Fraud Detection Under Siege: Practical Poisoning Attacks and Defense Strategies

TOMMASO PALADINI, FRANCESCO MONTI, MARIO POLINO, MICHELE CARMINATI, and STEFANO ZANERO, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy

Machine learning (ML) models are vulnerable to adversarial machine learning (AML) attacks. Unlike other contexts, the fraud detection domain is characterized by inherent challenges that make conventional approaches hardly applicable. In this paper, we extend the application of AML techniques to the fraud detection task by studying poisoning attacks and their possible countermeasures. First, we present a novel approach for performing poisoning attacks that overcomes the fraud detection domain-specific constraints. It generates fraudulent candidate transactions and tests them against a machine learning-based *Oracle*, which simulates the target fraud detection system aiming at evading it. Misclassified fraudulent candidate transactions are then integrated into the target detection system's training set, poisoning its model and shifting its decision boundary. Second, we propose a novel approach that extends the adversarial training technique to mitigate AML attacks: during the training phase of the detection system, we generate artificial frauds by modifying random original legitimate transactions; then, we include them in the training set with the correct label. By doing so, we instruct our model to recognize evasive transactions before an attack occurs. Using two real bank datasets, we evaluate the security of several state-of-the-art fraud detection systems by deploying our poisoning attack with different degrees of attacker's knowledge and attacking strategies. The experimental results show that our attack works even when the attacker has minimal knowledge of the target system. Then, we demonstrate that the proposed countermeasure can mitigate adversarial attacks by reducing the stolen amount of money up to 100%.

CCS Concepts: • **Applied computing → Online banking**; • **Computing methodologies → Machine learning**; • **Security and privacy → Software and application security**.

Additional Key Words and Phrases: adversarial machine learning, poisoning attacks, fraud detection systems, adversarial training

1 INTRODUCTION

Machine learning (ML) techniques are employed in multiple fields, such as speech recognition [22], computer vision [35], natural language processing [40], and anomaly detection [49]. Nowadays, machine learning solutions are also widely used to detect banking frauds [4, 7, 14, 15, 18, 37, 47, 61]. Unfortunately, machine learning models are vulnerable to adversarial machine learning (AML) attacks [6, 54]. For example, an attacker may craft "adversarial examples", i.e., malicious perturbated input with additional non-random noise that the machine learning model confidently misclassifies. Such examples can be exploited to systematically evade the classifier (*evasion attacks*) and if included in the training set, may degrade the performance of the learned model (*poisoning*

Authors' address: Tommaso Paladini, tommaso.paladini@polimi.it; Francesco Monti, francesco4.monti@mail.polimi.it; Mario Polino, mario.polino@polimi.it; Michele Carminati, michele.carminati@polimi.it; Stefano Zanero, stefano.zanero@polimi.it, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Via Giuseppe Ponzio 34/5, 20133, Milano (MI), Italy.

2471-2566/2023/8-ART $15.00
https://doi.org/10.1145/3613244

attacks) [12]. Interestingly, adversarial examples crafted against a model are usually effective against similar models. This phenomenon, referred to as the "transferability property" [21, 51], suggests that a malicious agent can craft adversarial examples to conduct an attack without a comprehensive knowledge of the target system. The good news is that the success of adversarial examples can be reduced via defense techniques such as adversarial training [26], anomaly detection [44], and feature distillation [39], or through statistical tests [27]. Only a small fraction of research work demonstrates the feasibility of AML attacks and defenses in the fraud detection domain. Among them, Carminati et al. [17] show how a determined attacker, with limited resources and standard machine learning algorithms, can build a surrogate of the original fraud detection system (FDS) and use it to evaluate carefully crafted frauds before submitting them on behalf of their victims. Other approaches solve an optimization problem to generate transactions that mimic the victim behavior [16] or adapt existing solutions used in the image recognition domain [19]. Since an attacker can only indirectly interact with the target model, traditional approaches are hardly applicable to this context. The same problem also affects the proposed countermeasures. Existing mitigation strategies in computer vision require assumptions that do not hold for the instances of AML attacks against FDSs. In particular, adversarial training may not generalize to our problem: fraudulent transactions that evade the classifier may not resemble adversarial examples because they are not obtained by directly adding noise in feature space. Other mitigations for poisoning attacks involve outlier detection mechanisms [44, 50]. To directly identify poison samples in the training set before the training phase of the ML model happens, they apply label sanitization algorithms on the training dataset [45] and robust learning algorithms that detect input samples that degrade the performance of the model [30]. In the attacks against FDSs, malicious transactions are not outliers with respect to legitimate transactions, but *inliers* crafted to mimic the original behavior of the victim. Furthermore, the attacker cannot tamper with the training dataset of the FDS since the labels of the transactions are directly assigned by the financial institution. This demonstrates the importance of deepening the research in the fraud detection field from both the point of view of attacking and defensive strategies.

In this paper, we study the application of poisoning attacks and their possible countermeasures in the financial fraud detection context. We propose ① a novel approach to craft poisoning samples that adapts and expands existing solutions, overcoming the challenges of the domain under analysis; ② a novel defense strategy in the form of an *adversarial data augmentation* scheme, directly inspired by adversarial training [26], for reducing the probability of successful evasion of the FDS classifier.

The proposed AML attack is based on a hybrid combination of machine learning techniques and parametric decision rules, working together to generate and validate evasion and poisoning samples, which are iteratively refined to reduce their suspiciousness. First, we generate candidate adversarial transactions exploiting parametric decision rules (i.e., heuristics), which consider the specific constraints of the fraud detection domain. Then, we use a machine learning-based *Oracle*, which simulates the target fraud detection system, to validate the generated frauds. After that, we test the frauds validated by the Oracle against the detection systems under attack, aiming at evading it. If the target fraud detection system flags the adversarial examples as legitimate, it will integrate them into its training set, poisoning its model and shifting its decision boundary in favor of the attacker – e.g., by progressively *increasing the stolen amount of funds* during the poisoning attack. We model an attacker with different degrees of knowledge of the target system: perfect knowledge (*white-box*), partial knowledge (*gray-box*), and no knowledge (*black-box*). We also design different attack strategies, which define the attacker's behavior, i.e., the number, the amount of money, the nationality of frauds injected, and the poisoning process speed. Finally, we study different update policies – i.e., how often the models are retrained to include new data.

The proposed defense strategy aims at countering adversarial attackers by training the target detection system to recognize their stealthy patterns. First, we simulate adversarial attacks by generating artificial frauds that mimic the user's behavior through simple heuristics. Then, we use the FDS to classify them, similarly to what the attacker does with the Oracle. Finally, we add the misclassified samples in the training dataset and retrain the FDS model. By adopting such a strategy, we show examples of adversarial fraud to the model before the attack

occurs, anticipating the attacker's behavior. In particular, to replicate the adversarial frauds submitted by the attacker, we modify random original legitimate transactions based on the prior knowledge of the input features that the attacker can directly control while committing fraud.

Using two real anonymized bank datasets with only legitimate transactions augmented with synthetic frauds, we show how a malicious attacker can compromise state-of-the-art fraud detection systems by deploying adversarial attacks. Our attack – with no mitigation – achieves different outcomes depending on the experimental settings. Unsurprisingly, our attacks go unnoticed in the white-box scenario, and the attacker steals up to a million euros from thirty pseudo-randomly selected victims. In the more realistic gray-box and black-box scenarios, the attacks are detected between 55% and 91% of the time. However, the attacks last, on average, enough time – up to a couple of months – to steal large amounts of money from victims' funds (on average between € 41,995 and € 388,342). Our defense strategy mitigates such attacks by detecting most adversarial frauds placed by the attacker at their first evasion attempt. When the attacker has limited knowledge of the system, we can sometimes detect all the frauds and completely stop the attack. In worse cases, we can still detect a sufficient number of frauds and decrease the average stolen amount from 31.89% to 95.55%. In the full knowledge scenario, where the attacker is aware of all the system details and its mitigation, our countermeasure still reduces the overall stolen amount between 18.12% and 84.26% with respect to the base configuration of the system. The only exception is the active learning (AL) based model, a variant of [57] and [37], for which we record lower improvements and even some cases of deterioration. In brief, our results show that we thwart the attempts of the attacker to find new evasive transactions and, therefore, force them to search for more complex strategies to evade classification. In addition, we evaluate the performance trade-off between the detection of adversarial transactions and the increase of the false positive rate, varying the number of artificial frauds injected during training with our countermeasure. We observe a collateral increase of false positives, between 13.37% and 82.72%, depending on the ML model used by the FDS. Finally, we compare the proposed defense technique with state-of-the-art mitigations based on anomaly detection [44] and adversarial training [26]. Our solution outperforms such mitigations that achieve no significant improvement against adversarial attacks and, in some cases, even scarcely reduce the system's performance.

The main contributions of this paper are:

- A novel approach based on a hybrid combination of machine learning techniques and parametric decision rules to perform poisoning attacks against fraud detection methods under different degrees of attacker's knowledge. To the best of our knowledge, this is the first work about poisoning attacks in the fraud detection context. We evaluate our attack against state-of-the-art fraud detection systems simulating different fraudsters' strategies.
- A novel *plug-and-play* and *non-intrusive* adversarial-training-based approach to mitigate the adversarial machine learning attacks in the fraud detection domain.

The remaining of the paper is structured as follows: in Section 2, we review the theoretical background required to understand the problem statement, then we present the given problem, the state-of-the-art solutions, and the challenges that we must overcome. In Section 3, we report the threat model of the poisoning attack. In Section 4, we provide an overview of the banking dataset used for our experimental validation, while in Section 7.2, we describe the fraud detection systems under attack. In Section 5 and Section 6, we respectively describe in detail our attack and mitigation approaches. In Section 7, we describe the experimental validation process and discuss the obtained results. In Section 8, we describe the main limitations of our approach and give possible directions for future research work. Finally, in Section 9, we summarize the results of our work and draw conclusions.

2 BACKGROUND AND RELATED WORKS

Along with electronic payments, Internet banking fraud keeps increasing in terms of volume and value by each year, resulting in considerable financial losses for institutions and their customers [33]. Among the different

typologies of fraud, banks consider large-scale *cyberattacks* as the most dangerous threat [33]. Financial institutions face well-organized – and sometimes even state-backed – cybercriminal groups, who are responsible for digital heists [53]. Malicious actors also foster a *virtual* underground economy [25] on the dark web, where they sell malware tools and customers' private information. Fraud is a costly phenomenon for financial institutions, which estimate to recover only less than 25% of the economic losses, leading to the conclusion that fraud prevention is essential [33]. Typical schemes of Internet banking fraud are *information stealing* and *transaction hijacking* [16]. In information stealing, the fraudster steals the credentials and other relevant information from its victim, like a one-time password (OTP) code, to connect to their account and perform fraudulent transactions. In this case, the connection is established on the attacker's device. With the transaction hijacking scheme, the attacker takes over legitimate transactions made by the victim and redirects them to controlled bank accounts. This scheme is more challenging to identify because of the connection originating on the victim's device. Common means that fraudsters exploit for Internet banking fraud are *phishing*, the practice of deceiving a victim by presenting them with a highly credible fraudulent website, usually as an alternate version of a legitimate one [23]; *banking trojans*, a class of malware purposely designed to steal credentials and financial information from infected devices; *social engineering*, the practice of manipulating individuals into divulging sensitive information to a malicious actor [36].

2.1 Banking Fraud Detection Systems

As manual inspections of the entire flow of banking transactions by human experts may require an unreasonable amount of resources, researchers and experts in the cybersecurity field have produced automated tools, fraud detection systems (FDSs). Such tools efficiently distinguish fraudulent behavior from legitimate one based on historical data. Fraud detection is essential for a variety of domains such as banking fraud [2, 14, 15, 20], credit card transactions [31, 43], and e-Commerce fraud [41]. Research in banking fraud detection is restrained by the lack of publicly available datasets and privacy-related issues [15].

In literature, there exist examples of FDS solutions based on supervised ML algorithms, such as neural networks (NNs) [7, 13, 43], support vector machines (SVMs) [8, 34, 48], random forest (RF) [4, 8, 34, 58, 59], logistic regression (LR) [4, 29, 58], Extreme Gradient Boosting (XGBoost) [61], hierarchical attention mechanisms [2]. There are also examples of unsupervised solutions [41] and works that adopt *hybrid approaches*, combining different techniques to provide multiple perspectives on the same problem [13, 15, 37, 57]. In the last years, also *pseudo-recommender system problem* have been proposed to solve the banking fraud detection problem. We refer the reader to [1, 3] for further details of fraud detection systems functioning.

2.2 AML Attacks against Fraud Detection Systems

Banking fraud detection systems, like other ML-based solutions, are vulnerable to adversarial machine learning attacks. AML is an emerging field that explores machine learning under the condition that an adversary tries to disrupt the correct functioning of a learning algorithm [28]. The goal of AML is to produce robust learning models *i.e.*, models capable of resisting the opponent. Popular kinds of AML attacks are *evasion* and *poisoning attacks*. In evasion attacks, smart attackers craft examples that evade classification at inference time [9]. In poisoning attacks, attackers craft or modify examples in the training dataset, causing the ML algorithms to learn poor-performing models [11, 12]. The success of these approaches has been mostly proved in the domain of image classification [26, 55]. However, in the context of fraud detection, such approaches for adversarial attacks hardly adapt since samples are manipulated and evaluated with *aggregated features* and not with their direct features. A possible attacker can only submit *raw* transactions to the banking system, where the only input is represented by a few attributes, e.g., the amount, the recipient's IBAN, and the time instant of the operation execution [17]. The aggregated features on which the transactions are evaluated by the FDS are computed using

historical data of the banking customers. Therefore, as directly operating on aggregated examples may lead to unfeasible sequences of transactions, the attacker has to find the sequences of raw adversarial transactions that, once aggregated, resemble the desired aggregated adversarial example.

In the fraud detection context, AML attacks pose a direct threat in terms of economic damage to financial institutions and their customers. Moreover, in literature, few research works demonstrate AML attacks against FDSs [16, 17, 19].

These systems are vulnerable to *mimicry attacks* [16], in which the attacker disguises their frauds as legitimate transactions to avoid alerting the detector. The attacker infects their victims' devices through different means (see Section 2), observes their spending patterns, and uses such useful information to *mimic* them (*i.e.*, reproduce their behavior). Meanwhile, the attacker will try to maximize their illegitimate profit while remaining undetected. This attack can be formulated as an optimization problem [16].

FDSs are also vulnerable to evasion attacks, as shown by Carminati et al. [17]. Their approach works as follows: the attacker, depending on their knowledge of the target FDS in terms of features, dataset, learning algorithm, parameters/hyperparameters, and past data of their victims, builds a surrogate of the real FDS, called Oracle. After observing their victims' behavior, the attacker crafts stealthy frauds to be submitted on their victims' behalf by choosing the least suspicious timestamp and amount. At last, the attacker aggregates such transactions with the victims' data at their disposal, classifies them using their Oracle, and sends only the fraudulent transfers classified as legitimate. The described attack results in an integrity violation.

Cartella et al. [19] adapt state-of-the-art adversarial attacks to tabular data with custom constraints on non-editable input variables of transactions, decision thresholds, and loss function. The authors also address the *imperceptibility* of the adversarial samples, i.e., the similarity of the generated adversarial transactions to regular ones to the eye of a human operator, by evaluating their distance from original samples on a custom norm.

2.3 AML Mitigations

In the last few years, various mitigations have been proposed against AML attacks. Given the scope of this work, we focus on the defense mechanisms against poisoning attacks. We refer the reader to [5, 38, 60] for a detailed overview of mitigations against evasion attacks.

The setting of poisoning attack mitigations can be modeled as a game between two players [50]: the attacker and the defender. The former wants the ML algorithm to learn a *bad model*, and the latter tries to learn the correct one. Training on a poisoned dataset means that the defender has failed. Defenses for poisoning can be *fixed* if they do not depend on the generated poisonous data or *data-dependent* otherwise. In practice, unless the defender knows the true distribution that generates poisonous data, fixed defenses are unfeasible. In general, such defenses require strong assumptions on the attacker's goal and the procedure that generates the poisoning samples [24, 30, 44, 45]. The approach proposed by Paudice et al. [44] consists in building a distance-based anomaly detector to find poisoning data using only a small fraction of trusted data points, the *trusted dataset.*

Their solution cannot be applied to our approach since one of its main assumptions does not hold against our attack: adversarial transactions are not outliers with respect to the users' regular behaviors [17]. Other approaches [45] work with the assumption that the attacker can directly control the labels in the training dataset of the FDS, which also does not hold in our case. Jagielski et al. [30] propose an algorithm for regression with high robustness against a wide class of poisoning attacks, in particular, poisoning attacks formalized as a bi-level optimization problem with gradient ascent. The algorithm removes points with large residuals, focusing on *inliers*, poisoned points with a similar distribution of the training dataset. Adversarial transactions are inliers that are meant to degrade the performances of the model only for a particular class of users, i.e., the victims of the attack. Furthermore, the attacker's poisoning process may be so slow that observing a significant performance reduction would require collecting a large number of adversarial transactions over a long period of time. The economic

damage posed by fraud requires our approach to reject them as soon as possible to avoid losing large amounts of capital to the fraudster. Consequently, the approach proposed by Jagielski et al. [30] may not be directly applied to this research work.

