower and so detectors are weaker. The adversary is able to steal more money with respect to the Grey Box scenario, with both conservative and greedy strategies. This is not true for Light Gradient Boosting, from which an attacker steals less money, 733,666€ against 972,268€ with a conservative approach. However, the attack detection rates are higher, since the adversary adopts a weekly update policy: Random Forest model detects 45.45% of national frauds crafted according to a greedy strategy, whereas 36.36% when trained with a weekly update policy. Regarding foreign fraudulent transactions, detectors behave very differently. Some models perform better if trained with a weekly update policy, such as Logistic Regression and XGBoost, others work well

with a bi-weekly update policy, such as CatBoost.

The poisoning amount strategy is very interesting in this scenario. Regarding national frauds, we don't have consistent advantages from this approach. The amount of stolen money is lower and the attack detection rate is higher with respect to the standard strategy. However, we obtain a significant result for foreign frauds. As we stated in the previous Section 7.3.2, focusing only on poisoning the amount is very beneficial against foreign victims. In this scenario, since the adversary makes use of a weekly update policy, this result is even more pronounced. For instance, in the Black Box scenario, considering a CatBoost detector, an attacker using a conservative amount strategy steals 114,075€ from foreign users, while in White Box just 84,006€. White Box attacks represent the best case possible, but with this approach, the attacker is able to overcome them. Notice that it's not true with a greedy strategy, through which the fraudster steals only 40,134%. In some cases, the attacker has benefits when adopting a more aggressive approach, such as when we attack Light Gradient Boosting or Random Forest; on the contrary, it's better to use a more cautious strategy when facing CatBoost or XGBoost detectors. In Figure 7.6, Figure 7.7 and Figure 7.8 we provide a visual overview of the attack detection rate trend in three different scenarios where the banking algorithm is Light Gradient Boosting.

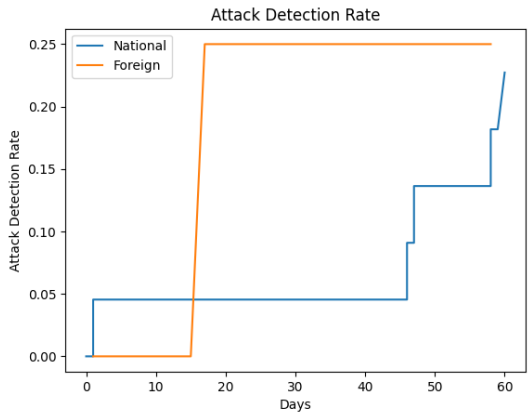

Figure 7.6: LightGB, Weekly Policy, Conservative Strategy

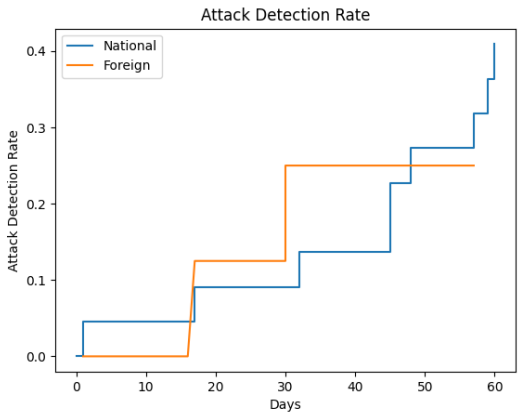

Figure 7.7: LightGB, Weekly Policy, Conservative Strategy Amount

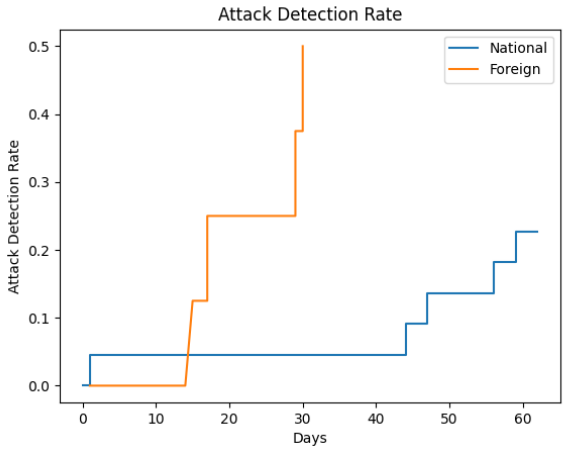

Figure 7.8: LightGB, Weekly Policy, Conservative Strategy Count

There are no cases in which all the attacks are detected. The attack detection rate, in these three combination, is never 100%. Regarding national attacks, they are detected at most 40% of the time. Instead, looking at the standard strategy and the amount strategy, foreign ones are detected on average 25% of the time while we get 50% if the adversary makes use of a count strategy. An important consideration is that foreign frauds are usually detected in the first 30 days, and then remain unnoticed.

