IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 November 2022, accepted 10 December 2022, date of publication 14 December 2022,
date of current version 20 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3229124

RESEARCH ARTICLE

Turning Federated Learning Systems Into Covert Channels

GABRIELE COSTA[1], (Member, IEEE), FABIO PINELLI[1], SIMONE SODERI[1], (Senior Member, IEEE), AND GABRIELE TOLOMEI[2]

[1]SySMA Unit, IMT School for Advanced Studies, 55100 Lucca, Italy
[2]Department of Computer Science, Sapienza University of Rome, 00185 Rome, Italy

Corresponding author: Gabriele Costa (gabriele.costa@imtlucca.it)

This work was supported by the EU H2020 Project "SPARTA" under Grant 830892.

ABSTRACT Federated learning (FL) goes beyond traditional, centralized machine learning by distributing model training among a large collection of edge clients. These clients cooperatively train a global, e.g., cloud-hosted, model without disclosing their local, private training data. The global model is then shared among all the participants which use it for local predictions. This paper proves that FL systems can be turned into covert channels to implement a stealth communication infrastructure. The main intuition is that, during federated training, a malicious sender can poison the global model by submitting purposely crafted examples. Although the effect of the model poisoning is negligible to other participants and does not alter the overall model performance, it can be observed by a malicious receiver and used to transmit a sequence of bits. We mounted our attack on an FL system to verify its feasibility. Experimental evidence shows that this covert channel is reliable, efficient, and extremely hard to counter. These results highlight that our new attacker model threatens FL infrastructures.

INDEX TERMS Federated learning, adversarial attacks, machine learning security, covert channel.

I. INTRODUCTION

Federated learning (FL) [48], [49] has emerged as the leading technology for implementing distributed, large scale and efficient machine learning (ML) infrastructures. The main idea is that multiple clients connect to the FL system, and collaboratively train a shared, global model. Frequently FL networks consist of a centralized, e.g., cloud-hosted, server and many edge clients that iteratively run FL rounds. Each round consists of the following steps.

1) The server sends the current, global model to the clients and appoints some of them for training.
2) Each selected client locally trains its copy of the global model with its own private data. Then it sends the resulting local model back to the server.
3) The server updates the global model by applying an *aggregation function* to the local models of the clients.

The associate editor coordinating the review of this manuscript and approving it for publication was Biju Issac.

FL allows clients to concurrently train a shared global model, without disclosing private training data. Hence, FL provides great benefits in terms of both scalability and privacy. Since this process smoothly integrates with ubiquitous, distributed infrastructures, it has been applied to IoT [77], Fog computing [84], autonomous vehicles [61], smartphones [81], and wearable devices [15]. Thus, nowadays, billions of devices are connected to one or more FL systems.

The growing adoption of FL also raises security concerns, for instance, about the confidentiality, integrity, and availability of FL systems. As a consequence, several authors considered attack scenarios such as *data poisoning*, where an adversary pollutes the training set with maliciously crafted examples [32], and *model poisoning*, in which the attacker directly attempts to tamper with the global model parameters [6]. Also, a large body of work deals with privacy leakage that may expose the local data of some clients [50]. However, very little research has been done for studying the emerging

exploitation opportunities, i.e., new attacks carried out *by means of* FL systems.

In this paper, we discuss a recent attack scenario that we originally reported in [16]. Briefly, it consists of an adversary implementing a *covert channel* [37] over an FL system. Covert channels allow an attacker to establish illicit communication between two agents (e.g., two devices) that should stay isolated. In theory, since no trust relationship exists among the clients, FL should not be intended to support the creation of covert channels. In practice, being shared among the participants, the global model can be turned into a communication channel. More specifically, two FL clients, i.e., a sender and a receiver, can agree on an aimed poisoning strategy that allows them to transfer one bit. In this way, they exploit the global model updates as a physical communication medium.

We start by describing our attacker model and the covert channel implementation strategy. Our attacker only requires limited capabilities and, thus, it appears very realistic. As a matter of fact, communications are established by poisoning the training set of a single client, i.e., the sender. Poisoned examples are crafted by modifying benign input examples through simple, effective and efficient heuristics.

We show how to implement our attack on an FL system, where clients collaboratively train a global model to recognize handwritten digits of the popular MNIST dataset [39]. In our FL system, private training data is represented by a subset of the MNIST dataset that is randomly assigned to each client. Such an implementation is often given as a template of a generic FL system in official tutorials.[1] Also, we demonstrate the feasibility on another image recognition task, i.e., for the CIFAR-10 dataset [36].

Our experiments highlight that FL-based covert channels are an actual threat that, to the best of our knowledge, has been neglected so far. Moreover, we show that channel performance in terms of capacity and quality can support real communications. Since parallel covert channels can exist in a single FL system, the channel bandwidth can also scale up. Finally, experiments confirm that covert channels implemented in this way are hard to detect and counter.

The main contributions of this paper are listed below.

- A novel attacker model for FL-based covert channel.
- A general attack implementation strategy.
- A prototype applied to image classification tasks.
- Experiments on the performance of the channels.
- A discussion of possible mitigation mechanisms.

The rest of the paper is organized as follows. Section II describes the main background concepts used in this work. In Section III, we revise the literature about FL security and application-level covert channels. Section IV introduces our attacker model, and Section V details the implementation of the covert channel. In Section VI, we describe the properties of our covert channel. Section VII presents our experiments. In Section VIII, we discuss detectability, countermeasures, and exploitability. Finally, Section IX concludes the paper.

[1]For example, see https://www.tensorflow.org/federated/tutorials/federated_learning_for_image_classification

II. BACKGROUND

In this section, we provide the reader with the essential context needed to understand the subject of this work.

A. MACHINE AND FEDERATED LEARNING

We consider the *supervised learning* task as the reference example of a typical ML problem. The goal of supervised learning is to estimate a function that maps an input to an output, based on a sample of observed input-output pairs, called *examples*, which is usually referred to as *training set*.

More formally, let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ be a training set of n examples. Each $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional vector of *features* representing the i-th input and $y_i \in \mathcal{Y}$ is its corresponding output value. Here, we focus on *classification* problems where $\mathcal{Y} = \{1, \ldots, \ell\}$ and each y_i is known as the *class label* (as opposed to regression problems where $\mathcal{Y} \subseteq \mathbb{R}$).

Supervised learning assumes the existence of an unknown target function $g : \mathcal{X} \mapsto \mathcal{Y}$ that maps any feature vector to its corresponding output. The goal is therefore to estimate a function m^*, namely a parametric model, that best approximates g on $\mathcal{D}$. More specifically, the optimal parametric model m^* is the one that minimizes the value of a loss function $\mathcal{L}$, which measures the cost of replacing the true g with m^* on the training set. In other words, learning m^* reduces to the following optimization problem, also known as empirical risk minimization (ERM) [74].

$$m^* = \text{argmin}_m \mathcal{L}(m, \mathcal{D}) \tag{1}$$

Depending on the supervised learning task, different loss functions can be adopted. For example, cross-entropy is commonly used for classification [52], whilst mean squared error is typically employed in regression settings [29].

The standard framework above assumes that the actual training procedure, i.e., the optimizer used to solve (1), runs on a centralized location where the whole dataset $\mathcal{D}$ is stored. In the case of FL, instead, the learning process is distributed among several clients that collaboratively train a shared, global model with a centralized server acting as an orchestrator. Thus, the FL framework consists of a centralized server S and a set of distributed, federated clients $\mathcal{C}$, such that $|\mathcal{C}| = n_c$. Each client $c \in \mathcal{C}$ has access to its own private training set $\mathcal{D}_c$, namely the set of its local labeled examples.

The generic t-th round of FL runs the following steps.