Adversarial training, introduced by Goodfellow et al. [26], is a mitigation approach that reduces the success of the evasion of adversarial examples against deep convolutional neural networks. The principle of adversarial training is to teach the model how to recognize adversarial examples by encoding a procedure that generates adversarial examples, such as the Fast Gradient Sign Method (FGSM), within the training algorithm. The authors achieve this goal by including an adversarial regularization term in the loss function of the ML model, as follows:

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon \text{sign}(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y))) \tag{1}$$

Geiping et al. [24] extend such a framework and propose a generic approach for defending against training-time poisoning attacks. During each iteration of batch gradient descent, the drafted batch of samples is split into two sets with probability p. Then, on one of the sets, a data poisoning attack is applied until its samples are reclassified with the desired label. Last, the batches are merged into a single one, the poisoned samples have the correct label, and the model is regularly trained. The proposed approach proves effective in image classification, but there is no direct solution for applying such a strategy to our domain and to models that cannot be optimized with gradient descent. Furthermore, Bai et al. [5] examine the generalization capability of adversarial training under three perspectives: standard generalization, adversarially robust generalization, and generalization on unseen attacks. In general, with respect to the first two properties, the authors argue that adversarial training falls short for its performance trade-off with regular examples and for the tendency of adversarially trained models to *overfit* on perturbed examples of the training set. However, the last property is the one that interests us the most: adversarial training generalizes poorly to new, unseen attacks. This suggests that the attacks developed so far do not represent the space of all the possible perturbations and, for this limitation, a model trained with adversarial regularization should be able to solve the evasion of examples generated with the Fast Gradient Sign Method (FGSM), but not to recognize the adversarial transactions generated according to our attack approach.

3 THREAT MODEL

To properly define the scope of our work, we identify the threat model according to a framework commonly used in literature [6, 10, 12, 21]. This framework defines the attacker's goals, their knowledge of the system under attack, and their influence over the input data *i.e.*, which manipulations are allowed.

3.1 Attacker's Goal

AML attacks may violate different security properties of the system, manipulate a different set of samples, and have different influences on the target algorithm. An attacker may bypass the defending system, thus gaining access to the services or the resources protected (*integrity violation*), or compromise the system functionalities for everyone, including legitimate users (*availability violation*), or retrieve confidential information from the learning algorithm, such as user personal data (*privacy violation*). The attack specificity property refers to which samples the attacker wants the model to misclassify. They may either misclassify a small set of selected samples (*targeted attack*) or find and then exploit misclassified samples starting from any possible set (*indiscriminate attack*). The influence refers to the impact the attacker has on the classifier itself. In a *causative* attack, the attacker interacts with the training set. They directly influence the learning algorithm, by changing its decision boundaries. Instead, in an *exploratory* attack, the attacker only interacts with the test set. They use their knowledge of the target classifier (already trained) to craft samples that are misclassified at test time. The error specificity refers to what kind of misclassification the attacker is interested in and is relevant only in multi-class scenarios. The objective

can either be to classify the sample to any class different from the true one (*generic* misclassification) or to classify the sample as a specific target class, different from the true one (*specific* misclassification).

In our work, the threat model is defined as follows: the attacker performs a causative poisoning attack, which is an integrity violation. The attack is targeted against some specific users (i.e., the ones that had their credit card information stolen and/or have installed a Trojan), but usually, the information-gathering process is generic (e.g., whoever falls for a phishing campaign). The classification is binary: the error specificity corresponds to performing fraudulent transactions on behalf of a legitimate user and having them accepted as legitimate.

3.2 Attacker's Knowledge

We model the attacker's knowledge of the target FDS as a tuple $\theta = (F, A, w, D, U, P)$, where F represents the knowledge of the machine learning model features, A represents the knowledge of the particular learning algorithm, w represents the knowledge of the parameters/hyperparameters of the model, D represents the knowledge of the model training set, U represents the knowledge of the past transactions of the victims, P represents the knowledge of the *update policy* of the FDS, the time interval between each new training phase of the system. The poisoning attack unfolds in different scenarios depending on the attacker's knowledge of the system. The possible attack scenarios are the following: *white-box*, *gray-box*, *black-box*. We identify the type of knowledge of one of the variables with the following symbols: x refers to full knowledge, $\tilde{x}$ to partial knowledge, $\hat{x}$ to no knowledge.

Black-box. The black-box scenario represents the best case for the defender, the one in which the attacker has zero knowledge of the target system. They only have at their disposal some of the past victims' transactions in order to compute aggregated features. The tuple describing this scenario is $\theta_{bb} = (\hat{F}, \hat{A}, \hat{w}, \hat{D}, \tilde{U}, \hat{P})$.

Gray-box. In this scenario, the attacker has only partial knowledge of the target system. They know the features used by the learning algorithm, its training update policy, and some past data of the victims but have no knowledge of the actual algorithm, parameters/hyperparameters, and training dataset. The tuple describing this scenario is $\theta_{gb} = (F, \hat{A}, \hat{w}, \hat{D}, \tilde{U}, P)$.

White-box. The white-box scenario, instead, represents the worst case for the defender: the attacker has full knowledge of the target system. This scenario is hardly achievable in reality, considering the vastness of security measures applied by banks but provides an overview of the worst-case economic damages. The tuple describing this scenario is $\theta_{wb} = (F, A, w, D, U, P)$.

3.3 Attacker's Capability

The constraints on the attacker's actions and data manipulation within the target system's data processing pipeline are defined by the attacker's capabilities. In the context of banking fraud detection, the attacker can observe the banking movements of their victims and submit transactions on their behalf or hijack legitimate ones toward a controlled account. These assumptions are realistic if we consider the possible technological means at their disposal. As for the data manipulation side, the attacker can place examples in the FDS training dataset, but they cannot control all of the input and aggregated features. Aggregated features of the transactions are out of the attacker's control: the FDS will automatically aggregate the transactions with past user data. The attacker can place an arbitrary number of transactions against a victim and for each of them, they control the input amount and timestamp, the time instant of the banking transfer execution. The attacker has also no *label influence*, meaning that they cannot control the labels of their fraudulent transactions in the FDS dataset, which are instead assigned by the system itself.

4 DATASET ANALYSIS

We use three datasets containing real bank transfers from an Italian bank: `DO2012_13`, `DO2014_15a`, `DA2014_15b`. As their names suggest, each dataset contains transactions recorded in a specific time interval. Datasets `DO2012_13` and `DO2014_15a` have been manually inspected by domain experts and contain only legitimate transfers. `DA2014_15b` contains real examples of frauds. The datasets respectively contain 583,920, 470,950, and 627 banking transfers (i.e., the "transactions") and 53,823, 58,504, and 97 unique customers. More details are provided in Section 7.1. In all of the datasets, users' sensitive personal data have been replaced with their hashed values, in order to respect privacy while preserving valuable information. The attributes that appear in our datasets are: `Amount` (amount of the bank transfer in EUR), `UserID` (unique identifier of the customer), `Timestamp` (date and time of the execution of the bank transfer), `IP` (IP address of the client that requested the transaction), `IBAN` (IBAN code associated to the beneficiary account), `IBAN_CC` (country code (CC) of the transfer beneficiary IBAN), `ASN_CC` (tuple of CC and autonomous system number (ASN) of the connection), and `SessionID` (unique identifier of the online banking session).

5 POISONING ATTACK APPROACH

Our attack approach works under the following assumptions:

- [A1] **Automatic and periodic FDS re-training.** The ML model of the FDS is periodically updated by re-training on new incoming transactions, according to specific update policies. This assumption represents the expected behavior of state-of-the-art FDSs.
- [A2] **Attacker dataset.** The attacker possesses a banking dataset with real bank transfers, which allows them to train their Oracle. This assumption is difficult to satisfy because banks do not publicly share their data.
- [A3] **Victim control.** The attacker has observed past transactions of their victims, they are capable of observing victims' transactions during the attack and can submit transactions on their behalf. The attacker can realistically achieve it using financial malware and it is necessary to build an approximation of the user's behaviors.
- [A4] **Attack detection.** The attack is detected if the FDS classifies as a fraud in one of the transactions submitted by the attacker.
- [A5] **Victim protection.** A victim is protected only if fraud against him is detected by the FDS. This assumption is realistic because usually bank operators manually review only the transactions flagged by the FDS.

5.1 Overview

As shown in Figures 1 and 2, our attack approach consists of two phases: evasion phase and poisoning phase. During the evasion phase, the attacker makes their first attempt at evading the FDS classifier against the chosen victims. The poisoning phase starts after the evasion phase and spans the entire duration of the attack. In this phase, the attacker keeps committing frauds with increasingly higher value over time after the victims are successfully defrauded during the first phase. From this point on, having in mind that the fraudulent transactions crafted by an attacker can be functionally both evasion and poisoning examples for the target FDS, we refer to them as *adversarial transactions*. In fact, such transactions are evasion examples since their goal is to be misclassified by the target system, and poisoning examples since, if misclassified, they are placed in the training dataset with the wrong label, causing the classifier to learn an altered concept. We also divide adversarial transactions into two categories depending on which phase the attacker crafts and injects them: *evasion phase transactions (EPTs)* and *poisoning phase transactions (PPTs)*. On an operational level, we subdivide both phases into three consecutive steps: Snooping, Crafting, and Injection. As shown in fig. 2, the attacker repeatedly follows such steps: the first iteration represents the evasion phase, while the following ones represent the poisoning phase. The first step, Snooping, consists of retrieving the victims' past transactions. The attacker furtively collects and gathers

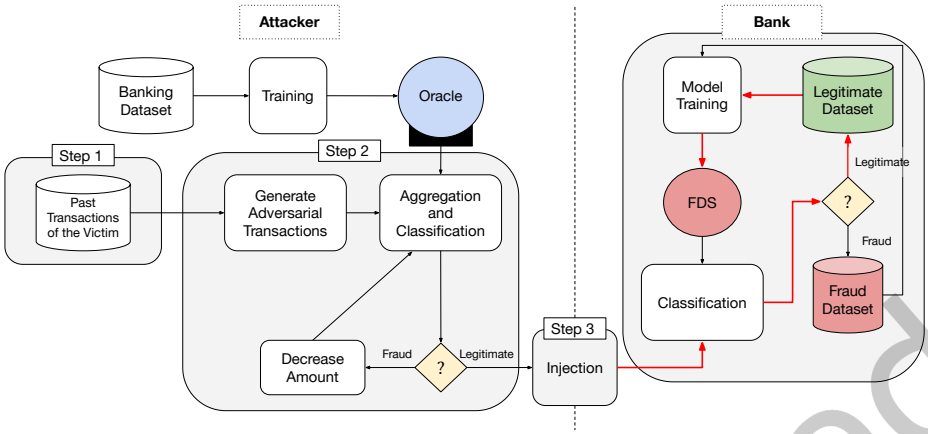


Fig. 1. Overview of our poisoning attack approach.

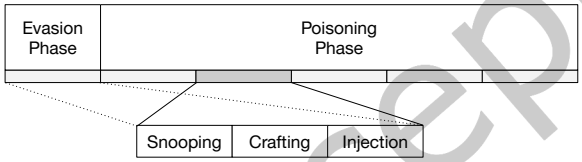


Fig. 2. Attack phases and inner steps.

information about the victims to understand their behavior. During the Crafting step, the attacker generates their adversarial transactions relying on domain-based decision rules (heuristics) and the victims' spending patterns collected so far. The attacker checks the crafted adversarial transactions with the Oracle, a surrogate of the real ML-based detector. The Oracle predicts the suspiciousness of each transaction by outputting a confidence score, a scalar value between 0 (legitimate) and 1 (fraud). Then, they select the frauds with confidence scores smaller than a predefined threshold, which depends on their knowledge of the target system. When the Oracle rejects a fraud, the attacker regenerates it by lowering its value. Decreasing the amount of a fraudulent transaction reduces its suspiciousness, as frauds generally tend to have higher monetary value than legitimate transactions [42]. This loop continues until either the Oracle accepts the transaction or the amount becomes smaller than a fixed value. In the latter case, the attacker discards the transaction because it no longer brings significant profit. During the last step, Injection, the attacker commits the frauds filtered by the Oracle on behalf of their victims by different technical means at their disposal. The target FDS will analyze such transactions by aggregating them with historical user data and ultimately classify the resulting example. If the frauds deceive the FDS, they are successfully injected into the banking database as legitimate examples. Otherwise, the FDS will discard them, and the user enters a state of protection. In this case, the attacker can no longer defraud the victim, as their account is on hold until further investigation by the financial institution. Let us recall that the attacker's Oracle is not a perfect replica of the target FDS. This step ends with the attacker passively waiting for the FDS to re-train on the poisoned training dataset accordingly to its update policy. As a result, the FDS will possibly learn a model altered by the injected adversarial transactions. With successful injection attempts, the attacker can iterate the whole process over time, exploiting the poisoned FDS and crafting frauds with higher amounts.

5.2 Oracle

The attacker strives to replicate the FDS under attack with their Oracle. Their similarity depends on the attacker's knowledge of the target system. In the white-box scenario, the attacker can build an Oracle that fully replicates the FDS. However, in limited knowledge scenarios (black-box and gray-box), the attacker has only incomplete information available. Under these knowledge conditions, the attacker can only build a surrogate model. They do not also know the training dataset used by the FDS nor its algorithm class. Therefore, the attacker has to use an alternative dataset to train its replicas and select the best-performing one among them. In particular, in the black-box scenario, the attacker has even less knowledge of their target: they do not know the features of the FDS model and its update policy. We assume that the attacker builds their aggregate features like the FDS and updates their Oracle according to a bi-weekly update policy since it leads to fewer updates of the Oracle. The attacker does not also know the actual function used to assign sample weights. Given t as the difference in hours between the Oracle training timestamp and the transaction timestamp, and k constant, we approximate it with the following linear function: $\hat{w}_{transaction} = 1 - \frac{t}{k}$.

5.3 Crafting Step: Generating Adversarial Transactions

We assume that the attacker can interact with the FDS by directly controlling the number of adversarial transactions to inject, their timestamps, and amounts. Aggregated features are directly calculated by the FDS once transactions are submitted. Other attributes cannot be decided arbitrarily by the attacker without affecting their own fraudulent goal, *e.g.*, `IBAN`, `UserID`. Therefore, we assign random values to other fields of the crafted adversarial transaction. Considering that most of the frauds have an international `IBAN_CC`, we assign a national `IBAN_CC` (i.e., "IT") with a probability of 50% and an international one in the remaining cases.

The timestamp states the specific instant of time when a transaction is executed. We design a simple algorithm for the identification and replication of the victim's temporal behavior and the selection of proper transaction timestamps. Our algorithm limits the number of frauds to be executed in the same day and hour as multiple transactions in a short period of time are suspicious. In addition, we design an algorithm for the selection of the amount of monetary value stolen with each fraud. With our algorithm, we analyze past victim spending behavior, extracting meaningful data such as the mean and the standard deviation of the transaction values, and choose the amount accordingly. Our procedure progressively increments the number of adversarial transactions during the poisoning phase of the attack to exploit the poisoned FDS. The count is the number of frauds potentially executed during each attack iteration. Similarly to the amount, the attacker computes how many transactions the victim usually makes, then they increment it according to their strategy. The attacker can adopt different attack strategies for their choice of count and amount of adversarial transactions. We design these strategies to define the will of the attacker to act more carefully or recklessly. Depending on the chosen strategy, the amount increase during the poisoning phase can be faster (higher short-term gain, but less stealthy) or slower (lower short-term gain, but more stealthy).

5.4 Attacker strategies

We design three different attackers strategies: *greedy*, *medium*, and *conservative* strategy. The greedy strategy is the most *miopic* one, as the attacker maximizes the short-term gains, committing few transactions with high amounts. With the conservative strategy, the attacker acts more carefully and slowly increases the fraud value over time. The medium strategy falls between the last two, with the attacker trying to balance the number of transactions and the amount stolen. Strategies also define the limit over the value of the attacker's adversarial transactions over time, providing upper and lower bounds. In order to avoid detection, they set an upper bound on the value increase in terms of raw amount increase and standard deviation with respect to the user spending

Table 1. Configurations of the attacker's strategies.

	Attacker's Strategy		
	Conservative	**Medium**	**Greedy**
Count Increase (%)	40	33.33	25
Amount Increase (%)	75	125	175
Min Increase (€)	25	40	50
Max Increase (€)	2,500	4,000	5,000
StD Max Increase (%)	75	100	150
Min Increment from Previous Iteration (%)	20	30	40

power. The lower bound ensures that the attacker follows their goal, maximizing profits over time. The exact values of the aforementioned variables are defined in Table 1.