7.4. Regeneration Process Results

In this section, we study and comment on the results of the regeneration process, by analyzing which are the features that the Oracle suggests to change. In the White Box scenario, the adversary has a perfect replica of the target FDS, while in the Grey and Black scenarios, he or she relies on an Oracle which is based on Bagging techniques. This is why we show the complete regeneration results for each detector in the White Box scenario, while we report the average values for the other two scenarios since the Oracle doesn't change because the adversary has no knowledge about the target machine learning algorithm. In Table 7.12 you can see the results regarding the regeneration task: for each of the 4 features that an adversary can change during the attack, we report the fraction between how many times a specific feature has been regenerated and the total number of frauds.

White Box

		Feature	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Weekly Update	Conservative	IP (%)	Nat	57,24	67.34	46.38	48.57	55.31	47.76	24.49	30.91
			For	77.98	79.16	81.87	77.77	67.27	80.79	56.01	75.64
		IBAN (%)	Nat	17.14	42.85	33.71	22.04	5.30	35.72	6.73	11.43
			For	72.47	70.23	73.09	74.39	10.02	72.13	43.9	66.32
		CC_ASN (%)	Nat	44.18	67.34	43.98	46.02	61.90	44.46	25.10	28.77
			For	78.44	79.16	81.87	77.29	68.18	80.34	56.09	75.64
		Amount (%)	Nat	57.34	68.26	46.70	57.24	62.63	51.17	25.10	35.92
			For	78.89	79.26	82.45	78.74	68.18	81.16	56.09	78.24
	Greedy	IP (%)	Nat	55.24	68.73	39.27	50.16	57.38	41.56	26.15	29.12
			For	78.09	81.19	72.44	79.90	71.15	81.27	58.43	77.78
		IBAN (%)	Nat	16.59	46.92	33.28	25.53	6.48	32.92	7.11	13.56
			For	74.50	71.18	70.37	75.59	12.40	74.15	47.22	64.48
		CC_ASN (%)	Nat	46.97	64.11	38.85	45.74	63.82	42.41	23.23	28.96
			For	81.14	76.85	72.44	80.82	69.15	78.81	57.68	78.94
		Amount (%)	Nat	58.46	70.47	39.46	58.39	63.44	54.77	24.08	38.55
			For	72.71	82.24	72.44	79.91	71.43	78.46	58.88	79.71
		Feature	User	RF	XGB	LGB	CB	SVM	ANN	LR	AL
Bi-weekly Update	Conservative	IP (%)	Nat	61,25	61.57	46.46	52.25	57.27	46.82	19.93	39.54
			For	70.71	74.28	75.78	75.38	62.27	78.12	60	72.14
		IBAN (%)	Nat	11.73	29.26	20.73	14.63	4.76	27.63	2.89	10.93
			For	67.14	69.28	67.96	70.76	12.49	69.15	55	72.14
		CC_ASN (%)	Nat	54.82	64.30	44.05	49.19	63.82	41.56	19.77	35.04
			For	71.42	73.57	75.78	75.38	69.11	76.43	60	73.57
		Amount (%)	Nat	61.41	65.11	48.39	63.02	64.89	49.94	19.93	48.23
			For	73.57	74.28	76.56	76.15	72.13	78.51	60	73.57
	Greedy	IP (%)	Nat	66.31	68.44	43.29	51.43	59.46	48.85	25.63	47.12
			For	75.23	77.14	82.75	77.73	65.10	79.56	71.15	73.81
		IBAN (%)	Nat	15.29	37.48	22.32	18.24	7.82	29.18	8.98	15.25
			For	70.12	78.45	75.86	75.67	14.55	72.54	61.33	76.14
		CC_ASN (%)	Nat	55.23	67.55	42.44	52.08	67.66	43.87	25.46	41.65
			For	79.21	81.46	82.75	79.91	71.99	78.21	65.12	77.22
		Amount (%)	Nat	62.72	68.29	44.56	65.19	68.11	52.49	23.04	54.18
			For	81.86	86.17	83.33	79.91	81.17	80.34	73.98	77.12