1) S sends the current, global model $m^{(t)}$ to every client and selects a subset $\mathcal{C}^{(t)} \subseteq \mathcal{C}$, such that $1 \leq |\mathcal{C}^{(t)}| \leq n_c$.
2) Each selected client $c \in \mathcal{C}^{(t)}$ trains its local model $m_c^{(t)}$ by optimizing the same objective of (1) on its own private data $\mathcal{D}_c$, starting from $m^{(t)}$; the resulting $m_c^{(t)}$ is sent to S.
3) S computes $m^{(t+1)} = \phi(\{m_c^{(t)} \mid c \in \mathcal{C}^{(t)}\})$ as the updated global model, where ϕ is an *aggregation function* (e.g., FedAvg [48] or one of its variants [43]).

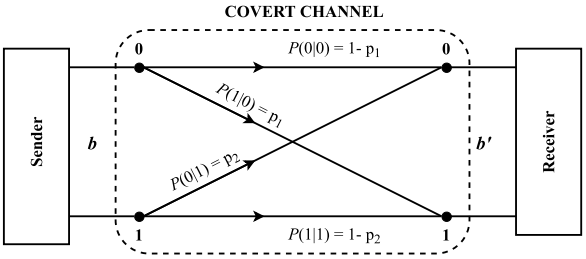

FIGURE 1. Binary memoryless covert channel.

In the beginning, $m^{(0)}$ may be randomly initialized. Then, FL rounds as the one described above are iteratively executed until convergence of the global model, i.e., until $t = T$ such that $m^{(T)} = m^*$. In practice, though, many FL models are continuously trained due to the highly dynamic nature of the infrastructure (e.g., new clients joining or leaving the system and fresh local data generated over time).

To simplify the notation, in the following we refer to m as the global model and to m_c as the local model of client c. Furthermore, we call $m(\boldsymbol{x}) = \hat{y} \in \mathcal{Y}$ (resp., $m_c(\boldsymbol{x}) = \hat{y}_c \in \mathcal{Y}$) the global (resp., local) model prediction on input $\boldsymbol{x}$.

B. CHANNELS AND COMMUNICATION QUALITY

In this paper, we are interested in *binary memoryless channels* (BMC) [47]. Briefly, an BMC has discrete input b and output b' such that $b, b' \in \{0, 1\}$. We describe the relationship between channel's input and output through the conditional probability $P(\mathbf{b}|\mathbf{b})$, where $\mathbf{b} = \{0, 1\}$. Figure 1 depicts a generic BMC in which $P(1|0) = p_1$ and $P(0|1) = p_2$ are the probabilities for input/output bit inversion errors. Thus, their complements give the probability of receiving the correct bit, e.g., $P(0|0) = 1 - p_1$. For a BMC, the *channel capacity* C is the maximum communication rate that the sender and receiver can reach over the channel. Following [47], C is computed as

$$C = \max_{P} \mathbb{I}(x; y) = \sum_{\substack{x \in \mathbf{b} \\ y \in \mathbf{b}}} P(x)P(y|x) \log\left(\frac{P(y|x)}{P(y)}\right), \quad (2)$$

where $\mathbb{I}(x; y)$ is the mutual information between the input x and output y. From (2), we obtain the channel capacity as

$$\begin{aligned} C &= 1 + \sum_{i=1,2} \frac{1}{2}(p_i + (1 - p_i)\log_2(1 - p_i)) \\ &= 1 - \sum_{i=1,2} \frac{H(p_i)}{2} = 1 - \frac{H(p_1)}{2} - \frac{H(p_2)}{2}, \end{aligned} \quad (3)$$

where H is the binary Shannon entropy function.

Communication quality is typically measured in terms of *bit error rate* (BER) and *signal-to-noise ratio* (SNR). In general, BER is obtained as the number of bit inversions, e.g., due to channel noise, over the total number of transmitted bits. Instead, SNR is defined as the ratio of signal power to the noise power, i.e., s^2/n^2, where s is the channel signal and n is the noise. In general, when signal and noise are modeled by means of two random variables, called S and N respectively, we have that $s^2 = E[S^2]$ and $n^2 = E[N^2]$, where $E[\cdot]$ denotes the expected value. Moreover, when N has zero mean, i.e., $E[N] = 0$, n^2 reduces to $\sigma_n^2 = Var(N) = E[N^2] - E[N]^2$, that is $E[N^2]$ is equal to the variance of N.

III. RELATED WORK

The attacker model presented in this paper was originally put forward in our previous work [16]. Subsequently and independently, the very same attacker model was also reported in [30]. There the authors implement a spread spectrum overt channel via model poisoning. Their work confirms the relevance of our attacker model and outlines that there exist multiple exploitation techniques that real attackers can even combine.

To the best of our knowledge, our attacker model is not listed among the security challenges of FL, e.g., see [35]. In the following, we revise some related work about *adversarial attacks to ML*, and *application level covert channels* which are closer to our proposal.

A. ADVERSARIAL ATTACKS TO ML

There exists a large body of work investigating the security of both traditional, i.e., centralized, and federated ML against so-called *adversarial attacks* [9], [17], [25], [31], [33], [44], [53], [60], [72]. Adversarial attacks can have different targets. For instance, *model inversion attacks* [21], [22], *membership inference attacks* [50], [62], [66], and *property inference attacks* [2], [24] aim to violate the *confidentiality* of user's private training/test data. Also, an attacker can compromise the confidentiality/intellectual property of a model provider by stealing its model parameters and hyperparameters [41], [73], [75].

From this perspective, our attacker model belongs to adversarial attacks that tamper with the integrity and the performance of a predictive model [4]. These attacks are classified according to the stage(s) of the ML pipeline in which they occur. In particular, attacks can happen at training time only, both at training and at test time, or at test time only. Those are called *poisoning*, *backdoor*, and *evasion* attacks, respectively.

From another viewpoint, depending on the attacker's goal, adversarial attacks can be further classified as *untargeted* (or random) [8], [32], [40], [63], [78], [80] or *targeted* [1], [28], [54], [65]. The former aims to reduce the overall accuracy of the learned model at inference time, regardless of what specific testing examples get incorrectly classified. The latter forces the learned model to output attacker-desired labels for certain testing examples, e.g., predicting spam messages as non-spam, while not altering the output for other examples. Since targeted attacks have to do with a specific goal, they usually require the attacker to have rather strong capabilities.

In the following, we provide a detailed overview of the most prominent adversarial attacks.

1) POISONING ATTACKS

To compromise the performance of a predictive model, poisoning attacks can target two components of the training stage, i.e., the training dataset and the learning process. The former are known as *data poisoning* attacks [8], [20], [31], [32], [54]. The latter are referred to as *model poisoning* attacks [6], [19].

a: DATA POISONING

These attacks pollute the training dataset by injecting it with new malicious examples or by corrupting existing ones. There are two main types of data poisoning attacks, called *(i) clean-label* [65] and *(ii) dirty-label* [27], respectively. In clean-label poisoning attacks, the adversary has no control over the labeling process. The attacker simply injects a small number of slightly perturbed examples (whose labels remain correct) into the training set of the victim. These attacks have been proven effective only when the attacker has complete knowledge of the victim's model, i.e., under *white-box* assumption [60]. Such knowledge is needed to craft the malicious examples [69]. More recently, clean-label poisoning attacks for unknown, i.e., *black-box* [58], deep image classifiers have been explored [85].

In dirty-label poisoning, the adversary can introduce a number of training instances to be misclassified with a specific label in a targeted way. An example of dirty-label poisoning is the *label-flipping* attack [8], [23]. Here, the labels of honest training examples of one class are flipped to another class, while the features of the data are kept unchanged. For instance, a malicious agent can poison the training set of a handwritten digit recognition system by flipping all 1s to 7s.