6 MITIGATION APPROACH

Our mitigation approach is inspired by the adversarial training technique, adapted in the form of *adversarial data augmentation*, to a broad variety of ML models and to the domain of fraud detection [26]. It requires the following additional assumptions:

[A6] **Trusted dataset.** The training dataset of the FDS, before the attack takes place, is trusted and contains no adversarial transactions. This assumption is realistic as the financial institution can build it by having its own banking activities manually inspected by human experts, as for our dataset (see Section 4).

[A7] **Knowledge of the threat model.** The financial institution is aware of the threat model of AML attacks, such as the one described in this paper (see Section 3 and Section 5). However, the defender (i.e., the institution) has no knowledge of the specific configuration used by the attacker, e.g., the attacker's Oracle, the particular strategy adopted by the attacker, and the attacker's knowledge of the target system. This assumption captures a scenario where the institution is only aware of the existence and the general operation of adversarial attacks and, in particular, attacks against fraud detection systems, but it is not aware of the specific instance of the attacker. As shown by Jagielski et al. [30] and Paudice et al. [44], the knowledge of the threat model is often exploited to develop a defense mechanism.

6.1 Overview

The main principle behind our solution is the same as adversarial training [26]: we instruct our models on how to recognize adversarial examples by modifying the standard training procedure. As for the case of adversarial examples in the domain of image classification, the adversarial transactions crafted by the attacker are examples drawn from a different distribution from the one that generates other points within the same class. These transactions are less likely to appear in a regular banking transactions dataset and, therefore, they can be easily misclassified by the FDS. Given the similarity of the task, we build our defensive approach by drawing inspiration from the original adversarial training technique described in Goodfellow et al. [26]. In particular, we re-adapt it in the form of data augmentation, and modify randomly sampled transactions according to a set of heuristics that capture the assumptions over the attacker's behavior. As specified in Section 2.3, a similar, but more generic approach, has been also proposed by Geiping et al. [24], but we show its practical application and possible benefits in the fraud detection context. We start from the observation that adversarial training includes the attack procedure in the training procedure with the addition of a regularization term to the original stochastic gradient descent formula, which results in training the model on the original example and its adversarial counterpart

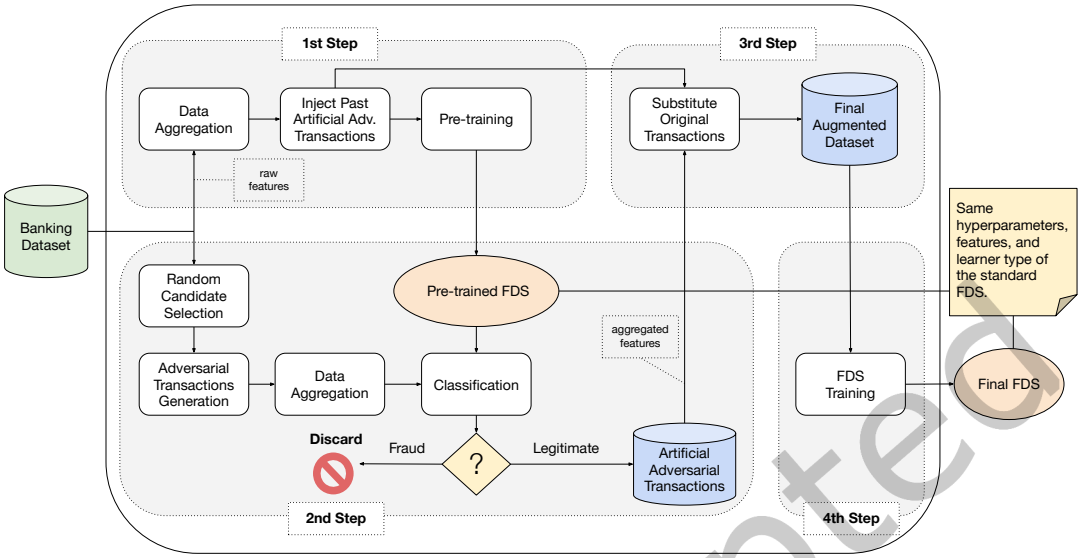


Fig. 3. Overview of the proposed mitigation.

obtained with FGSM (see Equation (1)). We obtain a similar effect by generating adversarial counterparts of real transactions from the training dataset and replacing them before the training phase of the ML model.

By anticipating the attacker, we are able to generate surrogate adversarial transactions and show them to the detector before the attack takes place. In this way, we can non-intrusively strengthen the original base FDS model, regardless of the choice of its implementation, feature set, and hyperparameters, and avoid designing an entirely new system.

As shown in Figure 3, we split the whole process into four steps: Pre-Training, Crafting, Augmentation, and Final Training. The first step consists in injecting into the training dataset the previously generated artificial adversarial transactions if any, and regularly training the FDS model using the same hyperparameters, feature set, and learner type found during the model selection steps. The *pre-trained model* will be used to classify the artificial adversarial transactions generated by our procedure. In the second step, we generate *artificial adversarial transactions* by modifying random original transactions, rather than creating new ones, and use the trained model to label the artificial samples, just like the attacker does with their Oracle. We only consider the artificial frauds that the FDS misclassifies as legitimate transactions and discard those correctly classified as frauds. In compliance with [A7], our decision rules for the generation of adversarial transactions are based on prior knowledge of the raw input features that the attacker can directly influence while committing frauds. In the third step, we replace the original candidates in our training dataset with their corresponding artificial adversarial counterparts that have successfully evaded the pre-trained model. We insert such transactions with their correct label, *i.e.*, fraud. We refer to the resulting dataset as the *adversarial augmented dataset*. Finally, we obtain the final model that will be used for the classification of the incoming transactions by training again the FDS on the adversarial augmented dataset. The described adversarial training procedure is executed for each consecutive training phase of the FDS, defined accordingly to its update policy.

6.2 Adversarial Transaction Generation

We generate artificial adversarial transactions by directly modifying random original legitimate transactions, then calculate the aggregated features and try to evade the classification of the pre-trained model. Let us recall that the FDS does not label transactions directly in input space, the one with the raw transactions feature set (e.g., `IP`, `IBAN`, `IBAN_CC`, etc.), but aggregates transactions with past user data and evaluates them in the aggregated feature space (see Section 7.2). Reverting an aggregated transaction in feature space to a sequence of raw transactions represents a complex solution also for the defender. Therefore, also their adversarial transactions can be obtained more easily directly in the raw input space.

We start by randomly picking incoming transactions in the time interval between the last training instance and the current training time of the FDS. Let us call the set of selected transactions as *candidate transactions*. Before picking the original transactions, we filter users by the number of transactions they have performed and discard those with an insufficient amount of data (*e.g.*, taking into account only those that have at least three transactions). We choose candidate transactions with different rules depending on whether they will be transformed into EPT or PPT examples. For EPT examples we allow users whose transactions have never been selected in past iterations. For PPT examples, we ensure that the transaction executioner belongs to the set of users of previously generated artificial adversarial transactions. With regard to the number of transactions to choose from, we select at each training iteration the same number N of transactions from the dataset. Only at the first iteration all of the transactions generated will be EPT ones. In the next iterations, we subdivide the total number of adversarial samples N into EPT and PPT samples: we split N into $\lceil N\rho \rceil$ PPT samples and $N - \lceil N\rho \rceil$ EPT samples.

We let the defender exploit their domain knowledge of the described AML attacks against FDSs in order to turn original transactions into adversarial counterparts that are more general than the ones created by the attacker. Such modifications can be achieved by changing the values of the features that are out of the adversary's control while leaving intact those under their control. In particular, the defender is aware that the attacker can only control the number of transactions, their amount, and timestamp and that they select the least suspicious amount and timestamp depending on the victims' past behavior. Therefore, when generating a new EPT sample, we randomly change `IBAN`, `IBAN_CC`, `ASN_CC`, `IP`, `Confirm_SMS`, `SessionID` and leave untouched `Amount` and `Timestamp`. The only attributes which are not completely random are `IBAN_CC` and `ASN_CC`. We assign them by selecting random values according to their distribution in the original dataset. When generating a new PPT sample, for each of the features as `IBAN`, `IBAN_CC`, `ASN_CC`, `IP`, `Confirm_SMS`, `SessionID` with probability $p = 0.5$, we change them with new random values or use the value generated for the last adversarial sample against the same victim. Again, we keep the same `Amount` and `Timestamp` from the original candidate transaction. This simulates an attacker changing the beneficiary international bank account number (IBAN) and/or connection for the following frauds against the same victim.

In essence, the intuition behind our strategy is the following: an attacker, aiming to imitate regular user behavior, initiates transactions that closely resemble legitimate ones in most aspects (e.g., amount and timestamp). However, these transactions possess some inherent differences that cannot be overcome. These differences reside in raw features such as `IBAN`, `IBAN_CC`, `ASN_CC`, `IP`, `SessionID`. When we transform each candidate transaction into its adversarial counterpart by changing the aforementioned features, we obtain transactions that could have been plausibly submitted by a stealthy attacker[1]. To avoid leaking the attacker's decision rules and introducing any specific trend in the selected amount and timestamp over time, we avoid choosing any specific strategy when transforming original samples into PPTs. This ensures that Assumption [A7] is not violated. Therefore, our approach may be general enough to cover for variations of the attacker's decision rules and search in any possible direction after the evasion phase.

[1]For simplicity, as the attacker does, we let the defender focus on the fraud type of information stealing.

Original candidate transaction	
Amount	3100.0
CC_ASN	IT,AS3269
confirm_SMS	1
IBAN	fcee514...
IBAN_CC	IT
SessionID	2d90d9a...
IP	cdb452a...
Timestamp	2014-10-24 18:03:31
UserID	ebbd51e...
Fraud	0

Artificial adversarial transaction	
Amount	3100.0
CC_ASN	IT,AS12874
confirm_SMS	0
IBAN	48c8c58...
IBAN_CC	ES
SessionID	1e1b0d0...
IP	eda9d1d...
Timestamp	2014-10-24 18:03:31
UserID	ebbd51e...
Fraud	1

Fig. 4. Example of transformation of a candidate transaction into an artificial adversarial transaction.

We aggregate the adversarial transactions with the entire user history and then evaluate them with the pre-trained model, as the attacker does with their Oracle, emulating their principles but for defense purposes. We save for later use the accepted artificial adversarial transactions, namely the ones that are classified as legitimate, and discard the rejected ones. We must pay attention to the fact that, when calculating aggregated features for PPT samples, we must substitute also all of the original candidates with past generated adversarial samples. Also, when transactions are aggregated, known fraudulent transactions are excluded since they are not part of the user spending behavior. Therefore, we temporarily replace the labels of past adversarial samples with the label of the legitimate class to ensure their proper inclusion in the aggregation process. We repeat the entire Crafting step from the selection of candidates until we either find the desired number N of adversarial transactions or we reach a maximum number of iterations ($i_{\max}$). After each training over time, the FDS becomes more resistant to the artificial adversarial transactions, and it takes an increasingly longer time to pick valid candidate transactions.

7 EXPERIMENTAL EVALUATION

In this section, we present the experimental setting and discuss the final results. We describe the preliminary steps of data augmentation with synthetic frauds to build a dataset close to a real-world scenario with multiple fraudulent campaigns. Using the augmented dataset, we perform the feature engineering and optimization steps for the fraud detection systems targeted by our attack. In addition, we evaluate each ML model against the synthetic fraudulent scenarios and select the attacker's Oracle. Finally, we present the metrics that we use for the evaluation of our approaches and the final results. We perform four experiments. First, we evaluate the effectiveness of the poisoning attack against the fraud detection systems, considering the different attacking strategies and knowledge scenarios. Then, we evaluate the impact of the number of artificial adversarial transactions injected in the training set on the classifier performances. Finally, we test our mitigation against the AML attacks, comparing its performance with state-of-the-art mitigations.

7.1 Dataset Augmentation with Synthetic Frauds

Real banking datasets are highly imbalanced: frauds represent around 1% of the entire dataset [15]. To replicate a real banking dataset, we augment our datasets, briefly described in Section 4, with fraudulent artificial transfers generated by a procedure validated by banking domain experts [15]. In our procedure, we group users of the datasets in different sets of "banking profiles" depending on the number of banking transfers recorded and the average volume of expenses. We randomly pick an equal number of victims from the three profiles and generate 1% of fraudulent transactions. We simulate the two most common real attack schemes: information stealing and transaction hijacking briefly described in Section 2. The main difference between the two schemes is that

Table 2. Number of users, transactions and time interval of the augmented datasets.

	Users	Transactions	Fraud Ratio	Time Interval
DA2012_13	53823	588211	0.73%	12/2012 - 09/2013
DA2014_15	58508	475767	1.02%	10/2014 - 02/2015

with the latter, the fraudster uses the user's connection, so frauds will show as transactions having the same IP, SessionID and ASN_CC of the legitimate transactions initially submitted. We generate national transactions (i.e., IBAN_CC is "IT") only in 40% of the cases. We also simulate different *strategies* for our fraudsters. We model these strategies by selecting different values for three parameters: the amount of each transaction, the count (i.e., the number of frauds against a victim), and the attack duration. The fraudster may prioritize short-term gains, selecting a high amount or a high count in a short duration. They may also adopt an opposite approach, performing long-term attacks and committing multiple low-amount transactions over time. We also assume that to avoid detection, the fraudster may study the victim's spending behavior and stealthy craft frauds which try to mimic it. Last, the fraudster may perform only one high-value transfer during their attack, with a *single fraud attack*. Using the aforementioned technique, we create artificial frauds for our datasets. Finally, we merge real and artificial data into two datasets, DA2012_13 and DA2014_15, briefly described in Table 2. DA2012_13 contains real legitimate transactions from DO2012_13 and synthetic frauds; DA2014_15 contains real legitimate transactions from DO2014_15a, real frauds from DA2014_15b, and synthetic frauds.

7.2 Modelling Target Fraud Detection Systems

We evaluate our attack and mitigation approaches against six fraud detection systems, built on top of the most common algorithms used in literature for fraud detection: logistic regression (LR) [4, 29, 58], support vector machine (SVM) [8, 34, 48], random forest (RF) [4, 8, 34, 58, 59], neural network (NN) [7, 13, 43], Extreme Gradient Boosting (XGBoost) [61], and a variant of an active learning (AL) system [37, 57]. We follow a *system-centric* approach [16] by training a supervised machine learning model to recognize anomalous global patterns from aggregated transactions.

Feature Engineering and Aggregated Features. From the augmented datasets, we calculate the aggregated datasets which comprise a set of *direct features* and *aggregated features*. Direct features are obtained by input features of each sample, while aggregated features are obtained by aggregating transactions with past legitimate transactions of the same user. The direct features are:

- **amount**: no transformation from the original attribute Amount;
- **time_{x,y}**: cyclic encoding of the time of the transaction execution, directly calculated from the Timestamp attribute. Using sine and cosine transformations, time is encoded in two dimensions: time_x and time_y. This encoding solves the distance calculation between hours directly indicated with a number in the range $[0, 24)$. For example, the distance between 23 and 22 is $23 - 22 = 1$, but the distance between midnight and 23 is $0 - 23 = -23$. We obtain the encoding as follows:

$$t = ts_h * 3,600 + ts_{min} * 60 + ts_{sec} \tag{2}$$

$$\texttt{time\_x} = \cos \frac{t * 2\pi}{86400} \tag{3}$$

$$\texttt{time\_y} = \sin \frac{t * 2\pi}{86400} \tag{4}$$

- **is_national_iban**: a Boolean value indicating if the beneficiary IBAN has the same nationality of the online bank (*i.e.*, IT country code);

- **is_international**: a Boolean value that indicates if the beneficiary IBAN has the same nationality of the customer;
- **confirm_sms**: a Boolean value that indicates if the transaction requires an SMS message for confirmation.

Before proceeding with aggregated features, we define three sets: *group*, *function*, *time*.

- ***group*** is the set of original attributes composed by IP, IBAN, IBAN_CC, ASN_CC, SessionID;
- ***function*** is the set of operations composed by count, sum, mean, std, where:
  - count is the operation that returns the count of the given instances;
  - sum is the operation that returns the sum of the amounts of the transactions;
  - mean is the operation for the calculation of the average amount of the given transactions;
  - std is the operation that calculates the standard deviation of the transaction amounts;
- ***time*** is the set of possible time spans of 1h, 1d, 7d, 14d, 30d, respectively indicating one hour, one day, seven days, fourteen days, and thirty days.

Consider *group*, *function*, *time* as a value taken from the corresponding set, then the aggregated features are:

- ***group_function_time***: obtained by grouping past user transactions by the given *group* attribute, then sliding a time window of length *time* and applying *function* on the resulting set of transactions. For example, iban_count_1d is the aggregated feature that indicates the count of transactions in the last 24 hours toward the same IBAN.
- **time_since_same_*group***: time elapsed in hours since the last transaction made by the same user and toward the same *group* attribute value. For example, time_since_same_ip is the time elapsed in hours from the last transaction executed with the same IP address.
- **time_from_previous_trans_global**: time elapsed in hours since the last transaction made by the same user.
- **difference_from_*group*_mean*time***: difference of amount between the current transaction and the set of transactions in time window long as *time* and toward the same *group* attribute value.
- **is_new_*group***: a Boolean value that indicates if the user is submitting a transaction toward the value of the given attribute group for the first time. For example, is_new_asn_cc indicates whether it is the first time a user has connected from a certain ASN and associated CC.