Table 7.12: White Box Regeneration Results

Grey Box												
		Feature	User	Detectors		Feature	User	Detectors		Feature	User	Detectors
Weekly Update	Conservative	IP (%)	Nat	87.98	Conservative Am.	IP (%)	Nat	71.97	Conservative Count	IP (%)	Nat	91.49
			For	82.56			For	54.65			For	74.77
		IBAN (%)	Nat	81.13		IBAN (%)	Nat	73.45		IBAN (%)	Nat	85.43
			For	63.76			For	45.44			For	67.88
		CC_ASN (%)	Nat	5.38		CC_ASN (%)	Nat	4.11		CC_ASN (%)	Nat	7.09
			For	16.51			For	12.79			For	4.13
		Amount (%)	Nat	83.72		Amount (%)	Nat	61.06		Amount (%)	Nat	81.93
			For	96.33			For	84.88			For	88.53
	Greedy	IP (%)	Nat	96.68	Greedy Amount	IP (%)	Nat	69.79	Greedy Count	IP (%)	Nat	93.75
			For	91.83			For	52.30			For	80.11
		IBAN (%)	Nat	92.30		IBAN (%)	Nat	71.02		IBAN (%)	Nat	87.14
			For	88.35			For	40			For	71.54
		CC_ASN (%)	Nat	10.12		CC_ASN (%)	Nat	6.12		CC_ASN (%)	Nat	8.24
			For	15.89			For	13.84			For	11.50
		Amount (%)	Nat	94.83		Amount (%)	Nat	53.46		Amount (%)	Nat	89.97
			For	95.85			For	92.30			For	91.18
		Feature	User	Detectors		Feature	User	Detectors		Feature	User	Detectors
Bi-weekly Update	Conservative	IP (%)	Nat	76.20	Conservative Am.	IP (%)	Nat	56.98	Conservative Count	IP (%)	Nat	78.64
			For	66.42			For	35.36			For	62.85
		IBAN (%)	Nat	79.06		IBAN (%)	Nat	69.31		IBAN (%)	Nat	80.73
			For	66.42			For	43.90			For	62.14
		CC_ASN (%)	Nat	2.38		CC_ASN (%)	Nat	1.16		CC_ASN (%)	Nat	2.11
			For	17.14			For	9.31			For	5.7
		Amount (%)	Nat	63.68		Amount (%)	Nat	42.19		Amount (%)	Nat	60.93
			For	88.57			For	79.26			For	82.14
	Greedy	IP (%)	Nat	85.54	Greedy Amount	IP (%)	Nat	63.09	Greedy Count	IP (%)	Nat	87.88
			For	76.50			For	42.68			For	65.13
		IBAN (%)	Nat	85.01		IBAN (%)	Nat	71.72		IBAN (%)	Nat	85.31
			For	69			For	48.78			For	60.21
		CC_ASN (%)	Nat	7.41		CC_ASN (%)	Nat	2.24		CC_ASN (%)	Nat	5.54
			For	21.34			For	5.7			For	10.52
		Amount (%)	Nat	78.97		Amount (%)	Nat	46.42		Amount (%)	Nat	77.61
			For	90.5			For	79.76			For	76.45

Table 7.13: Grey Box Regeneration Results

Black Box

		Feature	User	Detectors		Feature	User	Detectors		Feature	User	Detectors
Weekly Update	Conservative	IP (%)	Nat	74.78	Conservative Am.	IP (%)	Nat	63.70	Conservative Count	IP (%)	Nat	78.19
			For	62.06			For	34.38			For	27.27
		IBAN (%)	Nat	74.48		IBAN (%)	Nat	69.26		IBAN (%)	Nat	79.20
			For	62.06			For	40.63			For	26.25
		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0
			For	0			For	0			For	0
		Amount (%)	Nat	62.27		Amount (%)	Nat	47.40		Amount (%)	Nat	49.87
			For	93.10			For	84.37			For	80.32
	Greedy	IP (%)	Nat	79.25	Greedy Amount	IP (%)	Nat	64.58	Greedy Count	IP (%)	Nat	83.15
			For	57.14			For	36.11			For	37.61
		IBAN (%)	Nat	74.23		IBAN (%)	Nat	65.41		IBAN (%)	Nat	79.16
			For	45.53			For	38.88			For	25.53
		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0
			For	0			For	0			For	0
		Amount (%)	Nat	72.70		Amount (%)	Nat	54.16		Amount (%)	Nat	69.89
			For	92.85			For	80.55			For	81.25
		Feature	User	Detectors		Feature	User	Detectors		Feature	User	Detectors
Bi-weekly Update	Conservative	IP (%)	Nat	73.42	Conservative Am.	IP (%)	Nat	60.94	Conservative Count	IP (%)	Nat	77.16
			For	55.35			For	30.55			For	31.25
		IBAN (%)	Nat	74.47		IBAN (%)	Nat	66.79		IBAN (%)	Nat	77.42
			For	50.02			For	36.11			For	18.75
		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0
			For	0			For	0			For	0
		Amount (%)	Nat	65.52		Amount (%)	Nat	44.53		Amount (%)	Nat	54.06
			For	91.07			For	86.11			For	81.25
	Greedy	IP (%)	Nat	79.65	Greedy Amount	IP (%)	Nat	61.60	Greedy Count	IP (%)	Nat	81.04
			For	61.76			For	33.31			For	39.2
		IBAN (%)	Nat	77.20		IBAN (%)	Nat	67.41		IBAN (%)	Nat	80.24
			For	48.53			For	33.34			For	31.98
		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0		CC_ASN (%)	Nat	0
			For	0			For	0			For	0
		Amount (%)	Nat	75.13		Amount (%)	Nat	45.98		Amount (%)	Nat	70.16
			For	92.64			For	80.29			For	75.18