In the FL setting, it is common to assume that the attacker controls some clients and their training sets. Thus, dirty-label attacks have been more often considered. In [72], the authors' study targeted label-flipping attacks on FL. They find that poisons injected late in the training process are significantly more effective than those injected early. Other proposals adopt a bi-level optimization approach for poisoning multi-task FL [70] and GAN-generated poisons [83].

b: MODEL POISONING

Different from data poisoning attacks, these attacks threaten the learning process directly, e.g., by changing some model parameters. Model poisoning is generally perceived as difficult to implement in centralized ML systems as it requires the adversary to access the target model, i.e., assuming either *grey-box* or *white-box* knowledge. On the other hand, model poisoning becomes rather feasible in the case of FL, where a malicious client has direct influence over the jointly-trained global model via its local parameters updates [6], [19], [45]. As with any poisoning attack, the adversary's goal is to cause wrong predictions of the FL model. However, she/he aims to force classification errors at inference time and without modifying test examples. In this respect, it is opposed to backdoor and evasion attacks (which are discussed below). In model poisoning, the misclassification results from the adversarial corruption of the training process, which can be achieved either by *gradient* or *learning rule* manipulation. In gradient manipulation, the adversary poisons local model gradients (or model updates), which are then sent to the central server for aggregation, thereby jeopardizing the global model performance [10]. In learning rule manipulation, instead, the attacker corrupts the actual training logic. In some cases, these model poisoning attacks have proven more effective than data poisoning, and an attacker can compromise the global model even when controlling a single client. For instance, in [6] the authors successfully achieve a stealthy targeted model poisoning attack by adding a penalty term to the objective function to minimize the distance between malicious and benign weight update distributions.

2) BACKDOOR ATTACKS

Backdoor attacks – also known as trojan attacks – exploit the adversary's capability of having (limited) access to input examples also at test time [25]. Hence, backdoor attacks exceed poisoning attacks, since the adversary can manipulate both training and test inputs. For this reason they are often considered more disruptive toward the victim model.

Like for standard poisoning attacks, we can distinguish between backdoor attacks affecting the data or the model. The former are referred to as backdoor data poisoning and consist of adding attacker-chosen examples to the training set. The attacker examples contain a particular *trigger* [14], [42], i.e., a distinguished feature that activates the backdoor. The model learned on such poisoned training set will embed a backdoor, which the attacker exploits at test time by submitting examples that contain the same trigger. Instead, backdoor model poisoning requires a stronger threat model, where the attacker can get direct access to the learning system and change the model's internals (i.e., parameters and architecture) to embed a backdoor [18], [34].

Interestingly enough, backdoor attacks have been proven ineffective under FL settings [3]. The main obstacle is that aggregation involves many clients and, assuming that the attacker only controls a minority, the effect of the adversarial updates is weakened. To overcome this limitation, the authors of [3] consider a model replacement approach, where the attacker scales up a malicious model update to increase its effect on the aggregation function.

In [79], the authors propose distributed backdoor attacks, which better exploit the decentralized nature of FL. Specifically, they decompose the backdoor pattern for the global model into multiple distributed small patterns, and inject them into training sets, used by up to 40% adversarial participants, at each round. Although more effective than global backdoor trigger injection, this approach comes at the price of controlling a significant subset of the FL clients.

3) EVASION ATTACKS

Adversarial attacks that occur only at test time are called evasion attacks. Here, the goal of the adversary is still to fool an ML model yet without tampering it at training time. In fact,

evasion attacks use so-called *adversarial examples* [4], [71], i.e., crafted (minimal) perturbations of test instances that cause prediction errors (either targeted or untargeted) when input to a legitimately trained model.

Evasion attacks may look similar to backdoor poisoning attacks [7]. However, the key difference between the two is that evasion attacks exploit the decision boundaries learned by an uncorrupted model to construct adversarial examples that are misclassified by the model. In contrast, backdoor attacks intentionally shift these decision boundaries as a result of a jeopardized training process, so that certain examples get eventually misclassified [25].

Several works have explored evasion attacks in the context of computer vision [11], where adversarial examples are obtained by adding random noise to test images. Even though such images look legitimate to a human, they are wrongly classified by the image recognition system. Also, more recent works investigate the applicability of evasion attacks to malware classification [68].

In the FL setting, the global model maintained by the server suffers from the same evasion attacks as in the conventional ML setting when the target model is deployed as a service. Moreover, at each FL training round the global model sent to the federated clients is exposed as a white-box to any malicious participant. Thus, FL requires extra efforts to defend against white-box evasion attacks [44].

In this work, we consider an attack scenario partially related to the one presented in [3]. As a matter of fact, our covert channel is implemented by means of maliciously crafted examples that carry a trigger as in backdoor attacks. In particular, our adversary (*i*) crafts malicious examples carrying a trigger (see Section V-A), and (*ii*) poisons the federated model to transmit one bit per trigger (see Section V-B).

B. APPLICATION LEVEL COVERT CHANNELS

In a general sense, a covert channel is any communication channel that is not intended for information transfer [37]. Although we are not aware of FL-based covert channels, some authors already investigated the implementation of covert channels at the application level. Most authors considered *encapsulation* of hidden communications in application-layer network protocols. Being the main application-level protocol, HTTP is the primary target for covert channel implementations, e.g., see [5]. Nevertheless, the entire TCP/IP ecosystem can be at risk, and we refer the interest reader to [51] for a survey. More recently, also web applications were proposed for the implementation of covert channels. For instance, in [64] the author considers social networks such as Facebook and Twitter. However, since these kinds of covert channels rely on already existing communications between devices, in practice, they usually do not break any sandbox policy.[2] Also, as the authors of [82] point out, most of these covert channels can be detected and some effective countermeasures exist, e.g., packet inspection can be used to detect illegal traffic. Possibly for these reasons, application-level covert channels are rare in the literature.

[2]On the contrary, they are very relevant, for instance, when considering inter-process channels as in [13].

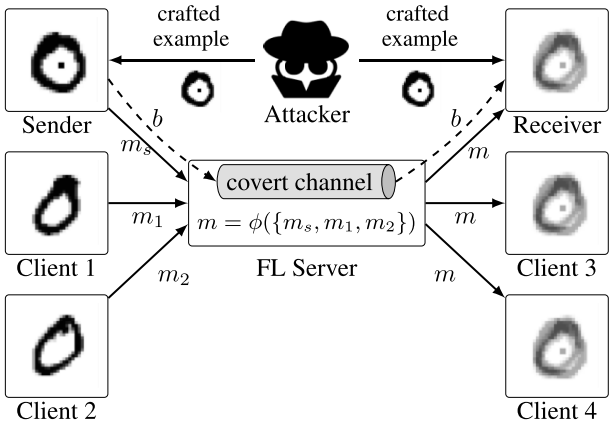

FIGURE 2. Overview of the attacker model.

Loosely speaking, also our proposal relies on a sort of encapsulation mechanism. However, here we do not wrap information inside protocol messages. Rather, we embed information inside FL models, which prevents standard detection techniques based on traffic inspection.

IV. ATTACKER MODEL

In this section, we present our attacker model. The attacker's goal is to establish a covert channel between two clients, namely *Sender* and *Receiver*, of an FL infrastructure. In terms of capabilities, our adversary resembles that of [3]. Here, we assume that both Sender and Receiver are controlled by the attacker. For instance, think of Sender as a malware-compromised device and the Receiver as the malware command and control server. Although they are compromised, we assume the attacker does not tamper with the standard FL infrastructure behavior, i.e., Sender and Receiver follow the FL client protocol. Furthermore, Sender and Receiver do not need to inspect nor jeopardize their local models to set up a covert channel. More precisely, Sender is only allowed to poison its local dataset and Receiver can only classify examples using its own local model. Intuitively, these assumptions hold for most FL systems.