Update Policy and Concept Drift. The simulation of the poisoning attack covers a period of two months of incoming transactions. We design our FDS to deal with the *concept drift*, intended as the changes of the customers' spending power over time, by adopting two different solutions. First, we include new data in batches at two fixed time intervals of respectively one and two weeks, which are chosen before the simulation of the attack. We refer to the intervals as *weekly* and *bi-weekly* update policies. Then, we assign discount weights to each example in the dataset. The discount exponential function increases the importance of the most recent transactions. Given t as the time difference in hours between the timestamp of the training phase and the timestamp of the transaction and a constant $k = 4380h$ $(0.5y)$, we assign transaction weights as follows: $w_{\text{transaction}} = e^{-\frac{t}{k}}$.

Feature Selection and Hyperparameter Tuning. We split dataset DA2014_15 (see Section 4) into training and test set. The resulting test set comprises the last two months of transactions, accounting for 35.76% of the original dataset. We use the training set to select hyperparameters and feature sets of the supervised models for fraud detection under attack. We start by reducing the large initial feature space, which accounts for 174 direct and aggregated features, with a filter method for *feature selection*. We exclude from pairs of highly correlated features the ones with lower correlation to the target variable. Then, for each model, we search for an initial optimal set of hyperparameters, following a grid search approach. In particular, as a validation strategy, we minimize the cross-validation error on the training set split in 3 folds of increasing size. Naively assessing the generalization performance of our models in terms of standard accuracy, given the high-class imbalance of our datasets, may

Table 3. Final performance evaluation of the FDS models.

Model	C-Accuracy	Precision	Recall	F1-Score	FPR	FNR	W-MCC	AUC-ROC	AUC-PRC	Dataset
					FDSs					
AL	92.08%	11.52%	93.66%	20.52%	9.36%	5.80%	84.22%	97.12%	77.89%	DA2014_15
LR	92.83%	19.70%	90.54%	32.36%	4.87%	9.46%	85.76%	96.61%	56.72%	DA2014_15
NN	93.92%	20.91%	92.45%	34.10%	4.61%	6.01%	87.87%	97.89%	73.66%	DA2014_15
RF	93.20%	14.54%	93.66%	25.17%	7.27%	6.62%	86.40%	97.44%	77.20%	DA2014_15
SVM	92.91%	21.36%	90.21%	34.54%	4.38%	9.75%	85.96%	– [2]	– [2]	DA2014_15
XGBoost	94.30%	19.72%	93.62%	32.57%	5.03%	6.37%	88.60%	98.14%	81.71%	DA2014_15
					Oracle					
XGBoost	96.43%	23.5%	95.45%	37.71%	2.58%	4.55%	92.88%	99.35%	84.63%	DA2012_13

lead to incorrect model choices. A model that always outputs the legitimate class label for every test sample scores an accuracy close to 99% [13]. Another remark is that false positives and false negatives do not bring the same cost to the financial institution: the highest damage is brought by undetected fraud and not by false alarms [37]. We solve this problem by evaluating the performance of the FDSs on a custom performance metric inspired by other works [37, 58], which we refer to as *C-Accuracy*. This metric drastically increases the weight associated with the correct classification of frauds (False Negatives, True Positives) with respect to legitimate transactions. Using the definitions of Cost [58] and Normalized Cost [37], where

$$\text{Cost} = FP + k * FN \tag{5}$$

and

$$\text{Norm\_Cost} = \frac{\text{Cost}}{TN + FP + k * (TP + FN)} \tag{6}$$

We define the C-Accuracy that estimates the saved costs by the financial institutions as:

$$\text{C\_Accuracy} = 1 - \text{Norm\_Cost} \tag{7}$$

Instead of arbitrarily setting the value of k, the weight of false negatives, we empirically estimate its value as the ratio of legitimate transactions over frauds to resemble a balanced accuracy metric:

$$k = \frac{TN + FP}{TP + FN} \tag{8}$$

Then, we run an additional round of feature selection with a wrapper method to further reduce the dimensionality of the feature space and obtain unique feature sets for each FDS. We finally optimize the model hyperparameters on the final feature sets.

According to our selection steps, we obtain 5 different FDS models, with 5 different feature sets, as shown in Table 4. Logistic regression (LR) uses L2 regularization with $C = \frac{1}{\lambda} = 5.46$. The neural network (NN) model is a Feed-Forward Neural Network, composed of multiple dense layers. The first input layer has a fixed dimension given by the selected input features (see Table 4). There are two hidden layers, each comprising 32 neurons with the "tanh" activation function. A dropout layer with a dropout rate of 0.30 is placed between the hidden layers. The last layer is responsible for the binary classification task, containing a single neuron activated by the sigmoid activation function. The random forest (RF) model is composed of 40 decision trees with max depth 5 and the "entropy" criterion. The SVM has a linear kernel with the squared hinge as a loss function and $C = \frac{1}{\lambda} = 0.28$. The XGBoost model uses 32 decision trees as base learners with a max depth of 2 and a learning rate of 0.4. Our model of active learning (AL) adopts an ensemble of two models, a supervised and an unsupervised method. We use the previously described random forest model alongside an autoencoder for the unsupervised part.

Table 4. Selected feature sets of the FDS models.

Model	Features
	FDSs
Shared	iban_sum30d, amount_count7d, is_national_iban, iban_count1d, iban_count30d, iban_std7d, iban_sum1d, iban_mean30d, is_international, Amount, iban_count7d, iban_std1d, time_since_same_iban_cc, iban_cc_mean14d, iban_sum14d, iban_std14d, amount_sum1d, iban_count14d, amount_sum7d, iban_sum7d
LR	amount_sum14d, time_since_same_asn_cc, iban_cc_mean1d, iban_std30d, amount_sum30d, is_new_iban, ip_mean30d, iban_cc_mean7d, difference_from_amount_mean30d, amount_count14d, iban_cc_mean30d, Time_x
NN	amount_sum14d, iban_cc_mean1d, difference_from_iban_mean30d, iban_std30d, difference_from_iban_mean7d, is_new_iban, difference_from_iban_mean14d, ip_mean30d, iban_cc_mean7d, time_since_same_ip, amount_count14d, time_since_same_iban, difference_from_iban_mean1d, Time_x
RF, AL	time_since_same_asn_cc, iban_std30d, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_iban_mean14d, time_since_same_ip, iban_cc_mean7d, amount_count14d, time_since_same_iban, difference_from_iban_mean1d
SVM	amount_sum14d, time_since_same_asn_cc, iban_cc_mean1d, amount_sum30d, iban_cc_count1h, is_new_iban, difference_from_iban_mean14d, ip_mean30d, iban_cc_mean7d, difference_from_amount_mean30d, amount_count14d, time_since_same_iban, iban_cc_mean30d, Time_x
XGBoost	amount_sum14d, time_since_same_asn_cc, iban_cc_mean1d, iban_std30d, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_iban_mean14d, ip_mean30d, time_since_same_ip, is_new_iban, time_since_same_iban, difference_from_iban_mean1d, Time_x
	Oracle
XGBoost	time_since_same_iban, iban_count7d, difference_from_iban_mean7d, iban_std14d, iban_std7d, iban_count14d, difference_from_iban_mean14d, is_national_iban, iban_count30d, is_international, iban_sum30d, iban_std30d, iban_sum7d, difference_from_iban_mean30d, time_since_same_iban_cc, iban_mean30d, time_since_same_ip, time_since_same_asn_cc, iban_sum1d, ip_mean30d, Amount, iban_cc_mean1d, asn_cc_mean30d, amount_count7d, difference_from_amount_mean7d

Detection Performance Evaluation. We evaluate the performances of our FDSs on the test set. We use the measures of precision, recall, F1-score, false positive rate (FPR), false negative rate (FNR), area under curve of receiver operating characteristic (AUC-ROC), area under curve of precision recall curve (AUC-PRC), matthews correlation coefficient (MCC), and C-Accuracy. Table 3 collects the values scored by the FDSs. All of the FDSs achieve recall higher than 90% but have low precision values (and consequently, F1-score), between 14.54% and 21.36%. This is a direct consequence of selecting models that maximize our C-Accuracy metric: the chosen FDSs prefer to raise many false alarms in exchange for a high fraud detection rate. However, attacking suspicious classifiers represents a worst-case scenario for the attacker. In fact, a suspicious FDS may flag higher volumes of transactions as false positives, requiring more effort for the attacker to craft adversarial transactions that remain undetected.

[2]The implementation of SVM that we use does not output class probabilities, so we cannot reliably calculate the estimate for this metric.

Attacker's Oracle. We select the Oracle model on the dataset `DA2012_13`, in accordance with Assumption [A2]. In particular, we split the dataset into training and test set, the latter being the last 20% of the recorded transactions. Among the same five options that we consider for our FDSs, we choose the XGBoost model for the Oracle, as it achieves the highest C-Accuracy score on the training set. Then, we optimize its feature set and hyperparameters following the same steps we use for the FDSs. The final performance scores are reported in Table 3.

7.3 Attack Evaluation Metrics

We evaluate the impact of the attack and the effectiveness of the mitigations using *ad hoc* evaluation metrics. First, we define the following terms: F_g is the set of all the adversarial transactions generated by the attacker; F_f is the set of the adversarial transactions filtered by the attacker's Oracle, where $F_f \subseteq F_g$; F_a is the set of the adversarial transactions generated by the attacker and misclassified by the FDS, where $F_a \subseteq F_f$; F_r is the set of the adversarial transactions generated by the attacker and correctly classified by the FDS, where $F_r \subseteq F_f$; D is the set of all the transactions, where $(F_a \cup F_r) \subseteq D$, respectively with the wrong (*i.e.*, legitimate) and correct (*i.e.*, fraud) class labels; V is the set of the victims of the poisoning attack; P is the set of victims protected by the FDS during the attack; W is the set of weeks of the attack simulation, where $W = \{0..7\}$; A_f is the amount of a fraud f, where $f \in F_g$; A_w is the total stolen amount in week $w \in W$; ΔT_f is the time difference between the time of execution of fraud f and the beginning of the attack. We evaluate the performance of our attack according to the following metrics:

- **Detection Rate**. Metric that identifies the number of the victims of the attack protected by the FDS with respect to the total number of victims.

$$\text{Detection Rate} = \frac{|P|}{|V|} \tag{9}$$

- **Weekly Increase**. Metric that calculates the average increase of capital stolen from all victims by each week.

$$\text{Weekly Increase} = \frac{1}{|W|} * \sum_{w \in W} \frac{A_{w+1} - A_w}{A_w} \tag{10}$$

- **Evasion Rate**. Ratio of frauds that successfully evade the FDS with respect to the total number of frauds submitted.

$$\text{Evasion Rate} = \frac{|F_a|}{|F_f|} \tag{11}$$

- **Injection Rate**. Proportion of adversarial transactions crafted by the attacker and classified as legitimate by the attacker's Oracle with respect to the number of frauds generated.

$$\text{Injection Rate} = \frac{|F_f|}{|F_g|} \tag{12}$$

- **Poisoning Rate**. Ratio of adversarial transactions injected in the banking dataset in relation to the total number of transactions.

$$\text{Poisoning Rate} = \frac{|F_a|}{|D|} \tag{13}$$

- **Detection Time**. Metric that represents the median time in days before an adversarial transaction is detected by the FDS.

$$\text{Detection Time} = \text{Median}\{\forall f \in F_r : \Delta T_f\} \tag{14}$$

- **Money Stolen**. Metric that represents the total amount stolen against all the victims with a single attack.

$$\text{Money Stolen} = \sum_{f \in F_a} A_f \tag{15}$$

From the defender's perspective, given the baseline performance of the system against the attack, a mitigation approach should achieve lower values of Money Stolen, Poisoning Rate, Injection Rate, Evasion Rate, Detection Time, Weekly Increase and higher value of Detection Rate.

7.4 Experiment 1: Poisoning attack against banking FDSs

In this experiment, we simulate our poisoning attack approach against all of the chosen FDS models with the feature sets listed in Table 4, using every combination of the attacker's strategy (greedy, medium, conservative, see Section 5.4), attacker's knowledge of the system (white-box, gray-box, black-box, see Section 3.2), and FDS update policy (weekly, bi-weekly, see Section 7.2). For each simulation, the attacker selects thirty random victims with different spending capabilities from the banking dataset `DA2014_15`. To reduce the variance of the results, we run three simulations and provide the average of the results as the final estimate. We consider the results of this experiment as the baseline performance of our FDSs against the poisoning attack. Our results show no substantial differences in the success of the attack against FDSs using weekly and bi-weekly update policies. Therefore, in Table 5 and Table 6, we report only the results obtained against FDSs with bi-weekly update policy, as in such a scenario, their update timestamps and the ones of the attacker's Oracle are synchronized. We mention the meaningful differences between the two update policies where present. As shown by Table 5, the attacker generally meets their goal if no mitigation is employed, even with little to no knowledge of the target system at all. Most of the FDSs block an insufficient amount of EPTs within the first two weeks of the attack, as their Detection Rate ranges from 53.33% and 91.11%. This phenomenon allows attackers to poison the detection system and steal substantial money capital from their victims over time. The attacker increases their profit on average between 22.92% and 127.69%, as shown by the Weekly Increase metric. We also observe that in all knowledge scenarios, the greediest strategies tend to be more rewarding for the attacker than the more conservative ones. Under certain conditions, the medium and conservative strategies outperform the greedy one. However, we find only two meaningful patterns. First, the medium strategy wins against the random forest model only if the FDS uses a weekly update policy. Second, against the neural network, more conservative strategies win only if the model is attacked under a white-box knowledge scenario. From these patterns, we deduce that the attacker has to balance their long and short-term goals if the detector under attack is strongly suspicious (i.e., with a high number of FPs).

As shown in Table 5, the Injection Rate values obtained within the limited knowledge setting provide an idea of the *effort* required by the attacker to successfully craft adversarial transactions. On average, the attacker's Oracle rejects 9 out of 10 generated adversarial transactions. The remaining transactions that are effectively injected by the attacker, are accepted by the target fraud detection systems with acceptance rates between 49.18% and 83.12%. The latter values are provided by the Evasion Rate and depend on the particular classifier employed by the financial institution. As a side note, these findings also show that adversarial transactions generated against one Oracle are also able to transfer to different ML models, as shown by Carminati et al. [17]. However, it is important to note that the Injection Rate metric, in this scenario, is constrained by the attacker's use of the same ML algorithm as the Oracle (i.e., XGBoost, see Section 7.2). Conversely, in the white-box scenario, the attacker's Oracle is an exact replica of the target FDS, meaning that the Injection Rate in this case depends on the defender's classifier. Table 6 highlights the varying levels of effort required by the attacker across the target systems. The results show that the hardest to attack is AL, while the least amount of effort is required against SVM.

Finally, we observe that the economic impact of the attack strongly depends on the classifier employed by the FDS. The choice of hyperparameters and feature sets besides the particular learning algorithm may also

Table 5. Poisoning attack results in limited knowledge scenarios against FDSs with a bi-weekly update policy and without any mitigation. We highlight in green the best value for each metric and in red the worst.