Table 7.14: Black Box Regeneration Results

7.4.1. White Box

In Table 7.12, we observe for each detector which features the attacker needs to change in order to craft an evasive fraud. Since the detectors are trained on different subsets of features, each detector shows a particular behavior. For instance, regarding the conservative strategy with a weekly update, Random Forest requires the regeneration of the IBAN only 17.14% of the total number of national frauds, while we notice a 72.47% when dealing with foreign ones. This happens because Random Forest, like the other models, gives more importance to the IBAN when evaluating foreign transactions. This concept can be also applied to the other features: foreign frauds are always more suspicious, so

the adversary needs to regenerate the features more frequently, including, if necessary, the amount. Logistic Regression model, which is the less resistant to poisoning attacks, as we stated in section 7.3.1, in the conservative strategy with bi-weekly update pushes the attacker to change the features less frequently: only the 20% of the national frauds has a regenerated IP, 2.8% has a new IBAN and 19.93% has a lower amount.

In Figure 7.9 and Figure 7.10 we report two scatter plots that emphasize the relationship between direct frauds (i.e., not regenerated) and regenerated frauds.

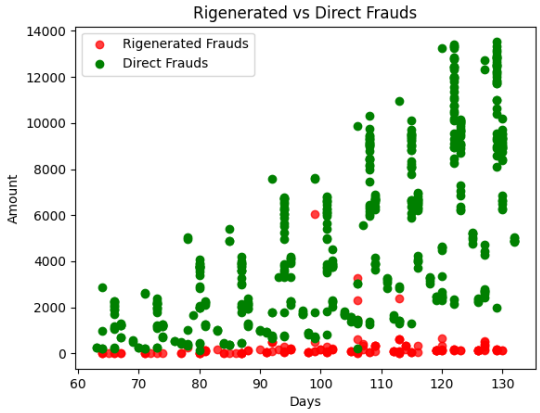

Figure 7.9: Logistic Regression, Bi-weekly Policy, Conservative Strategy

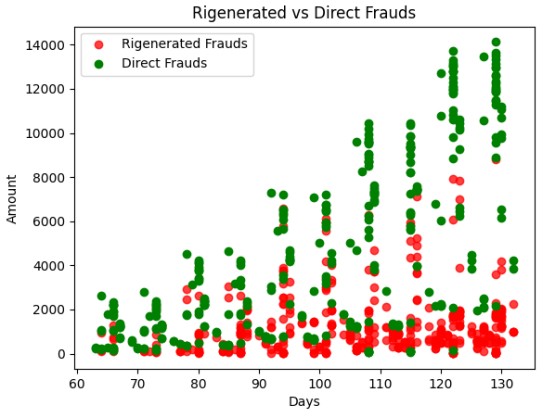

Figure 7.10: Active Learning, Bi-weekly Policy, Conservative Strategy

In Figure 7.9 we notice that in Logistic Regression, very few transactions are regenerated. Moreover, it's evident that the most frequently regenerated transactions are those that present lower amount: frauds with higher amount deceive the FDS more easily. On the other hand, as Figure 7.10 reports, an Active Learning detector is stronger and forces the attacker to regenerate the transactions.