The overall attacker model is schematically depicted in Figure 2. The FL server randomly selects a subset of clients at each federated round. Selected clients work as expected, i.e., they use their own private datasets to train their local models starting from the last global model received by the server (see Client 1 and Client 2 in Figure 2). Then, FL clients upload their newly trained local models to the server, aggregating them into an updated global model m through the aggregation function ϕ. At the end of each federated round, *all* the clients receive a copy of m. When selected, Sender (top left of Figure 2) poisons its local model m_s by training it with some malicious, attacker-provided examples. Its goal is to transmit a bit b by inducing a perturbation of the global model

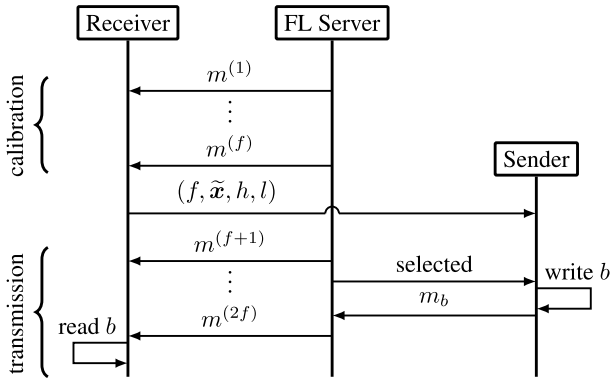

FIGURE 3. Communication protocol phases.

that the Receiver can test. On the other hand, Receiver (top right of Figure 2) uses m to classify test examples, e.g., the same malicious examples used by Sender. According to the classification outcome, the Receiver deduces whether 0 or 1 was sent. Implementing such a covert channel is non-trivial and depends on the underlying FL system. We discuss the implementation details in the next section.

V. COVERT CHANNEL IMPLEMENTATION

In this section, we detail the implementation strategy for creating the previously described covert channel. Without loss of generality, every implementation is based on the abstract protocol schematically depicted in Figure 3.

A transmission starts with a *calibration* phase during which Receiver observes the global model updates $m^{(1)}, \ldots, m^{(f)}$ (for f FL rounds). Eventually, Receiver computes *channel parameters* (see Section V-A) to be shared with Sender. For instance, these parameters can be provided through a secondary channel or hard-coded in Sender before its deployment. Then, Sender and Receiver synchronize on transmission frames of size f to send a bit b. During each frame, if selected, Sender trains its local model according to the channel parameters and the bit to be sent. In the meanwhile, Receiver monitors the global model updates and, at the end of the frame, it tests the received bit.

Below, we discuss the implementation of both the calibration and transmission phases.

A. CALIBRATION OF CHANNEL PARAMETERS

The first parameter to be determined is the size of the transmission frame f, i.e., the number of FL rounds used to transmit a single bit. Intuitively, too small values of f would increase transmission errors (e.g., Sender being never selected within a frame may result in a bit transmission error). On the opposite, if f is too large, channel throughput will be reduced. Finding optimal values of f is non-trivial as discussed in Section VII. Clearly, when the attacker knows the details of the target FL system, e.g., client selection probability (p_c), the desired value of f can be obtained analytically. For instance, when $p_c = 0.1$ and attacker wants Sender to be selected at least once with probability grater than 0.9, f is computed so that $1 - (1 - p_c)^f > 0.9$, i.e., $f = 22$.

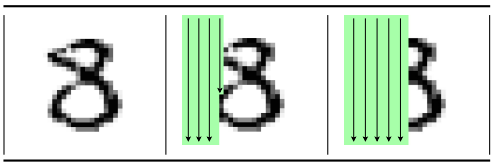

FIGURE 4. Linear transformation ψ_e of an example with $\alpha = 0.3$ (middle) and $\alpha = 0.5$ (rightmost).

Algorithm 1 Edge Example Binary Search Algorithm

Input: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \varepsilon > 0$
$\alpha_{\text{high}} := \text{H}$
$\alpha_{\text{low}} := \text{L}$
repeat
  $\boldsymbol{x}_{\text{high}} := \psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \alpha_{\text{high}})$
  $\hat{y}_{\text{high}} := m_r(\boldsymbol{x}_{\text{high}})$
  $\boldsymbol{x}_{\text{low}} := \psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \alpha_{\text{low}})$
  $\hat{y}_{\text{low}} := m_r(\boldsymbol{x}_{\text{low}})$
  if $* $ **then** $\hat{y}_{\text{high}} = \hat{y}_{\text{low}}$ **Failure**: cannot be equal
  $\alpha_{\text{mid}} := (\alpha_{\text{low}} + \alpha_{\text{high}})/2$
  $\boldsymbol{x}_{\text{mid}} := \psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \alpha_{\text{mid}})$
  $\hat{y}_{\text{mid}} := m_r(\boldsymbol{x}_{\text{mid}})$
  if $\hat{y}_{\text{high}} \neq \hat{y}_{\text{mid}}$ **then** $\alpha_{\text{low}} := \alpha_{\text{mid}}$ **else** $\alpha_{\text{high}} := \alpha_{\text{mid}}$
until $\alpha_{\text{low}} - \alpha_{\text{high}} < \varepsilon$
Output: $\widetilde{\boldsymbol{x}} := \boldsymbol{x}_{\text{high}}$, labels $h := \hat{y}_{\text{high}}$ and $l := \hat{y}_{\text{low}}$

Otherwise, Receiver must estimate f, e.g., by taking advantage of its selection notifications. For instance, it can set f as the number of rounds that it takes to be selected T times (for a constant T).

Another channel parameter to be determined is the crafted example $\widetilde{\boldsymbol{x}}$ carrying the trigger that Sender will use to poison its local training set. To generate $\widetilde{\boldsymbol{x}}$, Receiver applies a *linear transformation function*[3] ψ to a subset of k randomly selected examples from its local training set. Briefly, $\widetilde{\boldsymbol{x}} = \psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \alpha)$ means that, starting from $k > 0$ examples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$, ψ returns a new example $\widetilde{\boldsymbol{x}}$, where $\alpha \in [0, 1]$ is a parameter controlling the transformation.

To provide an intuition of this process, we put forward an example taken from the MNIST dataset. Consider $\boldsymbol{x}$ to be the representation of a generic MNIST input image, i.e., a 28×28 matrix of pixels flatten into a 784-dimensional vector. We define $\psi_e(\boldsymbol{x}, \alpha)$ as the function erasing, from left to right, a fraction α of a single example image $\boldsymbol{x}$ (i.e., $k = 1$). For instance, when $\alpha = 0.3$, ψ_e sets to 0 the 235 values associated with the leftmost pixels of the target image vector. More formally, $\psi_e(\boldsymbol{x}, \alpha) = \boldsymbol{x} - \boldsymbol{x}^{(\alpha)}$ where $\boldsymbol{x}^{(\alpha)} = [x^1, \ldots, x^{\lfloor \alpha \cdot 784 \rfloor}, 0, \ldots, 0]$ and x^i is the i-th element of $\boldsymbol{x}$. Figure 4 shows the behavior of ψ_e, where we highlighted the erased part of the example.

Intuitively, the attacker can select a function ψ based on some *semantic* property of the classification domain. For instance, in the previous example, ψ_e encodes the simple fact

[3]Although we do not explicitly prove it, the reader can easily check that all the transformation functions presented in the following are linear since they reduce to finite sums of matrices.

Algorithm 2 Sender Transmission Algorithm

Input: $f, \widetilde{\boldsymbol{x}}, h, l$
repeat
  if $r = 1$ **then**
    $b :=$ nextBit()
    $v := m(\widetilde{\boldsymbol{x}})$
    $v_\neg :=$ **if** $v = h$ **then** l **else** h
  if *selected by server for training* **then**
    $v_r := m(\widetilde{\boldsymbol{x}})$
    switch b **do**
      case *0* **do if** $v_r \neq v$ **then** train$(m_s, \widetilde{\boldsymbol{x}}, v)$
      case *1* **do if** $v_r \neq v_\neg$ **then** train$(m_s, \widetilde{\boldsymbol{x}}, v_\neg)$
    upload(m_s)
  $r := (r \% f) + 1$
until *transmission completed*

TABLE 1. Sender's model poisoning cases.