		AL	LR	NN	RF	SVM	XGBoost
Black-box							
Greedy	Money Stolen	€62,549	€238,913	€273,443	€65,107	€349,876	€264,530
	Weekly Increase	34.48%	127.69%	101.47%	39.55%	105.41%	97.29%
	Detection Time	9.99d	4.00d	5.35d	10.34d	4.45d	9.73d
	Detection Rate	84.44%	70.00%	60.00%	91.11%	56.67%	71.11%
	Evasion Rate	53.46%	73.27%	77.69%	49.18%	76.28%	70.40%
	Injection Rate	12.63%	12.48%	13.61%	12.65%	12.66%	10.28%
	Poisoning Rate (10^{-4})	135.20%	165.32%	223.45%	114.89%	236.75%	201.74%
Medium	Money Stolen	€49,278	€299,681	€309,195	€57,037	€335,487	€148,696
	Weekly Increase	21.57%	97.15%	86.74%	27.77%	87.67%	81.66%
	Detection Time	10.47d	2.89d	3.81d	10.31d	2.61d	10.07d
	Detection Rate	87.78%	60.00%	54.44%	86.67%	61.11%	71.11%
	Evasion Rate	57.29%	79.31%	83.12%	58.30%	73.96%	68.53%
	Injection Rate	11.81%	12.25%	12.83%	13.04%	12.72%	11.78%
	Poisoning Rate (10^{-4})	135.90%	212.24%	217.14%	149.21%	197.53%	212.94%
Conservative	Money Stolen	€46,622	€230,800	€217,882	€41,995	€187,034	€146,790
	Weekly Increase	21.25%	77.38%	77.85%	22.92%	72.36%	78.10%
	Detection Time	12.53d	4.24d	6.93d	16.18d	2.69d	14.26d
	Detection Rate	83.33%	57.78%	62.22%	88.89%	66.67%	72.22%
	Evasion Rate	63.66%	79.37%	74.90%	61.91%	74.23%	69.89%
	Injection Rate	9.17%	12.48%	9.24%	9.00%	11.67%	10.27%
	Poisoning Rate (10^{-4})	168.12%	192.63%	206.64%	184.92%	153.41%	215.04%
Gray-box							
Greedy	Money Stolen	€60,338	€371,697	€310,000	€105,473	€304,903	€190,480
	Weekly Increase	40.27%	84.78%	74.12%	63.70%	83.18%	82.35%
	Detection Time	9.27d	2.64d	6.15d	13.79d	3.99d	9.63d
	Detection Rate	86.67%	63.33%	80.00%	86.67%	57.78%	76.67%
	Evasion Rate	52.16%	78.26%	70.70%	64.67%	79.08%	65.96%
	Injection Rate	13.31%	15.75%	18.49%	11.51%	13.63%	12.56%
	Poisoning Rate (10^{-4})	120.49%	201.04%	192.63%	181.43%	194.73%	191.23%
Medium	Money Stolen	€50,892	€388,342	€266,144	€64,698	€241,863	€167,102
	Weekly Increase	31.80%	82.20%	54.61%	42.02%	63.62%	56.49%
	Detection Time	10.77d	3.66d	4.28d	10.69d	3.91d	11.93d
	Detection Rate	90.00%	53.33%	72.22%	84.44%	61.11%	62.22%
	Evasion Rate	51.57%	82.21%	73.70%	64.39%	76.99%	75.54%
	Injection Rate	11.94%	14.88%	14.53%	11.33%	11.79%	10.67%
	Poisoning Rate (10^{-4})	131.00%	239.55%	165.32%	177.22%	182.83%	222.75%
Conservative	Money Stolen	€60,172	€188,065	€252,878	€74,085	€205,869	€117,235
	Weekly Increase	35.29%	59.52%	53.33%	35.60%	66.54%	63.49%
	Detection Time	13.81d	3.75d	3.72d	10.25d	3.92d	12.43d
	Detection Rate	86.67%	66.67%	66.67%	85.56%	58.89%	67.78%
	Evasion Rate	63.45%	74.61%	76.39%	55.66%	79.82%	69.35%
	Injection Rate	9.43%	12.70%	13.04%	12.45%	10.16%	9.19%
	Poisoning Rate (10^{-4})	164.62%	163.92%	202.44%	148.51%	186.33%	215.04%

influence this phenomenon. In general, our results highlight that shallower model such as logistic regression, neural network, and SVM allows the attacker to steal the highest amounts of money capital from their victims. For such models, we record the worst values of Evasion Rate, Weekly Increase, and Money Stolen. The attacker gains the highest profits mostly from the SVM classifier.

Table 6. Poisoning attack results in white-box scenario against FDSs, with a bi-weekly update policy and without any mitigation. We highlight in green the best value for each metric, in red the worst.

		AL	LR	NN	RF	SVM	XGBoost
White-box							
Greedy	Money Stolen	€279,612	€871,092	€488,120	€347,093	€1,029,231	€468,046
	Weekly Increase	51.22%	82.34%	91.00%	53.90%	81.84%	82.67%
	Detection Time	–	–	–	–	–	–
	Detection Rate	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Evasion Rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
	Injection Rate	2.08%	16.90%	12.65%	2.24%	22.18%	5.33%
	Poisoning Rate (10^{-4})	209.45%	385.21%	241.66%	208.75%	351.60%	341.80%
Medium	Money Stolen	€327,907	€785,315	€461,355	€270,600	€835,888	€441,908
	Weekly Increase	45.77%	75.17%	76.99%	59.08%	78.42%	72.52%
	Detection Time	–	–	–	–	–	–
	Detection Rate	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Evasion Rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
	Injection Rate	3.70%	11.65%	14.98%	2.38%	28.53%	5.09%
	Poisoning Rate (10^{-4})	259.87%	334.10%	261.27%	265.47%	401.31%	341.80%
Conservative	Money Stolen	€213,026	€362,366	€555,422	€265,448	€895,550	€362,870
	Weekly Increase	43.83%	59.46%	65.49%	50.39%	82.79%	66.62%
	Detection Time	–	–	–	–	–	–
	Detection Rate	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Evasion Rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
	Injection Rate	3.03%	10.42%	14.16%	2.60%	18.41%	5.38%
	Poisoning Rate (10^{-4})	322.19%	273.88%	371.91%	261.27%	423.71%	425.81%

Black-box. In the black-box scenario, the attacker has no information about the target FDS, and their attacks use surrogate knowledge. The attacker gains from a minimum of €41,995 up to €349,876. The latter value, obtained from the SVM classifier, is around 5.37 times the amount stolen from a FDS using random forest, with the same attack strategy and update policy setup. Out of the 5 detectors, we observe that only random forest is able to shield the victims from the attack at a consistent rate. The attacker can achieve an Evasion Rate of 61.91% against the latter model. Other fraud detectors allow for higher values of Evasion Rate, up to 83.12% against neural network.

Gray-box. Our results show that the additional knowledge of the gray-box scenario does not bring a consistent benefit for the attacker, as observed by Carminati et al. [17]. Interestingly, the attack success against the FDS using XGBoost is mostly unchanged from the black-box scenario. Against this model, the attacker steals at most an amount of €190,480.21. Even when unknowingly using the same algorithm of the FDS for their Oracle, the knowledge of the relative feature set does not prove beneficial. However, the attacker has slightly more success only against the model of random forest: for every considered configuration, they obtain higher profits, peaking at €105,473, around 1.61 times the attacker's best record in the black-box scenario against the same random forest model.

White-box. In this scenario, the attacker possesses the required knowledge to probe the exact blind spots of the FDS models. This enables the attacker to remain undetected and maximize their illegal profits. For the random forest model, which proved to be the best-performing model in the limited knowledge scenarios, the attacker steals from 4.74 up to 6.32 times more money than the black-box scenario, reaching at most a capital of €347,093. However, these results are sensibly better than the ones obtained by other models, even when considering limited attacker's knowledge scenarios. The outcome for SVM and logistic regression shows the full potential of the attack. For example, by adopting a greedy strategy, the attacker manages to steal from a FDS that adopts a SVM model and a bi-weekly update policy, an amount of €1,029,231 by the end of the attack. This scenario also

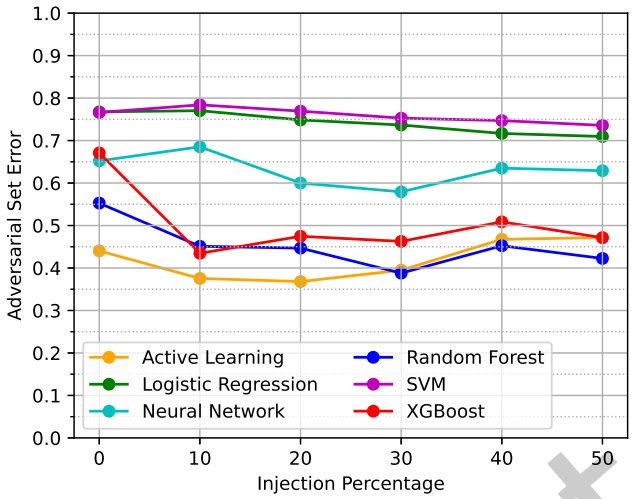


Fig. 5. Classification error over the adversarial validation set with different injection percentages.

highlights that the attacker spends more effort to craft their adversarial transactions against random forest and XGBoost and the least effort against SVM, as shown in Table 6 by the corresponding values of Injection Rate. Furthermore, if employing a weekly update policy, the outcome against the neural network model is sensibly worse for the defender under this knowledge scenario. In fact, the attacker reaches amounts of Money Stolen between €728,843 and €913,697, almost double of the Money Stolen against the same FDS with a bi-weekly update policy. However, we do not record such an increase for the other detectors.

7.5 Experiment 2: Performance trade-off with the proposed countermeasure

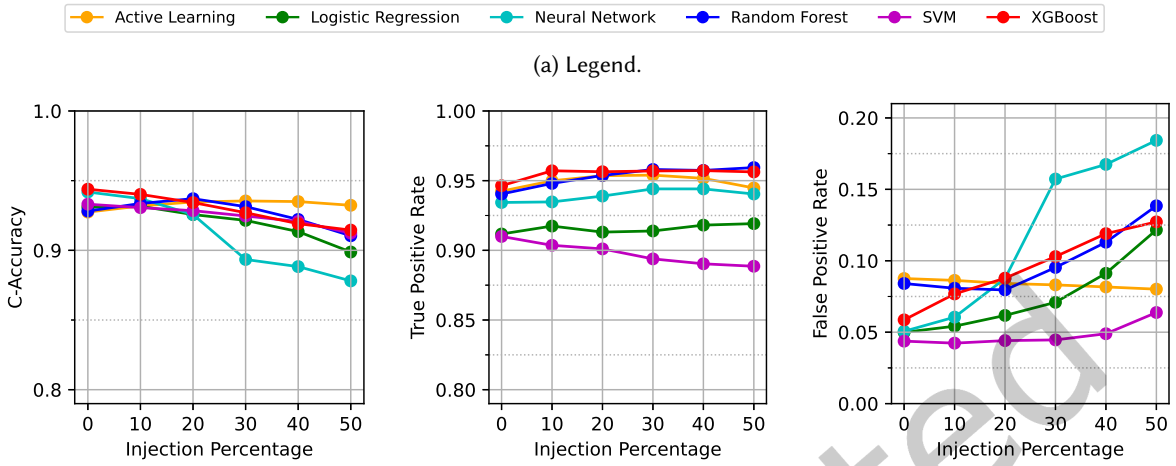
The goal of this experiment is to evaluate the impact of the number of artificial adversarial transactions injected in the training set on the classifier performances. In particular, we assess the performances ① on adversarial transactions and ② on *regular* examples of fraud. As a side effect, this experiment estimates the percentage of artificial adversarial transactions to inject to guarantee an effective mitigation of adversarial attacks. First, as in Goodfellow et al. [26], we evaluate the error rate of the FDSs, employing our countermeasure with different hyperparameters, on an adversarial validation set, i.e., a set comprising only of adversarial examples. We refer to such an error as the adversarial set error. We build our adversarial validation set by randomly picking adversarial transactions generated by the white-box attacks of the previous experiment. Thus, we avoid leaking decisions made by the attacker's Oracle trained on the attacker's dataset. We calculate the adversarial set error by assigning to each adversarial transaction a weight corresponding to the distance in hours between its timestamp and the beginning of the attack. Therefore, transactions that are far in time will have the smallest weights, as correctly identifying them will not bring any meaningful economical benefit to the defender. The final error estimate is given by the sum of the weights of the misclassified adversarial transactions over the sum of all the weights. We test different configurations of our countermeasure by varying the percentage of injected artificial transactions into the training set of the FDSs. We refer to this percentage as Injection Percentage. As shown by Figure 5, as we inject more artificial adversarial transactions, the adversarial set errors of most of the FDSs tend to decrease, but not monotonically. The errors of logistic regression and SVM are the only ones that keep decreasing after 10%. The errors of the other models tend to decrease until 30%, after which they start increasing again but never reach

the error recorded in the absence of mitigation, with the only exception of active learning. Finally, we observe that not all the models achieve comparable errors: active learning, random forest, and XGBoost achieve the best scores, reaching the lowest errors on adversarial transactions. The other models generally assess worse scores, with logistic regression and SVM obtaining the worst results, even when trained on a dataset with half-regular examples of frauds and half of the artificial ones generated by our mitigation.

Using the same set of hyperparameters of our countermeasure approach, we evaluate the performance trade-off of the FDSs on the test set of dataset `DA2014_15`, which does not contain adversarial transactions generated with our attack approach. By doing so, we assess the entity and the nature of the generalization performance trade-off of our FDSs due to the application of our defense approach. Specifically, we focus on their ability to recognize *regular* legitimate transactions and fraudulent transactions. We slide a time window of two weeks over the set, and, at each iteration, we train our FDSs with our countermeasure, evaluate the classification error on the following two weeks of transactions, and last, merge them into the training set. We then estimate the error as the average of the computed errors. In general, as we increase the quantity of artificial adversarial transactions injected, the C-Accuracy scores of the models slowly decreases. Our results show again that the behavior of the FDSs differs, with unequal performance variations. Some models are complex enough (in terms of hyperparameters, and feature sets) to distinguish artificial adversarial transactions from legitimate transactions to a greater extent. As shown in Figure 6b, the C-Accuracy scores of random forest, active learning, XGBoost, and SVM do not drop below 0.9 even with 50% of artificial frauds injected. Meanwhile, the performances of neural network significantly drop after 20%, and logistic regression after 40%. In particular, as shown in Figure 6c and Figure 6d, the performance reduction of the FDSs is mostly associated to the increase of the false positives. Again, the only exception is active learning, which shows a decrease of both true and false positive rates for Injection Percentage higher than 30%. However, as the rate of true positives does not generally decrease, except for SVM and active learning, our countermeasure approach mostly preserves the ability of the fraud detectors to recognize regular examples of frauds. Furthermore, the variation of the false positives for most of the models is relatively contained, steadily increasing each week. The only model that shows a significant divergence of false positives is the neural network, which records the steepest decrease of C-Accuracy. Therefore, it is possible to set a configuration of our countermeasure that possibly improves the response of the FDSs against the attack, while also limiting the collateral increase of the false positive rate.

7.6 Experiment 3: Performance of our countermeasure against the attack

In this experiment, we discuss the performance of our countermeasure against the poisoning attack. For each model, we use a configuration of hyperparameters that minimizes the adversarial set error and does not increase the false positives to more than double the value obtained without any mitigation. Therefore, in all of the configurations, we inject 50% artificial adversarial transactions for SVM, 40% for logistic regression, 30% for random forest, 20% for neural network and active learning, and 10% for XGBoost. The final results, listed in Table 7 and Table 8, show that our approach substantially reduces the monetary damage against all of the FDSs, in all of the attacker's knowledge scenarios and strategies adopted. In general, the XGBoost model achieves the best detection of the attack by almost completely halting it in the black and gray-box scenarios. In one of the tests performed against the XGBoost model, the attacker cannot evade the system even by having complete knowledge of the system (this explains the Evasion Rate lowering to 66.67%). The FDSs that scored the best results without any mitigation – random forest and active learning – achieved more contained improvements with our countermeasure. The former scores values of Detection Rate between 95.56% and 100%, i.e., meaning that we can also stop the attack for random forest. The latter model, instead, gains less benefit from our mitigation and represents the only exception where our approach occasionally increases the attack's success, as we observe a single increase of Money Stolen up to 31.89%. However, results collected with the weekly policy show that the



(a) Legend.

(b) C-Accuracy scores on the test set for different injection percentages.

(c) True positive rate on the test set for different injection percentages.

(d) False positive rate on the test set for different injection percentages.

Fig. 6. In-depth analysis of C-Accuracy decrease with different percentages of artificial adversarial transactions injected in the training set by our countermeasure.

attacker occasionally has slightly more success even in the other knowledge scenarios, recording few increases of Money Stolen, up to 38.70%. For the FDS that shows the weakest response against the attack in the absence of mitigation, SVM, our defense approach can reduce the Money Stolen up to 99.50%, and lower the Evasion Rate down a minimum of 8.94%, 88.80% less than the corresponding value obtained without our approach. We observe the weakest improvements on neural network, logistic regression, and active learning, for which our mitigation approach reduces the Money Stolen up to 91.88%.

Black and gray-box: evasion phase mitigation. We observe from Table 7 that for the black and gray-box configurations, with our countermeasure, the Detection Time metric generally decreases with respect to the baseline, at most by 83.63%. This suggests that our countermeasure improves the detection of the attack during the evasion phase and blocks most, if not all, of the EPTs. Depending on the number of adversarial transactions detected by the first week of the attack, we observe various improvements in Detection Rate, raising it between 75.56% (25.93% more than the baseline value) and 100%. In general, considering the results obtained in terms of Detection Time and Detection Rate, we record significant improvements of the other metrics with respect to the baseline. This is to be expected since by blocking most of the EPTs, the attacker commits transactions on behalf of fewer victims, thus stealing a reduced amount of capital and poisoning the system to a lesser extent.

Black and gray-box: poisoning phase mitigation. For the poisoning phase of the attack, meaningful metrics are the Weekly Increase, Evasion Rate, Injection Rate, and Poisoning Rate. As shown on Table 7, our mitigation approach always reduces the Poisoning Rate and Evasion Rate with respect to the baseline, respectively by at least 45.81% and 14.11%. On the Injection Rate metric at first glance, our countermeasure seems to achieve worse results than the baseline in the limited knowledge scenarios. This result is rather counter-intuitive: since the attacker has fewer victims at their disposal after the evasion phase, they generate fewer frauds, but most of the frauds they generate are accepted by the already poisoned system. Furthermore, the experimental results show that besides the overall decrease of Money Stolen and Poisoning Rate, a trend is still present, as shown by the

Table 7. Poisoning attack results in limited knowledge scenarios, against FDSs adopting a bi-weekly update policy and our mitigation approach. We highlight in red metric values that are worse than the baseline.