7.4.2. Grey Box

In the Grey Box scenario, the attacker creates an Oracle trained on a similar dataset using all the features at his or her disposal. The Oracle is always the same, independently of the target system algorithm, which the attacker doesn't know. According to the results reported in Table 7.13, we can state that our Oracle suggests more often changing IP and IBAN for national frauds, while hints to regenerate the CC_ASN and the amount for foreign ones. We can notice that for each feature, the percentage of regenerated transactions is higher than that of the White Box: the reason is that our Oracle is a powerful model, which tries to filter transactions so that they could be less suspicious as

possible. In this way, an attacker can mount durable poisoning attacks, but he or she is forced to lower the amount even when that's not necessary. For instance, in a greedy approach against detectors trained according to a weekly policy, the adversary has to reduce the amount for 94.83% of national frauds and 95.85% of foreign ones.

When adopting an amount strategy, the fraudster tries to regenerate the transactions less frequently, as Table 7.13 reports, since he or she wants to consistently increase the average transactions' amount of the victim. On the contrary, in the count strategy, the Oracle suggests of change almost always the IP and the IBAN features: in some cases, such as the greedy count strategy in a weekly update context, out of 100 transactions, we change the IP more than 93 times.

In Figure 7.11, 7.12 and 7.13 we report scatter plots that provide a visual explanation of how much our Oracle regenerate the frauds for the three different strategies.

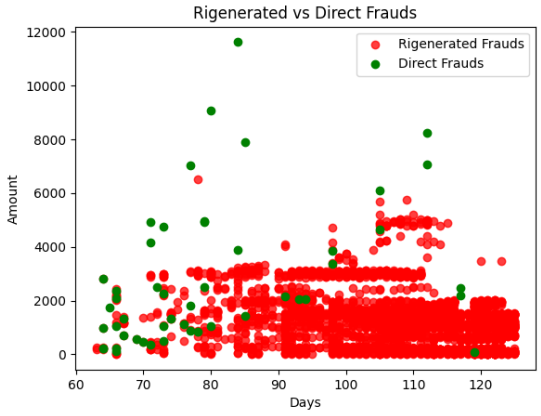

Figure 7.11: Oracle, Weekly update, Greedy strategy

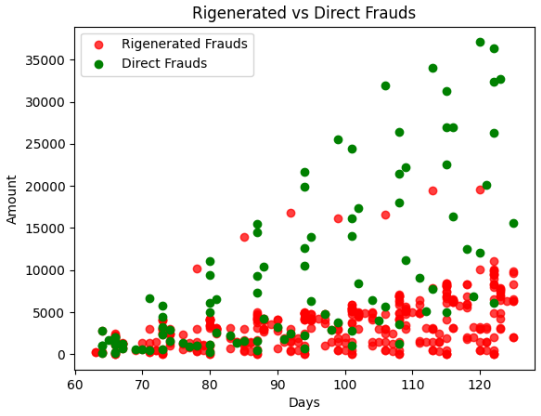

Figure 7.12: Oracle, Weekly Update, Greedy Strategy Amount

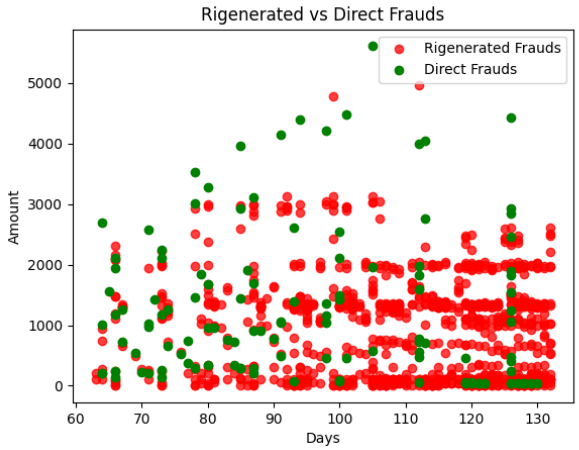

Figure 7.13: Oracle, Weekly Update, Greedy Strategy Count

We can state that in all three cases, the Oracle regenerates most of the transactions. Our Oracle is really powerful, it's difficult to deceive it: this is why we are able to get very low attack detection rates, as explained in Section 7.3.2. Regarding the amount strategy, the Oracle needs to be more flexible, because the adversary wants to poison the victim's spending pattern in a consistent way. In fact, in Figure 7.12, you can see that the regeneration task happens less frequently. In the count strategy, it's easy to notice that most of the transactions have a similar amount and they are often regenerated.

7.4.3. Black Box

In the Black Box scenario, the attacker builds the Oracle relying on just 50 features. Hence, we have an Oracle which is less powerful with respect to the Grey Box scenario. In fact, we have no features related to the Country Code, this is why the attacker never regenerates it, as Table 7.14 states. For what concerns the other features, we can apply the same reasoning of the previous Section 7.4.2.