$b = 0$		$b = 1$	
$v_r = v$	$v_r = v_\neg$	$v_r = v$	$v_r = v_\neg$
do nothing	train$(m_s, \widetilde{\boldsymbol{x}}, v)$	train$(m_s, \widetilde{\boldsymbol{x}}, v_\neg)$	do nothing

Algorithm 3 Receiver Bit Test Algorithm

Input: $f, \widetilde{\boldsymbol{x}}, h, l$
repeat
  if $r = 1$ **then**
    $v_1 := m(\widetilde{\boldsymbol{x}})$
  else if $r = f$ **then**
    $v_f := m(\widetilde{\boldsymbol{x}})$
    $b :=$ **if** $v_1 = v_f$ **then** 0 **else** 1
    received(b)
  $r := (r \% f) + 1$
until *transmission completed*

that the right half of a handwritten 8 looks like a 3. In the experiments of Section VII we consider two slightly different functions. Others can be found, e.g., in [12].

Receiver repeatedly applies ψ to the selected examples until it identifies an *edge* example $\widetilde{\boldsymbol{x}}$. Formally, given Receiver's local model m_r and an arbitrarily small $\varepsilon > 0$, we look for $\widetilde{\boldsymbol{x}} = \psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \bar{\alpha})$ such that $m_r(\psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \bar{\alpha})) = h$ and $m_r(\psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \bar{\alpha} + \varepsilon)) = l$, where $h, l \in \mathcal{Y}$ and $l \neq h$. Interestingly, given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$, Receiver can efficiently compute $\bar{\alpha}$ via binary search, as sketched in Algorithm 1.

The search procedure starts from the predefined interval [H, L].[4] At each iteration, two examples, i.e., $\boldsymbol{x}_{\text{high}}$ and $\boldsymbol{x}_{\text{low}}$, are generated and classified with Receiver's model m_r, so obtaining $\hat{y}_{\text{high}}$ and $\hat{y}_{\text{low}}$, respectively. If $\hat{y}_{\text{high}} = \hat{y}_{\text{low}}$, the algorithm terminates with a failure and no edge example is returned. Otherwise, the current search interval is split in half by computing $\alpha_{\text{mid}}, \boldsymbol{x}_{\text{mid}}$, and $\hat{y}_{\text{mid}}$. Then, if $\hat{y}_{\text{high}} \neq \hat{y}_{\text{mid}}$, the algorithm iterates on the first half of the current interval. Otherwise, if $\hat{y}_{\text{low}} \neq \hat{y}_{\text{mid}}$, the search procedure continues on the second half. The loop terminates when the interval width goes under a threshold ε, which represents the granularity of a single position in the example feature vector, e.g., a single pixel in the case of Figure 4. Eventually, the algorithm returns the edge example $\boldsymbol{x}_{\text{high}}$, as well as the two class labels h and l associated with the last, smallest interval.

It is worth noticing that, since it does not explore the entire feature space, Algorithm 1 might fail to generate edge examples for some inputs. In general, the effectiveness of this heuristic method depends on the choice of ψ. We empirically show that Algorithm 1 can generate millions of edge examples in our experimental settings with only two linear transformation functions (see Section VII-A).

At the end of the calibration phase, channel parameters generated by Receiver amount to the tuple $(f, \widetilde{\boldsymbol{x}}, h, l)$.

[4] In our experiments we use H = 0 and L = 1/2.

B. BIT TRANSMISSION

The transmission of one bit is based on the variations, during f FL rounds, of $m(\widetilde{\boldsymbol{x}})$ between h and l. In particular, by poisoning its local model, Sender drives $m(\widetilde{\boldsymbol{x}})$ to assume the desired value, while Receiver monitors it to read the transmitted bit. Transmissions are organized in consecutive frames of size f. Sender and Receiver are synchronized through an FL round counter $r \in \{1, \ldots, f\}$. Sender follows the procedure given in Algorithm 2.

At the beginning of each transmission frame, i.e., when $r = 1$, Sender sets the next bit b, and uses the last received global model $m_r = m$ to classify $\widetilde{\boldsymbol{x}}$, thus obtaining $v \in \{h, l\}$. Also, Sender sets $v_\neg \in \{h, l\}$ so that $v \neq v_\neg$. At each round r, if selected by the FL server, Sender classifies $\widetilde{\boldsymbol{x}}$ with the global model m, so obtaining v_r. Then, depending on b, Sender trains its local model m_s according to Table 1.

Intuitively, the purpose of the operation above is to keep the label assigned to $\widetilde{\boldsymbol{x}}$ by the global model when sending 0 and to flip it when sending 1. This channel implementation amounts to a Differential Manchester encoding [47]. Eventually, Sender uploads its local model to the FL server.

Concurrently, Receiver executes Algorithm 3. Receiver uses the global model m to classify $\widetilde{\boldsymbol{x}}$ both when $r = 1$ and $r = f$, so obtaining v_1 and v_f, respectively. Finally, Receiver reads 0 if $v_1 = v_f$ and 1 otherwise.

In Figure 5, we show as an example the transmission of 10 bits for the MNIST scenario discussed above, using $\widetilde{\boldsymbol{x}}$ of Figure 4, $h = 8$, and $l = 3$. The top diagram shows the internal scores assigned by the global model to the edge example $\widetilde{\boldsymbol{x}}$ during the transmission. By classifying $\widetilde{\boldsymbol{x}}$, Receiver observes the alternation of labels 8 and 3 (center). Then, Receiver interprets it as the sequence of bits 0101001110.

VI. CHANNEL DESCRIPTION

For each edge example, our implementation provides a digital, broadcast channel supporting *half duplex*[5] transmissions.

[5] Intuitively, Sender and Receiver cannot transmit at the same time, but they can alternate their roles.

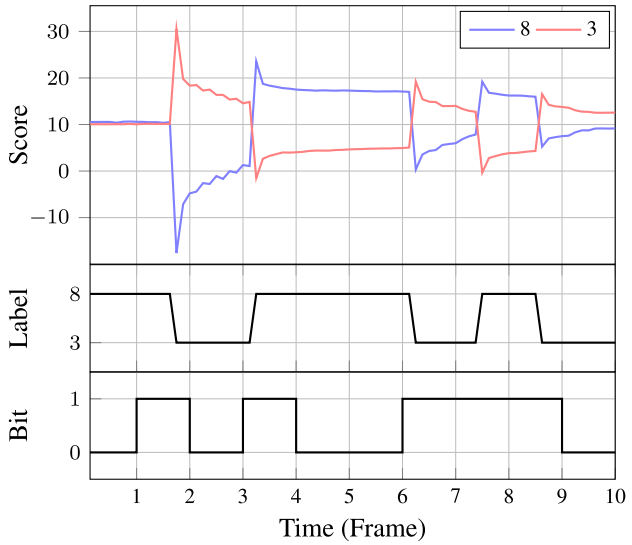

FIGURE 5. Transmission of bits 0101001110 over a channel.

In Section VII we will show that an attacker can even create several channels of this type to enlarge the communication bandwidth. Below, we detail the features of a single channel.

In terms of capacity, we treat the covert channel as a standard BMC (see, Section II). We just notice that the binary input of the channel is m_b, with $b \in \{0, 1\}$. Intuitively, inputs m_b represent Sender's local models m_s uploaded to the FL server. More precisely, we use m_b to distinguish between the two model poisoning cases used to transmit b (see Table 1).