		AL	LR	NN	RF	SVM	XGBoost
Black-box							
Greedy	Money Stolen	€54,517 (-12.84%)	€36,243 (-84.83%)	€151,071 (-44.75%)	€4,330 (-93.35%)	€30,637 (-91.24%)	€0 (-100.00%)
	Weekly Increase	67.24% (95.04%)	102.87% (-19.44%)	86.91% (-14.34%)	26.69% (-32.52%)	204.17% (93.69%)	0.00% (-100.00%)
	Detection Time	2.82d (-71.80%)	2.37d (-40.68%)	3.99d (-25.45%)	2.91d (-71.85%)	3.05d (-31.50%)	2.80d (-71.27%)
	Detection Rate	90.00% (6.58%)	92.22% (31.75%)	75.56% (25.93%)	97.78% (7.32%)	95.56% (68.63%)	100.00% (40.62%)
	Evasion Rate	44.87% (-16.07%)	36.87% (-49.68%)	61.92% (-20.30%)	13.46% (-72.63%)	28.39% (-62.78%)	0.00% (-100.00%)
	Injection Rate	14.15% (12.01%)	14.41% (15.51%)	14.63% (7.48%)	15.92% (25.84%)	13.24% (4.57%)	15.22% (48.10%)
	Poisoning Rate (10^{-4})	58.85% (-56.47%)	37.83% (-77.12%)	119.79% (-46.39%)	14.71% (-87.19%)	24.52% (-89.64%)	0.00% (-100.00%)
Medium	Money Stolen	€32,380 (-34.29%)	€123,942 (-58.64%)	€151,665 (-50.95%)	€0 (-100.00%)	€20,926 (-93.76%)	€720 (-99.52%)
	Weekly Increase	48.53% (124.94%)	96.01% (-1.17%)	75.72% (-12.71%)	0.00% (-100.00%)	86.83% (-0.96%)	16.64% (-79.63%)
	Detection Time	2.28d (-78.24%)	3.26d (12.71%)	2.55d (-33.07%)	2.83d (-72.55%)	2.89d (10.59%)	2.83d (-71.86%)
	Detection Rate	87.78% (0.00%)	87.78% (46.30%)	80.00% (46.94%)	100.00% (15.38%)	97.78% (60.00%)	100.00% (40.62%)
	Evasion Rate	48.40% (-15.52%)	55.38% (-30.17%)	51.73% (-37.76%)	0.00% (-100.00%)	18.90% (-74.45%)	4.50% (-93.43%)
	Injection Rate	16.76% (41.93%)	16.12% (31.57%)	14.31% (11.54%)	14.48% (10.99%)	13.57% (6.72%)	17.18% (45.88%)
	Poisoning Rate (10^{-4})	57.45% (-57.73%)	71.46% (-66.33%)	98.07% (-54.84%)	0.00% (-100.00%)	18.22% (-90.78%)	3.50% (-98.35%)
Conservative	Money Stolen	€61,491 (31.89%)	€28,696 (-87.57%)	€38,868 (-82.16%)	€7,177 (-82.91%)	€23,409 (-87.48%)	€3,270 (-97.77%)
	Weekly Increase	32.53% (53.10%)	61.20% (-20.91%)	57.29% (-26.40%)	17.19% (-24.99%)	64.07% (-11.46%)	17.07% (-78.15%)
	Detection Time	2.88d (-77.03%)	2.34d (-44.87%)	2.43d (-64.90%)	2.65d (-83.63%)	2.24d (-16.62%)	2.73d (-80.87%)
	Detection Rate	84.44% (1.33%)	90.00% (55.77%)	88.89% (42.86%)	97.78% (10.00%)	98.89% (48.33%)	95.56% (32.31%)
	Evasion Rate	43.88% (-31.07%)	42.54% (-46.40%)	40.43% (-46.03%)	14.29% (-76.93%)	8.55% (-88.49%)	15.03% (-78.49%)
	Injection Rate	13.38% (45.98%)	12.65% (1.35%)	14.17% (53.39%)	14.78% (64.18%)	15.42% (32.09%)	14.94% (45.43%)
	Poisoning Rate (10^{-4})	71.46% (-57.50%)	46.24% (-76.00%)	50.44% (-75.59%)	16.81% (-90.91%)	7.01% (-95.43%)	16.11% (-92.51%)
Gray-box							
Greedy	Money Stolen	€17,725 (-70.62%)	€63,516 (-82.91%)	€164,229 (-47.02%)	€30,054 (-71.51%)	€96,294 (-68.42%)	€0 (-100.00%)
	Weekly Increase	59.59% (47.95%)	77.39% (-8.72%)	74.47% (0.47%)	59.14% (-7.15%)	256.13% (207.91%)	0.00% (-100.00%)
	Detection Time	2.94d (-68.32%)	2.49d (-5.54%)	4.51d (-26.71%)	2.75d (-80.09%)	2.39d (-40.14%)	2.30d (-76.10%)
	Detection Rate	94.44% (8.97%)	88.89% (40.35%)	82.22% (2.78%)	95.56% (10.26%)	95.56% (65.38%)	100.00% (30.43%)
	Evasion Rate	29.89% (-42.70%)	46.44% (-40.66%)	60.72% (-14.11%)	22.86% (-64.64%)	32.10% (-59.40%)	0.00% (-100.00%)
	Injection Rate	14.83% (11.40%)	14.84% (-5.81%)	16.99% (-8.11%)	14.34% (24.58%)	14.41% (5.68%)	15.71% (25.08%)
	Poisoning Rate (10^{-4})	28.02% (-76.74%)	51.84% (-74.21%)	104.38% (-45.81%)	27.32% (-84.94%)	28.72% (-85.25%)	0.00% (-100.00%)
Medium	Money Stolen	€22,883 (-55.04%)	€99,872 (-74.28%)	€21,607 (-91.88%)	€9,162 (-85.84%)	€10,752 (-95.55%)	€0 (-100.00%)
	Weekly Increase	13.70% (-56.91%)	78.48% (-4.53%)	43.80% (-19.80%)	52.19% (24.20%)	64.52% (1.41%)	0.00% (-100.00%)
	Detection Time	2.91d (-73.00%)	3.74d (2.07%)	3.49d (-18.38%)	2.30d (-78.49%)	3.22d (-17.64%)	2.44d (-79.55%)
	Detection Rate	95.56% (6.17%)	87.78% (64.58%)	90.00% (24.62%)	95.56% (13.16%)	96.67% (58.18%)	100.00% (60.71%)
	Evasion Rate	18.52% (-64.09%)	51.80% (-36.99%)	44.25% (-39.96%)	20.71% (-67.83%)	17.82% (-76.85%)	0.00% (-100.00%)
	Injection Rate	15.87% (32.90%)	13.83% (-7.05%)	14.56% (0.18%)	16.03% (41.39%)	15.15% (28.46%)	14.22% (33.30%)
	Poisoning Rate (10^{-4})	28.02% (-78.61%)	68.65% (-71.34%)	56.05% (-66.10%)	23.12% (-86.95%)	13.31% (-92.72%)	0.00% (-100.00%)
Conservative	Money Stolen	€35,483 (-41.03%)	€98,163 (-47.80%)	€105,742 (-58.18%)	€2,785 (-96.24%)	€1,024 (-99.50%)	€0 (-100.00%)
	Weekly Increase	34.06% (-3.49%)	79.59% (33.73%)	67.38% (26.35%)	22.83% (-35.88%)	27.22% (-59.10%)	0.00% (-100.00%)
	Detection Time	3.08d (-77.66%)	3.19d (-14.88%)	2.78d (-25.33%)	2.68d (-73.81%)	3.12d (-20.51%)	2.39d (-80.80%)
	Detection Rate	88.89% (2.56%)	84.44% (26.67%)	82.22% (23.33%)	97.78% (14.29%)	97.78% (66.04%)	100.00% (47.54%)
	Evasion Rate	41.55% (-34.53%)	57.21% (-23.32%)	56.16% (-26.48%)	9.72% (-82.53%)	8.94% (-88.80%)	0.00% (-100.00%)
	Injection Rate	13.58% (44.03%)	14.98% (17.91%)	16.59% (27.24%)	13.37% (7.37%)	13.89% (36.69%)	15.11% (64.42%)
	Poisoning Rate (10^{-4})	57.45% (-65.10%)	76.36% (-53.42%)	97.37% (-51.90%)	9.81% (-93.40%)	6.31% (-96.62%)	0.00% (-100.00%)

Weekly Increase metric. The latter metric is generally lower from the baseline; however, in some situations, our approach leads to a few occasional higher increases. We record some of the high increases of Weekly Increase for SVM and active learning, respectively, up to 207.91% and 124.94%. Recall that for such cases, the Money Stolen is still reduced by 91.24% and 34.29%, respectively. However, this phenomenon suggests that once the attacker successfully evades the FDS with an EPT, the target FDS is, at that point, poisoned. The attacker can keep exploiting the FDS, which becomes gradually more accustomed to their behavior and struggles to detect the PPTs submitted by the attacker. In conclusion, our approach can stop most of the transactions during the first

Table 8. Poisoning attack results in the white-box knowledge scenario, against FDSs adopting a bi-weekly update policy and our mitigation approach. We highlight in red metric values that are worse than the baseline.

		AL	LR	NN	RF	SVM	XGBoost
White-box							
Greedy	Money Stolen	€210,141 (-24.85%)	€360,892 (-58.57%)	€374,888 (-23.20%)	€126,509 (-63.55%)	€161,967 (-84.26%)	€146,059 (-68.79%)
	Weekly Increase	81.90% (59.90%)	84.01% (2.03%)	96.03% (5.52%)	79.14% (46.84%)	75.86% (-7.30%)	72.74% (-12.02%)
	Detection Time	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)
	Detection Rate	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)
	Evasion Rate	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)
	Injection Rate	6.18% (197.30%)	11.45% (-32.25%)	10.23% (-19.11%)	4.99% (123.14%)	9.87% (-55.50%)	6.90% (29.51%)
	Poisoning Rate (10^{-4})	131.70% (-37.12%)	156.92% (-59.26%)	222.05% (-8.12%)	107.18% (-48.65%)	91.07% (-74.10%)	104.39% (-69.46%)
Medium	Money Stolen	€240,591 (-26.63%)	€230,812 (-70.61%)	€377,757 (-18.12%)	€76,589 (-71.70%)	€307,815 (-63.18%)	€84,849 (-80.80%)
	Weekly Increase	96.31% (110.42%)	77.52% (3.12%)	94.17% (22.32%)	55.09% (-6.74%)	91.87% (17.15%)	79.97% (10.28%)
	Detection Time	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)
	Detection Rate	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)
	Evasion Rate	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)
	Injection Rate	7.17% (93.78%)	10.41% (-10.69%)	9.59% (-36.02%)	4.64% (95.19%)	13.61% (-52.29%)	6.60% (29.75%)
	Poisoning Rate (10^{-4})	182.83% (-29.64%)	118.40% (-64.56%)	212.94% (-18.50%)	58.85% (-77.83%)	125.40% (-68.75%)	107.18% (-68.64%)
Conservative	Money Stolen	€169,580 (-20.39%)	€292,364 (-19.32%)	€226,713 (-59.18%)	€64,540 (-75.69%)	€270,540 (-69.79%)	€76,353 (-78.96%)
	Weekly Increase	69.35% (58.22%)	68.71% (15.55%)	74.74% (14.12%)	60.34% (19.74%)	77.42% (-6.48%)	44.91% (-32.60%)
	Detection Time	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)
	Detection Rate	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)
	Evasion Rate	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	66.67% (-33.33%)
	Injection Rate	5.88% (93.98%)	9.19% (-11.74%)	9.50% (-32.90%)	4.20% (61.43%)	18.88% (2.54%)	2.76% (-48.64%)
	Poisoning Rate (10^{-4})	146.41% (-54.56%)	135.91% (-50.38%)	186.34% (-49.90%)	69.36% (-73.45%)	191.24% (-54.87%)	58.15% (-86.34%)

evasion attempts, reducing the attack's impact as soon as possible, but it does not detect PPTs to the same degree of efficiency.

White-box scenario. According to the principle of security through obscurity, we evaluate our countermeasure in a white-box scenario, assuming that the attacker knows every detail of the system and its mitigation. In this scenario, the attacker achieves the best results for all the considered metrics, but they cannot achieve the same devastating results obtained without mitigation. By having complete knowledge of the system, the attacker is still never detected by the FDSs, achieving again 0% Detection Rate and 100% Evasion Rate (see Table 8). However, with our mitigation approach, the overall Money Stolen is reduced from 18.12% up to 84.26% from the baseline. To observe this phenomenon, let us consider the metric of Poisoning Rate, which tells us how many transactions have been successfully injected into the banking dataset by the attacker. With our countermeasure, this metric is reduced between 8.12% and 86.34%. This shows that the attacker cannot generate EPTs only by choosing the least suspicious timestamp and the most probable amount for any given victim. Even by progressively reducing the value of EPTs, the attacker evades the system against fewer victims. The reduction of Poisoning Rate, even in the white-box scenario, hints that the attacker needs to improve their strategy for evading the detector. Furthermore, in this scenario, we observe the first consistent improvement of the Injection Rate for three of the models, logistic regression, neural network, and SVM. We quantify these reductions between 10.69% and 55.50%. Recall that the Injection Rate tells us how many of the frauds generated by the attacker are effectively accepted by their Oracle, which, in this case, is the FDS itself. The lower this value is, the less of the frauds generated by the attacker are accepted by the FDS, and the more is the computational effort required for them to find transactions that evade classification. In conclusion, this result suggests that the attacker, for three of the tested models, usually crafts and injects adversarial transactions with the increased computational effort needed without mitigation.

Table 9. Results of our poisoning attack against FDSs with a bi-weekly and different mitigation approaches, in limited attacker's knowledge scenarios. For each metric, we provide the average on all attacker's strategies. For each machine learning model we highlight in red the worst value among all the tested mitigations, and in green the best.