8 | Limitations and Future Works

In this chapter, we explore the major limitations of our work and we discuss possible future works which could improve our results.

8.1. Limitations

One of the main limitations of our work is that we have dealt with not labeled datasets. Apart from some reports related to the more recent dataset, we need to create many fraudulent transactions which replicate possible attack patterns. This is an important limitation because the fraud generation process really influences our results. If we had adopted another fraud generation strategy, probably we would have ended up with very diverse results.

The most recent dataset, which is the dataset that the attacker targets, lasts only 4 months. Hence, our attacks last only 8 weeks. We have shown that there are attacks that are not detected by the FDS at all. It would be interesting to study and analyze poisoning processes that last more, in order to verify if they can be carried on for a long time.

Moreover, we were able to improve foreign transactions, because we build a reliable Oracle that in most cases allows us to achieve a zero detection rate. However, the poisoning process results really hard in some scenarios such as when we have a target Logistic Regression, which is the model that behaves best with foreign users, not allowing even a malicious transaction.

Another limitation derives directly from computational reasons. In general, the banking FDSs are powerful, but we could create more accurate systems, by using techniques that require much more computational power. In the hyperparameter tuning task, we used a random grid search, exploring just 30 combinations. Moreover, in the feature selection process, we adopt a filter approach, which slightly affects the detectors' performance. We could have exploited wrapper solutions in order to get more powerful machine learning models.

8.2. Future works

All models that we worked with are system-centric fraud detection systems, which means that they classify incoming transactions based on the whole dataset. Moreover, they are time-insensitive, and they do not keep into account time relationships between transactions. Future works could deal with user-centric FDSs, which focus on the individual user, and time-sensitive detectors, such as time series analysis models.

In addition, we show how an attacker can build a reliable Oracle by exploiting ensembling methods. It could be interesting to analyze the proposed attack approach against FDSs which apply ensembling too. In fact, as the attacker, a financial institution could use Bagging techniques in order to improve the performance of its model.

A future work could also deal with national and foreign users independently. We have seen how it's more difficult to perform foreign frauds, but we have also shown how it's possible to overcome this limitation by poisoning only the amount. An attacker could exploit a strategy to defraud national users and another one to steal money from foreign victims.

9 | Conclusions

In this dissertation, we have shown how the most popular state-of-art banking detectors behave when dealing with poisoning attacks. We impersonated an attacker and we mounted poisoning attacks against the banking detectors previously trained. More specifically, we proposed a novel approach according to which an adversary can build a very reliable oracle and manipulate smartly a specific set of transaction features. We provided results for each update policy, each strategy, and each scenario.

With our approach, we are able to steal a consistent amount of money in every scenario. In Monti's work [27], in a partial knowledge scenario the adversary was capable to steal up to 551,236€ and in a no knowledge scenario up to 394,239€, by attacking 30 victims. In this thesis, we are able to perform malicious transactions that amount to more than 4 million euros in a Grey Box attack and more than 3 in a Black Box one, by defrauding 15 customers. Moreover, our detection rates are all low, for both national and foreign users, sometimes even zero. We found out that poisoning the amount is less cautious and more effective than poisoning the count, especially for foreign users. The detection time is often very high, it goes from 30 to 60 days. On the contrary, Monti's attacks lasted on average, between two weeks and a month. Beyond the poisoning attacks results, we have deeply analyzed the feature regeneration process and we have studied which are the features that the adversary has to change more frequently at each iteration.

Bibliography

[1] BEM Research. "*Rapporto sull' E-banking, Internet Banking in Europa: Italia a -20*".

[2] IC3. "*Internet Crime Annual Report 2020*".

[3] D. Clark. "*Online banking: Fraud losses in the United Kingdom 2010-2020*".

[4] Financial Information Unit. "*Annual Report 2020*".

[5] Papernot *et al.* "*The Limitations of Deep Learning in Adversarial Settings*".

[6] Biggio *et al.* "*Poisoning Attacks against Support Vector Machines*".

[7] Krishna *et al.* "*Thieves on Sesame Street! Model Extraction of BERT-based APIs*".

[8] Weston *et al.* "*Plastic Card Fraud Detection using Peer Group analysis*".

[9] Bolton *et al.* "*Statistical Fraud Detection: A Review*".

[10] Sanchez *et al.* "*Association Rules applied to Credit Card Fraud Detection*".

[11] Vaishali. "*Fraud Detection in Credit Card by Clustering Approach*".

[12] Olszewski. "*Fraud detection using self-organizing map visualizing the user profiles*".