In terms of communication quality, we consider BER and SNR, as discussed in Section II. While BER is straightforward, defining SNR in our context requires more attention. In general, SNR is computed by periodically sampling the channel signal s and noise n at the end of each transmission frame t. Here, defining s and n is non-trivial since our channel does not rely on a physical medium. Indeed, a covert channel consists of two prediction labels, namely h and l, and a perturbed example $\widetilde{\boldsymbol{x}}$. For each label $i \in \{h, l\}$, at time t we can measure the prediction score assigned by the global model to i when classifying $\widetilde{\boldsymbol{x}}$ at each frame's end. Also, assuming that prediction scores range within the interval $[Z, -Z]$,[6] we normalize the prediction scores by linearly scaling them in $[1, -1]$. Thus, for each label i, the label signal $z_i(t)$ amounts to the normalized score described above. We define the overall received signal $z(t) = z_h(t) - z_l(t)$, i.e., as the differential, normalized signal.

Since a direct measure of the transmitted signal $s(t)$ cannot be computed, we approximate it to the differential signal that switches its intensity between 1 and -1. Intuitively, this is equivalent to stating that Sender attempts to transmit 0 by

[6] Z can be estimated as the maximum score observed during a transmission.

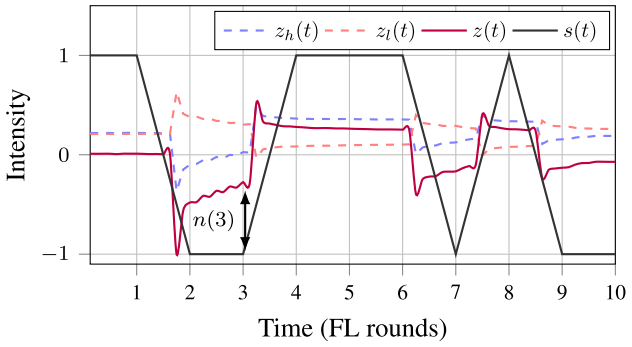

FIGURE 6. Signals and noise for the channel of Figure 5.

setting $z(t) \cdot z(t+1) = 1$ and 1 by setting $z(t) \cdot z(t+1) = -1$.[7] Thus, for each frame t, we set

$$s(t) = \frac{1 - 2b}{s(t-1)}, \tag{4}$$

where b is the bit transmitted during frame t, and, by construction, $s(0) = 1$. Then, we define the noise at time t as $n(t) = z(t) - s(t)$.

The overall intuition behind our definitions of $z(t)$, $s(t)$ and $n(t)$ is given in Figure 6. There, red and blue dashed lines denote z_h and z_l signals, i.e., the normalized version of the scores of Figure 5. Instead, the purple line denotes $z(t)$ and the gray line denotes $s(t)$. Also, the vertical arrow shows the value of $n(t)$ at $t = 3$. From this and (4), we define the SNR of the covert channel as

$$SNR = \frac{z(t)^2}{\sigma_n^2}, \tag{5}$$

where σ_n is the standard deviation of the *normalized* noise $\bar{n}(t)$ defined as $\bar{n}(t) = \frac{n(t)}{\max_{t'} |n(t')|}$.

VII. EXPERIMENTS

In this section, we present our experimental results. The implementation of our covert channel, called *FedExp*, is available at https://github.com/fpinell/sec_federated_learning.

A. EXPERIMENTAL SETTING

All the tests described in this section were executed on Docker containers running on a dedicated Intel® Xeon® Gold 5218 2.30GHz CPU with 64 GB of memory. The implementation is based on the popular ML framework PyTorch [59]. We implement an FL system for handwritten digit classification in our testing environment.

1) DATASET DESCRIPTION

The MNIST dataset [39] is one of the most popular datasets used as a benchmark for training and testing image classifiers. It contains a total of 70,000 greyscale images of handwritten,

[7] Notice that this assumption is more restrictive w.r.t. the actual implementation, since Receiver only requires $z(t) \cdot z(t+1) < 0$ and $z(t) \cdot z(t+1) > 0$ to read 0 and 1, respectively.

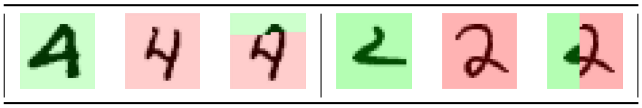

FIGURE 7. Applications of ψ_v to examples #22242 and #7596 (left) and ψ_h to examples #32481 and #18198 (right).

single digits. Digit images are taken from American Census Bureau employees and American high school students. Usually, 60,000 images are used for training and the remaining 10,000 for testing. Images are represented by 28×28 matrices of bytes, each byte representing a single pixel (where 0 is for background color, i.e., white, and 255 for the foreground color, i.e., black).

2) FL SYSTEM PARAMETERS

Our FL system is configured according to three parameters, i.e., the number of honest FL clients (n_c), the client selection probability (p_c), and the neural network architecture (τ).

At startup, both the training and the test portions of the full MNIST dataset are uniformly distributed randomly among the n_c honest clients. At each round, the server sends the current global model to all the clients; then, it randomly selects $\lfloor p_c \cdot n_c \rfloor$ clients. Thus, selected clients use their portion of the MNIST dataset to train their own local models. Local models are then sent back to the server, which aggregates them and updates the global model using the standard federated averaging function FedAvg [48].

The FL system we implemented supports two different types of neural network architectures. The first one is a fully connected Neural Network (NN) [38], composed of three layers with 200 neurons each. The second one is a Convolutional Neural Network (CNN) [39], [55] with two convolutional layers, each one of size 3×3, and one fully connected output layer. After each convolutional layer, we use a Max Pool 2D layer with kernel size 2×2. Both networks use a ReLU activation function and are trained by minimizing cross entropy loss function via stochastic gradient descent [26].

3) ATTACKER PARAMETERS

Receiver is added to the FL system after 200 training rounds. The frame size f can be manually configured. Otherwise, during the calibration phase, Receiver estimates f as the number of rounds it takes to select $T = 4$ times by the server. Another parameter is the number k of parallel covert channels to be established. Receiver generates k edge examples, one for each channel, to be used during the communication. Each edge example is generated by applying one of the following two linear transformation functions to the randomly selected MNIST examples $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

- $\psi_v(\boldsymbol{x}_1, \boldsymbol{x}_2, \alpha)$ combines the upper α fragment of $\boldsymbol{x}_1$ with the lower $(1-\alpha)$ fragment of $\boldsymbol{x}_2$.
- $\psi_h(\boldsymbol{x}_1, \boldsymbol{x}_2, \alpha)$ combines the leftmost α fragment of $\boldsymbol{x}_1$ with the rightmost $(1-\alpha)$ fragment of $\boldsymbol{x}_2$.

TABLE 2. Parameters of FL system (top) and attacker (bottom).

Par.	Description	Range	Example	Default
n_c	Number of honest clients	$\mathbb{N}$	10, 50	10
p_c	Client selection probability	$[0, 1]$	0.1, 0.5	0.5
τ	Neural network type	NN, CNN	NN	NN
f	Frame size	$\mathbb{N}$	6, 8	auto
k	Number of channels	$\mathbb{N}$	10, 20	1
n_b	Transmission length (bits)	$\mathbb{N}$	10, 100	10
w	Transmission pattern	$(0\|1)^*$	10, 101	auto

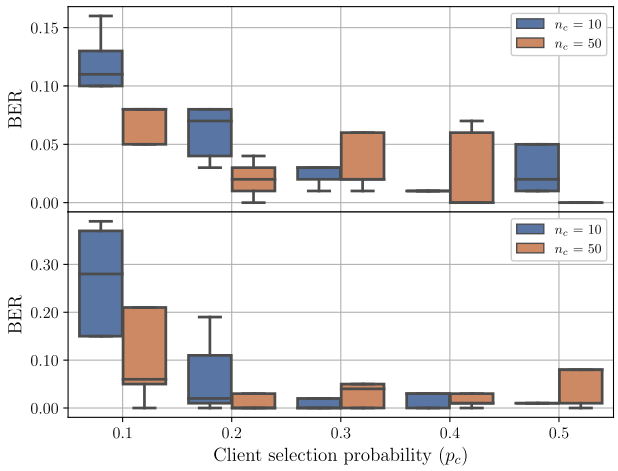

FIGURE 8. Channel BER for NN (top) and CNN (bottom).