		AL	LR	NN	RF	SVM	XGBoost
Black-box							
Money Stolen	AD (OTS)	€56,449 (6.88%)	€250,027 (-2.51%)	€297,387 (11.45%)	€68,122 (24.51%)	€321,947 (10.71%)	€242,523 (29.92%)
	AD (AE)	€61,616 (16.66%)	€288,605 (12.53%)	€227,809 (-14.63%)	€79,861 (45.96%)	€235,640 (-18.97%)	€178,045 (-4.62%)
	Our approach	€49,463 (-6.35%)	€62,960 (-75.45%)	€113,868 (-57.33%)	€3,836 (-92.99%)	€24,991 (-91.41%)	€1,330 (-99.29%)
	Adv. training	–	–	€301,997 (13.18%)	–	–	–
Weekly Increase	AD (OTS)	30.79% (19.51%)	86.91% (-13.73%)	94.13% (6.14%)	32.56% (8.23%)	88.34% (-0.15%)	94.34% (10.11%)
	AD (AE)	29.79% (15.63%)	97.65% (-3.06%)	83.46% (-5.89%)	38.03% (26.41%)	80.08% (-9.49%)	80.34% (-6.24%)
	Our approach	49.43% (91.86%)	86.70% (-13.94%)	73.31% (-17.34%)	14.63% (-51.37%)	118.36% (33.77%)	11.23% (-86.89%)
	Adv. training	–	–	103.73% (16.96%)	–	–	–
Detection Time	AD (OTS)	10.04d (-8.69%)	3.14d (-15.25%)	4.18d (-21.99%)	13.41d (9.25%)	3.23d (-0.62%)	15.67d (38.02%)
	AD (AE)	11.05d (0.48%)	3.13d (-15.77%)	4.30d (-19.73%)	14.99d (22.10%)	3.31d (1.87%)	14.52d (27.83%)
	Our approach	2.66d (-75.83%)	2.66d (-28.41%)	2.99d (-44.26%)	2.80d (-77.22%)	2.73d (-16.13%)	2.79d (-75.46%)
	Adv. training	–	–	3.44d (-35.90%)	–	–	–
Detection Rate	AD (OTS)	89.63% (5.22%)	66.30% (5.92%)	62.96% (6.92%)	87.04% (-2.08%)	62.59% (1.81%)	64.81% (-9.33%)
	AD (AE)	83.70% (-1.74%)	68.89% (10.06%)	65.93% (11.95%)	81.48% (-8.33%)	61.48% (0.00%)	70.37% (-1.55%)
	Our approach	87.41% (2.61%)	90.00% (43.79%)	81.48% (38.36%)	98.52% (10.83%)	97.41% (58.43%)	98.52% (37.82%)
	Adv. training	–	–	66.30% (12.58%)	–	–	–
Evasion Rate	AD (OTS)	51.67% (-11.13%)	76.01% (-1.69%)	77.34% (-1.57%)	59.84% (5.97%)	79.39% (6.10%)	72.58% (4.27%)
	AD (AE)	56.25% (-3.24%)	74.51% (-3.63%)	75.58% (-3.81%)	63.08% (11.71%)	78.42% (4.81%)	69.95% (0.50%)
	Our approach	45.72% (-21.36%)	44.93% (-41.89%)	51.36% (-34.63%)	9.25% (-83.62%)	18.61% (-75.13%)	6.51% (-90.64%)
	Adv. training	–	–	75.72% (-3.63%)	–	–	–
Injection Rate	AD (OTS)	12.81% (14.39%)	12.31% (-0.76%)	12.46% (4.76%)	12.84% (11.02%)	13.19% (6.83%)	10.00% (-7.22%)
	AD (AE)	12.09% (7.92%)	12.68% (2.26%)	11.77% (-1.06%)	10.55% (-8.83%)	12.68% (2.67%)	10.92% (1.37%)
	Our approach	14.76% (31.79%)	14.39% (16.05%)	14.37% (20.83%)	15.06% (30.20%)	14.08% (13.97%)	15.78% (46.44%)
	Adv. training	–	–	11.46% (-3.67%)	–	–	–
Poisoning Rate (10^{-4})	AD (OTS)	127.49% (-12.92%)	168.82% (-11.18%)	194.03% (-10.06%)	161.58% (7.96%)	194.27% (-0.83%)	241.19% (14.90%)
	AD (AE)	148.74% (1.59%)	173.02% (-8.97%)	188.20% (-12.77%)	186.09% (24.33%)	191.23% (-2.38%)	221.58% (5.56%)
	Our approach	62.58% (-57.25%)	51.84% (-72.72%)	89.43% (-58.55%)	10.51% (-92.98%)	16.58% (-91.54%)	6.54% (-96.88%)
	Adv. training	–	–	180.03% (-16.56%)	–	–	–
Gray-box							
Money Stolen	AD (OTS)	€51,083 (-10.59%)	€280,430 (-11.27%)	€300,112 (8.60%)	€77,425 (-4.90%)	€317,273 (26.46%)	€170,570 (7.77%)
	AD (AE)	€41,586 (-27.21%)	€267,961 (-15.21%)	€264,225 (-4.38%)	€55,604 (-31.71%)	€250,216 (-0.26%)	€163,914 (3.56%)
	Our approach	€25,364 (-55.61%)	€87,184 (-72.41%)	€97,193 (-64.83%)	€14,000 (-82.80%)	€36,023 (-85.64%)	€0 (-100.00%)
	Adv. training	–	–	€265,699 (-3.85%)	–	–	–
Weekly Increase	AD (OTS)	31.76% (-11.25%)	81.80% (8.34%)	69.09% (13.84%)	45.39% (-3.65%)	76.03% (6.92%)	73.06% (8.34%)
	AD (AE)	24.53% (-31.45%)	80.70% (6.89%)	69.35% (14.27%)	35.44% (-24.76%)	72.12% (1.41%)	69.92% (3.68%)
	Our approach	35.78% (-0.01%)	78.49% (3.95%)	61.88% (1.97%)	44.72% (-5.07%)	115.95% (63.05%)	0.00% (-100.00%)
	Adv. training	–	–	63.10% (3.97%)	–	–	–
Detection Time	AD (OTS)	10.62d (-5.84%)	3.56d (6.39%)	4.48d (-5.05%)	12.91d (11.51%)	3.85d (-2.35%)	15.66d (38.17%)
	AD (AE)	10.91d (-3.27%)	3.31d (-1.21%)	4.50d (-4.68%)	12.25d (5.81%)	3.06d (-22.49%)	14.50d (27.93%)
	Our approach	2.98d (-73.62%)	3.14d (-6.25%)	3.59d (-23.83%)	2.58d (-77.74%)	2.91d (-26.18%)	2.38d (-79.03%)
	Adv. training	–	–	3.95d (-16.28%)	–	–	–
Detection Rate	AD (OTS)	87.78% (0.00%)	61.85% (1.21%)	71.85% (-1.52%)	87.41% (2.16%)	57.04% (-3.75%)	70.00% (1.61%)
	AD (AE)	92.96% (5.91%)	62.96% (3.03%)	74.07% (1.52%)	86.67% (1.30%)	60.74% (2.50%)	68.15% (-1.08%)
	Our approach	92.96% (5.91%)	87.04% (42.42%)	84.81% (16.24%)	96.30% (12.55%)	96.67% (63.12%)	100.00% (45.16%)
	Adv. training	–	–	69.63% (-4.57%)	–	–	–
Evasion Rate	AD (OTS)	52.28% (-6.19%)	77.12% (-1.59%)	73.21% (-0.52%)	59.10% (-4.02%)	80.87% (2.85%)	71.66% (1.96%)
	AD (AE)	50.98% (-8.51%)	76.04% (-2.96%)	72.29% (-1.78%)	59.34% (-3.63%)	79.81% (1.50%)	71.59% (1.85%)
	Our approach	29.98% (-46.20%)	51.82% (-33.88%)	53.71% (-27.02%)	17.77% (-71.14%)	19.62% (-75.04%)	0.00% (-100.00%)
	Adv. training	–	–	75.70% (2.86%)	–	–	–
Injection Rate	AD (OTS)	13.33% (15.32%)	13.56% (-6.10%)	13.95% (-9.12%)	11.77% (0.03%)	12.09% (1.93%)	10.98% (1.62%)
	AD (AE)	11.95% (3.38%)	13.18% (-8.74%)	14.99% (-2.38%)	10.92% (-7.20%)	11.64% (-1.87%)	11.41% (5.62%)
	Our approach	14.76% (27.67%)	14.55% (0.72%)	16.05% (4.51%)	14.58% (23.91%)	14.48% (22.08%)	15.02% (38.94%)
	Adv. training	–	–	14.65% (-4.57%)	–	–	–
Poisoning Rate (10^{-4})	AD (OTS)	131.93% (-4.88%)	194.50% (-3.47%)	178.16% (-4.62%)	163.68% (-3.18%)	195.67% (4.10%)	229.98% (9.69%)
	AD (AE)	123.76% (-10.77%)	191.00% (-5.21%)	173.25% (-7.25%)	151.54% (-10.36%)	180.96% (-3.73%)	219.71% (4.79%)
	Our approach	37.83% (-72.73%)	65.62% (-67.43%)	85.93% (-54.00%)	20.08% (-88.12%)	16.11% (-91.43%)	0.00% (-100.00%)
	Adv. training	–	–	194.50% (4.12%)	–	–	–

Table 10. Results of our poisoning attack against FDSs with a bi-weekly and different mitigation approaches, in the white-box scenario. For each metric, we provide the average on all attacker's strategies. For each machine learning model we highlight in red the worst value among all the tested mitigations, and in green the best.

		AL	LR	NN	RF	SVM	XGBoost
White-box							
Money Stolen	AD (OTS)	€285,762 (4.48%)	€649,354 (-3.50%)	€697,900 (39.13%)	€284,469 (-3.37%)	€766,343 (-16.72%)	€390,891 (-7.87%)
	AD (AE)	€309,838 (13.28%)	€751,404 (11.66%)	€691,589 (37.87%)	€289,942 (-1.51%)	€744,962 (-19.05%)	€476,434 (12.29%)
	Our approach	€206,770 (-24.40%)	€294,690 (-56.21%)	€326,453 (-34.92%)	€89,213 (-69.69%)	€246,774 (-73.18%)	€102,420 (-75.86%)
	Adv. training	–	–	€472,592 (-5.79%)	–	–	–
Weekly Increase	AD (OTS)	46.61% (-0.71%)	70.01% (-3.20%)	83.71% (7.56%)	51.61% (-5.22%)	71.64% (-11.58%)	71.74% (-2.98%)
	AD (AE)	53.15% (13.23%)	69.49% (-3.91%)	83.28% (7.00%)	49.97% (-8.24%)	74.61% (-7.90%)	73.27% (-0.90%)
	Our approach	82.52% (75.79%)	76.75% (6.11%)	88.31% (13.47%)	64.86% (19.10%)	81.72% (0.87%)	65.87% (-10.91%)
	Adv. training	–	–	69.68% (-10.46%)	–	–	–
Detection Time	AD (OTS)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)
	AD (AE)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)
	Our approach	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)	– (0.00%)
	Adv. training	–	–	– (0.00%)	–	–	–
Detection Rate	AD (OTS)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)
	AD (AE)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)
	Our approach	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)	0.00% (0.00%)
	Adv. training	–	–	0.00% (0.00%)	–	–	–
Evasion Rate	AD (OTS)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)
	AD (AE)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)
	Our approach	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	100.00% (0.00%)	88.89% (-11.11%)
	Adv. training	–	–	100.00% (0.00%)	–	–	–
Injection Rate	AD (OTS)	2.58% (-12.29%)	13.14% (1.19%)	15.55% (11.64%)	2.95% (22.66%)	20.50% (-11.04%)	5.22% (-0.95%)
	AD (AE)	2.74% (-6.84%)	14.20% (9.32%)	14.04% (0.76%)	2.77% (15.21%)	22.28% (-3.28%)	5.72% (8.60%)
	Our approach	6.41% (118.29%)	10.35% (-20.32%)	9.77% (-29.84%)	4.61% (91.69%)	14.12% (-38.71%)	5.42% (2.96%)
	Adv. training	–	–	17.48% (25.48%)	–	–	–
Poisoning Rate (10^{-4})	AD (OTS)	246.57% (-6.55%)	333.39% (0.70%)	355.10% (21.77%)	281.35% (14.76%)	323.36% (-17.55%)	378.67% (2.40%)
	AD (AE)	251.00% (-4.86%)	352.77% (6.56%)	363.03% (24.49%)	275.98% (12.57%)	368.40% (-6.07%)	377.97% (2.21%)
	Our approach	153.65% (-41.76%)	137.07% (-58.60%)	207.11% (-28.98%)	78.46% (-68.00%)	135.90% (-65.35%)	89.91% (-75.69%)
	Adv. training	–	–	284.15% (-2.56%)	–	–	–

7.7 Experiment 4: Performance comparison with other countermeasures

In this experiment, we compare the performance of our countermeasure approach with two state-of-the-art approaches for mitigating AML attacks. First, we test the approach based on anomaly detection with different unsupervised algorithms, autoencoder (AE), a type of feed-forward neural network, and one-time sampling (OTS) [52], based on k-Nearest Neighbor. Then, we study an adversarial training strategy originally conceived for adversarial examples in image classification [26]. Our results show no meaningful differences in the effect of the tested mitigations that depend on the attacker's strategies or the two update policies. Therefore, we provide in two distinct tables, Table 9 and Table 10, the average of each metric for all the attacker's strategies against the fraud detection systems, adopting the bi-weekly update policy as a baseline again.

Anomaly Detection. As shown in Table 9 and Table 10, the mitigation based on anomaly detection, with both one-time sampling ("AD (OTS)") and the autoencoders ("AD (AE)"), does not bring any consistent benefit against the attack with respect to the baseline performance of the FDSs. This result confirms our expectations, given the assumptions over the adversarial transactions generated by the attacker: such frauds are not outliers with respect to regular banking transactions. On average, the results obtained with both anomaly detectors tend to oscillate between lower and higher values of Money Stolen. We attribute such oscillations to the choice of different victims made by the attacker at the beginning of the simulation rather than to the tested approach actively improving the performances of the FDSs. We mitigated such a side effect by repeating the experiments three times. If we take metrics that link to both the evasion and the poisoning phase of the attack, we can also observe no distinct improvement. However, with the one-time sampling algorithm, this approach achieves slightly better values in terms of Money Stolen in the white-box scenario for all of the models, except for active learning

and neural network model, for which it allows the attacker to steal respectively on average 4.48% and 39.13% more capital. This result may suggest that even if the attacker is slowly poisoning the system, only in the long run, this mitigation may start detecting transactions that drift largely from the regular user behavior and force the attacker to commit transactions with lower monetary value. OTS also reduces the average Weekly Increase in the white-box scenario, from 2.98% to 11.58%. Remark that with our countermeasure, in the same knowledge scenario, the total capital stolen by the attacker is, on average, reduced between 24.40% and 75.86%.

Adversarial Training. For this mitigation, the only meaningful comparisons regard the results obtained with all of the other FDS configurations that adopt the neural network model. The neural network trained with adversarial training, which we refer to as "Adv. training" in Table 9 and Table 10, shows neither constant improvements nor excessive decay of the baseline performance of the neural network model against the poisoning attack. On average, this mitigation achieves slightly better results than the baseline only in the gray and white-box scenarios, but it does not match the results obtained with our mitigation approach. We observe that, on average, of the attacker's strategies, the Money Stolen is respectively 3.85% and 5.79% lower than the result obtained with the regularly trained neural network model. However, with our countermeasure, the monetary damage is reduced on average by 64.83% and 34.92%. Adversarial training obtains sensibly worse results in the black-box scenario, where it allows the attacker to steal on average 13.18% more than in the absence of the mitigation. With respect to the other metrics, we observe no meaningful improvement with respect to the standard neural network model. In conclusion, by adopting adversarial training – based on FGSM [26] – as a *generic* adversarial training strategy, we bring no benefits to the FDSs with respect to their standard performance. This shows that adversarial training techniques bring small benefits over unseen attacks [5]: strengthening a model against FGSM may not provide a sufficient cover for the adversarial transactions generated according to our poisoning attack and the evasion attack proposed by Carminati et al. [17]. The adversarial samples crafted by the attacker resemble legitimate transactions only by their direct features (i.e., `Amount`, `Timestamp`, etc.). The attacker tries to mimic the victim during the evasion phase, selecting amounts and timestamps that appear less suspicious to our FDSs. Probably, the same samples may show in feature space very different perturbations than the ones obtained with the Fast Gradient Sign Method.

8 LIMITATIONS AND FUTURE WORKS

The datasets used in this work come from the same institution; therefore, even if not belonging to the same temporal period, the attacker trains their Oracle on a dataset that shows similarities to the one used by the FDSs under attack. Consequently, in our experimental settings, we did not consider the possible impacts of different data distributions. An interesting development could consist in analyzing such a relation by having the attacker and the defender use different training data in terms of data distribution and size, and ultimately evaluate the effects on our attack and defense approach. Our banking datasets also lack a sufficient amount of real frauds, so we had to augment them with artificial data.

Another important aspect is that we test our attack and defense approaches against system-centric FDSs. An interesting development could be to evaluate the efficacy of our attack and countermeasure against FDSs with different architectures, including those built with the application of deep learning. Such systems may be able to automatically find better features from data and eventually achieve better results overall. The last important limitation that we identified is that our mitigation approach seems to be more effective in countering the evasion phase of the poisoning attack. After the poisoning phase, a different approach may be required to distinguish transactions placed by a stealthy attacker from the ones of their victims. There is a large room for improvement: our procedure for the generation of PPTs is too general and does not capture as effectively the behavior of the attacker. Possible solutions could consist in increasing the amount of candidate PPTs over time or forcing the procedure to find transactions with a higher amount of the previously generated adversarial transactions

against the same victim. As a consequence of searching candidate PPTs in a reduced portion of feature space, the performance trade-off of such an approach could be more contained with respect to our current solution. Therefore, we propose, as an input for future research, to extend our countermeasure by exploring different strategies for generating PPTs and studying their efficacy against different attacker's strategies, under a game theoretic framework. In addition, the capabilities and limitations of the defense strategy that we proposed can be further explored under different, more formal perspectives, potentially taking into account whether formal guarantees [32] can be formulated, as done in the malware detection domain [46]. Paired with the experimental results that we have shown in our work, such a study may bring a more comprehensive vision of this problem. Another possible development could consist in evaluating the robustness of our defense approach against adaptive attacks [56]. Future works may also be directed toward our poisoning attack, rather than the defense. In particular, a significant extension of our work may consist in identifying and exploring possible strategies for the retrieval of an input transaction, or part of it, from a sample in feature space (often referred to through our work as "aggregated transaction"). This study may allow us to find a feasible approach to estimate the gradient of the loss of the target FDS model for a given input transaction. Furthermore, this will allow us to extend our current formulation of the attack to the commonly employed optimization problem and allow the attacker to adapt AML attacks developed in other contexts.