[13] Xuan *et al.* "*Random forest for credit card fraud detection*".

[14] Meng *et al.* "*A Case Study in Credit Fraud Detection With SMOTE and XGBoost*".

[15] Zhang *et al.* "*Customer Transaction Fraud Detection Using Xgboost Model*".

[16] Gyamfi *et al.* "*Bank Fraud Detection Using Support Vector Machine*".

[17] Asha *et al.* "*Credit card fraud detection using artificial neural network*".

[18] Yunlong Li *et al.* "*Online Transaction Detection Method Using Catboost Model*".

[19] Taha *et al.* "*An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine*".

[20] Sahin *et al.* "*Detecting credit card fraud by ANN and logistic regression*".

[21] Carminati *et al.* "*Amaretto: An Active Learning Framework for Money Laundering Detection*".

[22] Carminati *et al.* "*FraudBuster: Temporal Analysis and Detection of Advanced Financial Frauds*".

[23] Carminati *et al.* "*BankSealer: A decision support system for online banking fraud analysis and investigation*".

[24] Carminati *et al.* "*Security Evaluation of a Banking Fraud Analysis System*".

[25] HaratiNik *et al.* "*FUZZGY: A hybrid model for credit card fraud detection*".

[26] Jain *et al.* "*A Hybrid Approach for Credit Card Fraud Detection using Rough Set and Decision Tree Technique*".

[27] Monti. "*Poisoning Attacks Against Banking Fraud Detection Systems*".

[28] Carminati *et al.* "*Evasion Attacks against Banking Fraud Detection Systems*".

[29] Zeager *et al.* "*Adversarial learning in credit card fraud detection*".

[30] Nocedal. "*Updating quasi-Newton matrices with limited storage*".

[31] Sayge *et al.* "*Evasion Attacks with Adversarial Deep Learning Against Power System State Estimation*".

[32] Goodfellow *et al.* "*Explaining and Harnessing Adversarial Examples* ".

[33] Wiyatno *et al.* "*Maximal Jacobian-based Saliency Map Attack*".

[34] Combey *et al.* "*Probabilistic Jacobian-Based Saliency Maps Attacks* ".

[35] Wiyatno *et al.* "*Maximal jacobian-based saliency map attack*".

[36] Biggio *et al.* "*Security Evaluation of Pattern Classifiers under Attack*".

[37] Biggio *et al.* "*Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning*".

[38] Ambra Demontis *et al.* "*Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks*".

[39] Suciu *et al.* "*When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks* ".

[40] Le Borgne *et al.* "*https://fraud-detection-handbook.github.io*".

[41] Jha *et al.* "*A Descriptive Study of Credit Card Fraud Pattern*".

[42] European Central Bank. "*6th report on card fraud*".

[43] Nadim *et al.* "*Analysis of Machine Learning Techniques for Credit Card Fraud Detection*".

[44] Bhattacharyya *et al.* "*Data mining for credit card fraud: A comparative study*".

[45] Ministero dell'Economia e della Finanza. "*Rapporto statistico sulle frodi con le carte di pagamento 2021*".

[46] Ho, Tin Kam. "*Random Decision Forests*".

[47] Breiman. "*Random Forests*".

[48] Chen *et al.* "*Story and Lessons behind the evolution of XGBoost*".

[49] "*https://github.com/microsoft/LightGBM*".

[50] Brownlee *et al.* "*Gradient Boosting with Scikit-Learn, XGBoost, LightGBM, and CatBoost*".

[51] "*https://github.com/catboost/catboost*".

[52] "*https://catboost.ai/*".

[53] Vapnik *et al.* "*Support-Vector Networks*".

[54] McCulloch *et al.* "*A Logical Calculus of Ideas Immanent in Nervous Activity*".

[55] Ivakhnenko *et al.* "*Cybernetic Predicting Devices*".

[56] Cramer. "*The Origins of Logistic Regression*".

[57] Bergstra *et al.* "*Random Search for Hyper-Parameter Optimization*".

[58] Snoek *et al.* "*Practical Bayesian Optimization of Machine Learning Algorithms*".

[59] Larsen *et al.* "*Design and regularization of neural networks: the optimal use of a validation set*".

[60] Chapel *et al.* "*Choosing multiple parameters for support vector machines*".

[61] Chuong *et al.* "*Efficient multiple hyperparameter learning for log-linear models*".

[62] Jiang *et al.* "*A New Oversampling Method Based on the Classification Contribution Degree*".

[63] Mckee. "*Three Digital Commerce Growth Opportunities*".