Functions ψ_v and ψ_h resemble the example function ψ_e of Section V. Figure 7 shows two edge examples generated with Algorithm 1 when executing ψ_v and ψ_h on MNIST pairs (#22242, #7596) and (#32481, #18198), respectively. In the first case, the two original examples are both classified as 4, and the resulting edge example is classified between 9 and 4. In the second case, the two images are classified as 2, and the edge example is classified between 2 and 4.

For the generation of edge examples, we randomly selected pairs of pictures from the MNIST dataset, and applied Algorithm 1 with both ψ_h and ψ_v. Over 6,000 considered pairs, we obtained 101 edge examples. Thus, among all possible pairs of MNIST examples, approximately 1.7% might be used to create edge examples. This amounts to millions of possible channel implementations with ψ_v and ψ_h over the whole MNIST dataset.

When Receiver completes the calibration phase, Sender is configured with the generated channel parameters. Then, Sender joins the FL client network, consisting of $n_c + 2$ clients, and starts the transmission. Transmission parameters include n_b, i.e., the number of bits to be sent on each channel. By default, Sender automatically generates random bit sequences. Optionally, instead of random bits, one can specify a fixed bit pattern w. Finally, when selected, Sender trains its local model in the same way, e.g., using the same training epochs as all the other clients. The only difference resides in the training examples. That is, we assume Sender to have the same capabilities as honest clients.

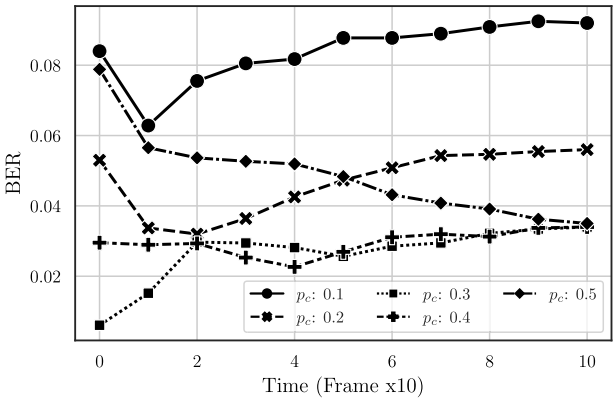

FIGURE 9. Average BER on NN for 10-bit transmission slots.

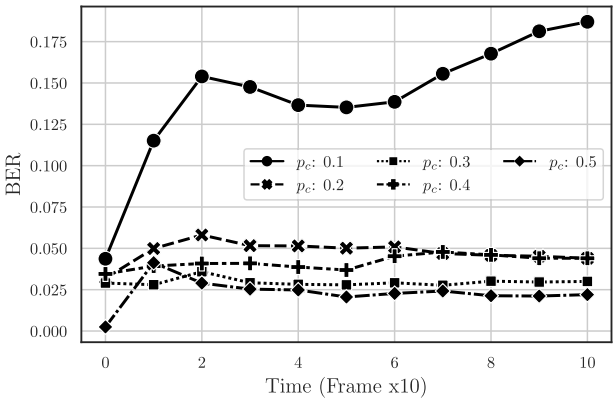

FIGURE 10. Average BER on CNN for 10-bit transmission slots.

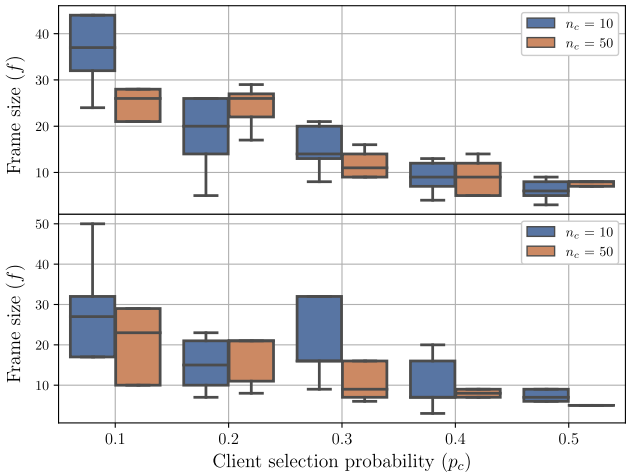

FIGURE 11. Frame length estimation on NN (top) and CNN (bottom).

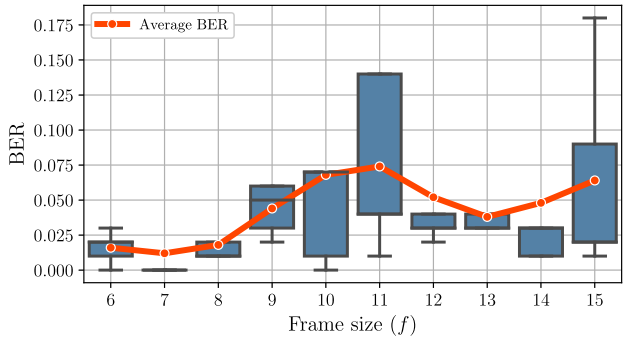

FIGURE 12. BER vs. frame size (τ = NN, $n_b = 100$, $n_c = 10$, $p_c = 0.5$).

Table 2 summarizes the experiment parameters for the FL system (top) and the attacker (bottom).

B. RESULTS

We assess channel quality in terms of BER and SNR. The box diagram[8] of Figures 8 shows single channel BER measured on NN (top) and CNN (bottom) under different settings. In particular, we consider $n_c \in \{10, 50\}$, $p_c \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, $n_b = 100$, and f estimated by Receiver as detailed in Section VII-A. Under the same settings described above, Figures 9 and 10 show the average BER for 10-bit transmission slots.

Results show that, when $p_c > 0.1$, BER tends to stay below 6% already with our simple heuristics for estimating f. Nevertheless, the attacker can achieve better performance by searching for optimal values of f (as discussed in Section V, BER is mainly affected by choice of f). Figures 11 shows the frame size estimated at calibration time for the NN (top) and CNN (bottom) experiments presented above. Instead, Figures 12 and 13, depict the cumulative BER for transmissions with increasing values of f, from 6 to 15. Interestingly enough, NN and CNN exhibit different behaviors, which confirms our expectation that finding the optimal value of f is non-trivial. In particular, under the same settings, CNN tends to behave monotonically, i.e., increasing f reduces BER, while NN does not exhibit e consistent behavior. The experiments suggest that the optimal frame size in the considered interval is $f = 7$ for NN and $f = 12$ for CNN.

Transmitted bit sequences may also impact communication quality. An example of BER for different transmission patterns is shown in Figure 14. There, we considered five different patterns of transmitted bytes, i.e., 00, 0F, 55, FF, and Random, for both NN and CNN. In both cases, we set f to the optimal values above, i.e., 7 and 12 for NN and CNN, respectively. As one might expect, the 00 pattern results in the minimum BER. The reason is that, since the FL system is well-trained, in most cases, Sender does not need to poison the global model at all. However, different patterns slightly affect BER in NN, which still stays below 4%, while have almost no effect on CNN. These results highlight that different types of networks may change the behavior of the channel. In general, finding the optimal channel parameter may require the attacker to carry out transmission tests, and this operation is allowed under our attacker model

The number of parallel covert channels an FL system can support is another crucial aspect, as it multiplies the total transmission bandwidth. Intuitively, we expect that an FL

[8]Each box represents 5 distinct experiments.