9 CONCLUSIONS

This paper proposed ① a novel approach to craft poisoning samples that expands existing solutions and ② a novel defense strategy, directly inspired by adversarial training [26], that mitigates adversarial machine learning attacks – the one proposed in this paper and by Carminati et al. [17] – overcoming the challenges of the fraud detection domain. We validated our offensive approach by simulating attacks against six different FDSs with two update policies, under increasing attacker's knowledge scenarios, and different strategies. In our mitigation approach, we strengthened the supervised, system-centric, fraud detection system classifiers against evasion by adapting the principle behind adversarial training, originally conceived for protecting visual classifiers against the evasion of adversarial examples, to a different domain and to a broad class of machine learning models. Our results showed that our attack was successful even in a black-box scenario, i.e., a case in which the attacker doesn't have any information about the target system. The attacks were detected between 55% and 91% of the time, but they lasted enough time to steal significant amounts of money. We showed that FDSs are more vulnerable to greedier attack strategies. With our mitigation, we effectively reduced the economic damage of the attacks in every knowledge scenario, by recognizing most of the adversarial transactions crafted by the attacker during the first evasion attempt. We achieved this result at a low generalization performance trade-off and we were able to consistently detect the attack as soon as possible, reducing the base detection time by at most 90%. Even in the worst-case scenario, where the attacker possesses every detail of the fraud detection system, we were able to reduce the number of frauds injected by the attacker on average from 28.98% to 75.69%. We showed that naively adopting existing defense mechanisms – anomaly detection and adversarial training – do not pose an effective strategy for the defender since they fail at recognizing adversarial transactions carefully crafted by a stealthy attacker that aims at replicating the original user behavior and fooling the classifier. Our results suggest that we can reduce and sometimes even completely stop the attack by building a strong supervised system-centric detector that blocks all of the first attempts at evasion.

REFERENCES

[1] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. 2016. Fraud detection system: A survey. J. Netw. Comput. Appl. 68 (2016), 90–113. https://doi.org/10.1016/j.jnca.2016.04.007

[2] Idan Achituve, Sarit Kraus, and Jacob Goldberger. 2019. Interpretable Online Banking Fraud Detection Based On Hierarchical Attention Mechanism. In 29th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2019, Pittsburgh, PA, USA, October

13-16, 2019. IEEE, 1–6. https://doi.org/10.1109/MLSP.2019.8918896
[3] Khaled Gubran Al-Hashedi and Pritheega Magalingam. 2021. Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019. Comput. Sci. Rev. 40 (2021), 100402. https://doi.org/10.1016/j.cosrev.2021.100402
[4] Alejandro Correa Bahnsen, Djamila Aouada, Aleksandar Stojanovic, and Björn E. Ottersten. 2016. Feature engineering strategies for credit card fraud detection. Expert Syst. Appl. 51 (2016), 134–142. https://doi.org/10.1016/j.eswa.2015.12.030
[5] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. 2021. Recent Advances in Adversarial Training for Adversarial Robustness. , 4312–4321 pages. https://doi.org/10.24963/ijcai.2021/591
[6] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can machine learning be secure?. In Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, Taipei, Taiwan, March 21-24, 2006, Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Paul Lin, Shiuhpyng Shieh, and Sushil Jajodia (Eds.). ACM, 16–25. https://doi.org/10.1145/1128817.1128824
[7] A. Bekirev, V. Klimov, M. Kuzin, and B. Shchukin. 2015. Payment card fraud detection using neural network committee and clustering. Optical Memory and Neural Networks 24 (07 2015), 193–200. https://doi.org/10.3103/S1060992X15030030
[8] Siddhartha Bhattacharyya, Sanjeev Jha, Kurian K. Tharakunnel, and J. Christopher Westland. 2011. Data mining for credit card fraud: A comparative study. Decis. Support Syst. 50, 3 (2011), 602–613. https://doi.org/10.1016/j.dss.2010.08.008
[9] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 8190), Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný (Eds.). Springer, 387–402. https://doi.org/10.1007/978-3-642-40994-3_25
[10] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2017. Security Evaluation of Pattern Classifiers under Attack. CoRR abs/1709.00609 (2017). arXiv:1709.00609 http://arxiv.org/abs/1709.00609
[11] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning Attacks against Support Vector Machines. In Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012. icml.cc / Omnipress. http://icml.cc/2012/papers/880.pdf
[12] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognit. 84 (2018), 317–331. https://doi.org/10.1016/j.patcog.2018.07.023
[13] Rüdiger W. Brause, Timm Sebastian Langsdorf, and Hans-Michael Hepp. 1999. Neural Data Mining for Credit Card Fraud Detection. In 11th IEEE International Conference on Tools with Artificial Intelligence, ICTAI '99, Chicago, Illinois, USA, November 8-10, 1999. IEEE Computer Society, 103–106. https://doi.org/10.1109/TAI.1999.809773
[14] Michele Carminati, Alessandro Baggio, Federico Maggi, Umberto Spagnolini, and Stefano Zanero. 2018. FraudBuster: Temporal Analysis and Detection of Advanced Financial Frauds. In Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Saclay, France, June 28-29, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10885), Cristiano Giuffrida, Sébastien Bardin, and Gregory Blanc (Eds.). Springer, 211–233. https://doi.org/10.1007/978-3-319-93411-2_10
[15] Michele Carminati, Roberto Caron, Federico Maggi, Ilenia Epifani, and Stefano Zanero. 2015. BankSealer: A decision support system for online banking fraud analysis and investigation. Comput. Secur. 53 (2015), 175–186. https://doi.org/10.1016/j.cose.2015.04.002
[16] Michele Carminati, Mario Polino, Andrea Continella, Andrea Lanzi, Federico Maggi, and Stefano Zanero. 2018. Security Evaluation of a Banking Fraud Analysis System. ACM Trans. Priv. Secur. 21, 3 (2018), 11:1–11:31. https://doi.org/10.1145/3178370
[17] Michele Carminati, Luca Santini, Mario Polino, and Stefano Zanero. 2020. Evasion Attacks against Banking Fraud Detection Systems. In 23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2020, San Sebastian, Spain, October 14-15, 2020, Manuel Egele and Leyla Bilge (Eds.). USENIX Association, 285–300. https://www.usenix.org/conference/raid2020/presentation/carminati
[18] Michele Carminati, Luca Valentini, and Stefano Zanero. 2017. A Supervised Auto-Tuning Approach for a Banking Fraud Detection System. In Cyber Security Cryptography and Machine Learning - First International Conference, CSCML 2017, Beer-Sheva, Israel, June 29-30, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10332), Shlomi Dolev and Sachin Lodha (Eds.). Springer, 215–233. https://doi.org/10.1007/978-3-319-60080-2_17
[19] Francesco Cartella, Orlando Anunciação, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. 2021. Adversarial Attacks for Tabular Data: Application to Fraud Detection and Imbalanced Data. In Proceedings of the Workshop on Artificial Intelligence Safety 2021 (SafeAI 2021) co-located with the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021), Virtual, February 8, 2021 (CEUR Workshop Proceedings, Vol. 2808), Huáscar Espinoza, John Alexander McDermid, Xiaowei Huang, Mauricio Castillo-Effen, Xin Cynthia Chen, José Hernández-Orallo, Seán Ó hÉigeartaigh, and Richard Mallah (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2808/Paper_4.pdf
[20] Jipeng Cui, Chungang Yan, and Cheng Wang. 2021. ReMEMBeR: Ranking Metric Embedding-Based Multicontextual Behavior Profiling for Online Banking Fraud Detection. IEEE Trans. Comput. Soc. Syst. 8, 3 (2021), 643–654. https://doi.org/10.1109/TCSS.2021.3052950
[21] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In 28th USENIX Security

Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 321–338. https://www.usenix.org/conference/usenixsecurity19/presentation/demontis

[22] Li Deng and Xiao Li. 2013. Machine Learning Paradigms for Speech Recognition: An Overview. IEEE Trans. Speech Audio Process. 21, 5 (2013), 1060–1089. https://doi.org/10.1109/TASL.2013.2244083

[23] Rachna Dhamija, J. D. Tygar, and Marti A. Hearst. 2006. Why phishing works. In Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006, Rebecca E. Grinter, Tom Rodden, Paul M. Aoki, Edward Cutrell, Robin Jeffries, and Gary M. Olson (Eds.). ACM, 581–590. https://doi.org/10.1145/1124772.1124861

[24] Jonas Geiping, Liam Fowl, Gowthami Somepalli, Micah Goldblum, Michael Moeller, and Tom Goldstein. 2021. What Doesn't Kill You Makes You Robust(er): Adversarial Training against Poisons and Backdoors. arXiv:2102.13624 https://arxiv.org/abs/2102.13624

[25] Miguel Gomez. 2020. Dark Web Price Index 2020. https://www.privacyaffairs.com/dark-web-price-index-2020/

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. (2015). http://arxiv.org/abs/1412.6572

[27] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick D. McDaniel. 2017. On the (Statistical) Detection of Adversarial Examples. CoRR abs/1702.06280 (2017). arXiv:1702.06280 http://arxiv.org/abs/1702.06280

[28] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar. 2011. Adversarial machine learning. In Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, Chicago, IL, USA, October 21, 2011, Yan Chen, Alvaro A. Cárdenas, Rachel Greenstadt, and Benjamin I. P. Rubinstein (Eds.). ACM, 43–58. https://doi.org/10.1145/2046684.2046692

[29] Fayaz Itoo, Satwinder Singh, et al. 2021. Comparison and analysis of logistic regression, Naïve Bayes and KNN machine learning algorithms for credit card fraud detection. International Journal of Information Technology 13, 4 (2021), 1503–1511.

[30] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. IEEE Computer Society, 19–35. https://doi.org/10.1109/SP.2018.00057

[31] Sanjeev Jha, Montserrat Guillen, and J. Christopher Westland. 2012. Employing transaction aggregation strategy to detect credit card fraud. Expert Syst. Appl. 39, 16 (2012), 12650–12657. https://doi.org/10.1016/j.eswa.2012.05.018

[32] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. 2021. Intrinsic Certified Robustness of Bagging against Data Poisoning Attacks. In Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. AAAI Press, 7961–7969. https://ojs.aaai.org/index.php/AAAI/article/view/16971

[33] KPMG. 2019. Global Banking Fraud Survey. https://assets.kpmg/content/dam/kpmg/xx/pdf/2019/05/global-banking-fraud-survey.pdf

[34] Seeja K.R. and Masoumeh Zareapoor. 2014. FraudMiner: A Novel Credit Card Fraud Detection Model Based on Frequent Itemset Mining. TheScientificWorldJournal 2014 (09 2014), 252797. https://doi.org/10.1155/2014/252797

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (2017), 84–90. https://doi.org/10.1145/3065386

[36] Katharina Krombholz, Heidelinde Hobel, Markus Huber, and Edgar R. Weippl. 2015. Advanced social engineering attacks. J. Inf. Secur. Appl. 22 (2015), 113–122. https://doi.org/10.1016/j.jisa.2014.09.005

[37] Danilo Labanca, Luca Primerano, Marcus Markland-Montgomery, Mario Polino, Michele Carminati, and Stefano Zanero. 2022. Amaretto: An Active Learning Framework for Money Laundering Detection. IEEE Access 10 (2022), 41720–41739. https://doi.org/10.1109/ACCESS.2022.3167699

[38] Qiang Liu, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor C. M. Leung. 2018. A Survey on Security Threats and Defensive Techniques of Machine Learning: A Data Driven View. IEEE Access 6 (2018), 12103–12117. https://doi.org/10.1109/ACCESS.2018.2805680

[39] Zihao Liu, Qi Liu, Tao Liu, Nuo Xu, Xue Lin, Yanzhi Wang, and Wujie Wen. 2019. Feature Distillation: DNN-Oriented JPEG Compression Against Adversarial Examples. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. Computer Vision Foundation / IEEE, 860–868. https://doi.org/10.1109/CVPR.2019.00095

[40] Marc Moreno Lopez and Jugal Kalita. 2017. Deep Learning applied to NLP. CoRR abs/1703.03091 (2017). arXiv:1703.03091 http://arxiv.org/abs/1703.03091

[41] Samuel Marchal and Sebastian Szyller. 2019. Detecting organized eCommerce fraud using scalable categorical clustering. In Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019, David Balenson (Ed.). ACM, 215–228. https://doi.org/10.1145/3359789.3359810

[42] Italian Ministry of Economy and Finance. 2021. Statistical reports on payment card fraud (italian version). Technical Report. https://www.dt.mef.gov.it/export/sites/sitodt/modules/documenti_it/antifrode_mezzi_pagamento/antifrode_mezzi_pagamento/Rapporto-statistico-sulle-frodi-con-le-carte-di-pagamento-edizione-2021.pdf

[43] Raghavendra Patidar, Lokesh Sharma, et al. 2011. Credit card fraud detection using neural network. International Journal of Soft Computing and Engineering (IJSCE) 1, 32-38 (2011).

[44] Andrea Paudice, Luis Muñoz-González, András György, and Emil C. Lupu. 2018. Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection. arXiv:1802.03041 http://arxiv.org/abs/1802.03041

[45] Andrea Paudice, Luis Muñoz-González, and Emil C. Lupu. 2018. Label Sanitization Against Label Flipping Poisoning Attacks. In ECML PKDD 2018 Workshops - Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11329), Carlos Alzate, Anna Monreale, Haytham Assem, Albert Bifet, Teodora Sandra Buda, Bora Caglayan, Brett Drury, Eva García-Martín, Ricard Gavaldà, Stefan Kramer, Niklas Lavesson, Michael Madden, Ian M. Molloy, Maria-Irina Nicolae, and Mathieu Sinn (Eds.). Springer, 5–15. https://doi.org/10.1007/978-3-030-13453-2_1

[46] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020. IEEE, 1332–1349. https://doi.org/10.1109/SP40000.2020.00073

[47] Javier Fernández Rodríguez, Michele Papale, Michele Carminati, and Stefano Zanero. 2022. A Natural Language Processing Approach for Financial Fraud Detection. In Proceedings of the Italian Conference on Cybersecurity (ITASEC 2022), Rome, Italy, June 20-23, 2022 (CEUR Workshop Proceedings, Vol. 3260), Camil Demetrescu and Alessandro Mei (Eds.). CEUR-WS.org, 135–149. http://ceur-ws.org/Vol-3260/paper10.pdf

[48] Y Sahin and Ekrem Duman. 2010. Detecting credit card fraud by decision trees and support vector machines. In World Congress on Engineering 2012. July 4-6, 2012. London, UK., Vol. 2188. International Association of Engineers, 442–447.

[49] Taeshik Shon and Jongsub Moon. 2007. A hybrid machine learning approach to network anomaly detection. Inf. Sci. 177, 18 (2007), 3799–3821. https://doi.org/10.1016/j.ins.2007.03.025

[50] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. 2017. Certified Defenses for Data Poisoning Attacks. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 3517–3529. https://proceedings.neurips.cc/paper/2017/hash/9d7311ba459f9e45ed746755a32dcd11-Abstract.html

[51] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daumé III, and Tudor Dumitras. 2018. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1299–1316. https://www.usenix.org/conference/usenixsecurity18/presentation/suciu

[52] Mahito Sugiyama and Karsten M. Borgwardt. 2013. Rapid Distance-Based Outlier Detection via Sampling. (2013), 467–475. https://proceedings.neurips.cc/paper/2013/hash/d296c101daa88a51f6ca8cfc1ac79b50-Abstract.html

[53] Symantec. 2017. ISTR - Financial Threats Review 2017. https://docs.broadcom.com/doc/istr-financial-threats-review-2017-en

[54] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1312.6199

[55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. (2014). http://arxiv.org/abs/1312.6199

[56] Florian Tramèr, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. 2020. On Adaptive Attacks to Adversarial Example Defenses. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/11f38f8ecd71867b42433548d1078e38-Abstract.html

[57] Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias, and Ke Li. 2016. AIˆ2: Training a Big Data Machine to Defend. In 2nd IEEE International Conference on Big Data Security on Cloud, BigDataSecurity 2016, IEEE International Conference on High Performance and Smart Computing, HPSC 2016, and IEEE International Conference on Intelligent Data and Security, IDS 2016, New York, NY, USA, April 9-10, 2016. IEEE, 49–54. https://doi.org/10.1109/BigDataSecurity-HPSC-IDS.2016.79

[58] Christopher Whitrow, David J. Hand, Piotr Juszczak, David John Weston, and Niall M. Adams. 2009. Transaction aggregation as a strategy for credit card fraud detection. Data Min. Knowl. Discov. 18, 1 (2009), 30–55. https://doi.org/10.1007/s10618-008-0116-z

[59] Shiyang Xuan, Guanjun Liu, Zhenchuan Li, Lutao Zheng, Shuo Wang, and Changjun Jiang. 2018. Random forest for credit card fraud detection. In 15th IEEE International Conference on Networking, Sensing and Control, ICNSC 2018, Zhuhai, China, March 27-29, 2018. IEEE, 1–6. https://doi.org/10.1109/ICNSC.2018.8361343

[60] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial Examples: Attacks and Defenses for Deep Learning. IEEE Trans. Neural Networks Learn. Syst. 30, 9 (2019), 2805–2824. https://doi.org/10.1109/TNNLS.2018.2886017

[61] Yixuan Zhang, Jialiang Tong, Ziyi Wang, and Fengqiang Gao. 2020. Customer Transaction Fraud Detection Using Xgboost Model. In 2020 International Conference on Computer Engineering and Application (ICCEA). 554–558. https://doi.org/10.1109/ICCEA50009.2020.00122