[64] James *et al.* "*An introduction to statistical learning*".

[65] Guyon *et al.* "*An Introduction to Variable and Feature Selection*".

[66] Opitz *et al.* "*Popular Ensemble Methods: An Empirical Study*".

[67] Kuncheva *et al.* "*Measures of diversity in classifier ensembles, Machine Learning*".

[68] Kearns *et al.* "*Thoughts on Hypothesis Boosting*".

[69] Zareapoor *et al.* "*Application of Credit Card Fraud Detection: Based on Bagging Ensemble Classifier*".

[70] Randhawa *et al.* "*Credit Card Fraud Detection Using AdaBoost and Majority Voting*".

[71] Jagielski *et al.* "*Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning*".

[72] Mozaffari-kermani *et al.* "*Systematic Poisoning Attacks on and Defenses for Machine Learning in Healthcare* ".

[73] Chen *et al.* "*Automated Poisoning Attacks and Defenses in Malware Detection Systems: An Adversarial Machine Learning Approach*".

[74] Brownlee. "*Stacking Ensemble Machine Learning With Python*".

[75] Soleymanzadeh *et al.* "*Cyberattack and Fraud Detection Using Ensemble Stacking* ".

[76] Brownlee *et al.* "*How to Develop Voting Ensembles With Python*".

[77] Chicco *et al.* "*The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation*".

List of Figures

1.1 Transaction Flow 7
1.2 Trained SVM for a Classification Problem 10
1.3 Compromised SVM after Injecting a Poison Sample 11

3.1 Mean Transaction Amount Per Day 21
3.2 Count Per Day 21
3.3 Mean Amount per Weekday 22
3.4 Count per Weekday 22
3.5 Mean Amount per Hour 23
3.6 Count per Hour 23

4.1 Training Process Flow 40
4.2 Confusion Matrix, Active Learning, Weekly Update 43
4.3 AUC-ROC, Active Learning, Weekly Update 43
4.4 AUC-PRC, Active Learning, Weekly Update 43

5.1 Attack Flow Graph 47
5.2 Regeneration Flow 58

6.1 System UML Diagram 60
6.2 UML Sequence Diagram 61

7.1 CatBoost, Weekly Policy, Conservative Strategy 80
7.2 CatBoost, Bi-weekly Policy, Conservative Strategy 80
7.3 XGBoost, Bi-weekly Policy, Greedy Strategy 82
7.4 XGBoost, Bi-weekly Policy, Greedy Strategy Amount 82
7.5 XGBoost, Bi-weekly Policy, Greedy Strategy Count 82
7.6 LightGB, Weekly Policy, Conservative Strategy 84
7.7 LightGB, Weekly Policy, Conservative Strategy Amount 84
7.8 LightGB, Weekly Policy, Conservative Strategy Count 85
7.9 Logistic Regression, Bi-weekly Policy, Conservative Strategy 89

7.10 Active Learning, Bi-weekly Policy, Conservative Strategy 89
7.11 Oracle, Weekly update, Greedy strategy 90
7.12 Oracle, Weekly Update, Greedy Strategy Amount 90
7.13 Oracle, Weekly Update, Greedy Strategy Count 91

List of Tables

3.1 General Information about the Datasets 20
3.2 Users' Category for 2014-15 Dataset 24
3.3 Features Values for Frauds Crafting 25
3.4 Generated Frauds 26
3.5 Generation Frauds Results 27

4.1 Features Selected 36
4.2 Hyperparameters Selected 38
4.3 Fraud Detection Systems Metrics, Weekly Update 41

5.1 Strategy Parameters 49
5.2 Victims Selection 51

7.1 Bagging and Majority Voting Oracle Performance 69
7.2 AdaBoost and Majority Voting Oracle Performance 70
7.3 Stacking Oracle Performance 70
7.4 Bagging Oracle Performance 71
7.5 White Box Attacks 72
7.6 Grey Box Attacks: Poisoning Both 73
7.7 Grey Box attacks: Poisoning Amount 74
7.8 Grey Box Attacks: Poisoning Count 75
7.9 Black Box Attacks: Poisoning Both 76
7.10 Black Box Attacks: Poisoning Amount 77
7.11 Black Box Attacks: Poisoning Count 78
7.12 White Box Regeneration Results 86
7.13 Grey Box Regeneration Results 87
7.14 Black Box Regeneration Results 88

List of Algorithms

5.1 Timestamp Selection Algorithm 53
5.2 Amount Selection Algorithm 54
5.3 Count Selection Algorithm 55
5.4 IPs Selection Algorithm 56