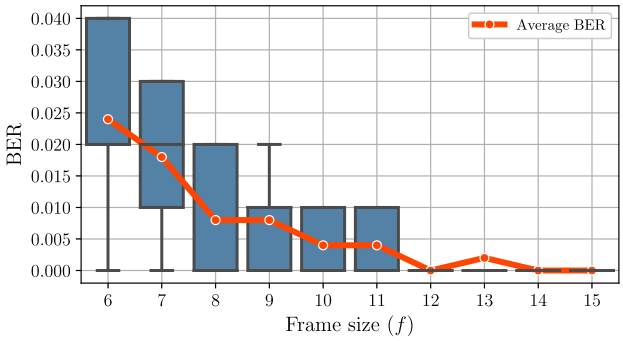

FIGURE 13. BER vs. frame size (τ = CNN, $n_b = 100$, $n_c = 10$, $p_c = 0.5$).

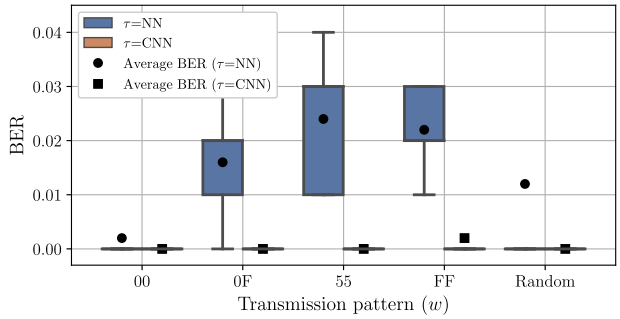

FIGURE 14. BER for patterns $w \in$ {00, 0F, 55, FF, Random} for both τ = NN and τ = CNN ($n_b = 100$, $n_c = 10$, $p_c = 0.5$, $f = 7$ and $f = 12$, for NN and CNN, respectively).

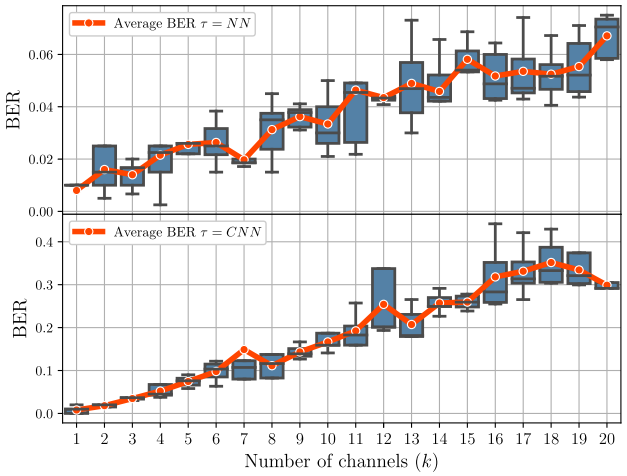

FIGURE 15. BER variation over the number of parallel covert channels k for NN and CNN (with $n_c = 10$, $p_c = 0.5$).

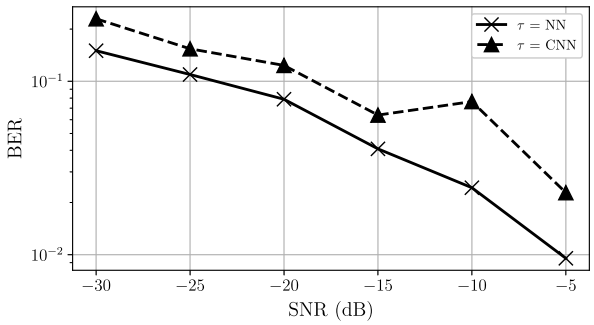

FIGURE 16. SNR versus BER for NN and CNN.

system can only host a finite number of channels before the model is saturated and the communication quality decays. Reasonably, we expect that saturation occurs first in simpler, shallow models. Figure 15 shows the increase of BER over 20 randomly generated channels in NN (top) and CNN (bottom), where the average BER increases with k.

We tested our definition of SNR (see Section VI) as an indicator of the covert channel quality. To this aim, we compared SNR versus BER and capacity. In both cases, for all the simulations, (*i*) we computed the SNR, and (*ii*) we binned SNR in 5 dB intervals. Then, we computed the average BER (capacity, respectively) for each bin. Figure 16 illustrates the BER when SNR increases, for both NN and CNN, under the settings presented above. Under the same settings, Figure 17 shows the relation between the SNR and channel capacity C. Our experiments show that higher values of SNR result in better communications, both in terms of lower BER and higher successful transmission rate. This trend follows the expected behavior of a channel in the classical information theory [47]. Thus, our experiments confirm that (5) is an appropriate definition of SNR.

Figure 18 shows the average BER of 5 tests transmitting 100 bits with a different number of clients, ranging from 100 to 500. The results confirm that average BER stays below 10% as it does with a lower number of clients.

To show the feasibility of our attack on other tasks, we also mounted it on a CIFAR-10 [36] FL classifiers. Such a classifier relies on a neural network implementing (a simplified version of) VGGNet [67]. As for MNIST, we tested our method by measuring the BER for 100 bits transmissions using different frame sizes (f). This experiment, reported in Figure 19, confirms that our covert channel attack is general w.r.t. the used dataset and classification algorithm.

VIII. DISCUSSION AND FUTURE WORK

The main goal of this paper is to prove the feasibility of a new, covert channel attack on FL systems. Our attack opens a number of research questions that are worth being investigated in the future. We briefly discuss them below.

A. CHANNEL THROUGHPUT

Our experiments highlight that covert channels can be instantiated on both NN and CNN networks, although NN provides better performances in terms of channel quality, transmission rate (f), and bandwidth. To better understand the actual throughput, consider a system where FL rounds occur every hour. If the attacker can implement 20 parallel covert channels with $f = 12$, a 128-bit key can be transmitted in $128 \cdot 12/20 = 76.8$ hours, i.e., less than 3 days. Although the transmission rate may seem low, there can be practical exploitation, e.g., by Advanced Persistent Threats. Also, our covert channel can be combined with others, thus implementing *hybrid*

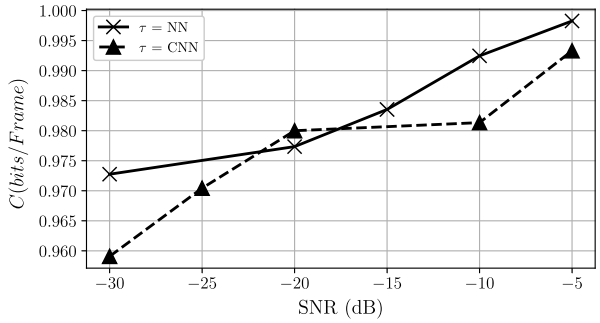

FIGURE 17. SNR versus channel capacity (C) for NN and CNN.

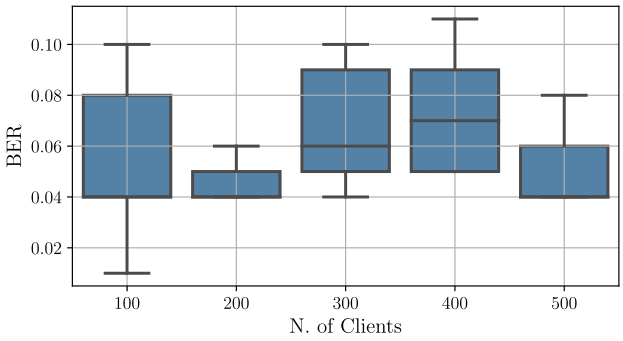

FIGURE 18. Average BER over 100 bits with 100, 200, 300, 400, and 500 clients using NN for the classification task.

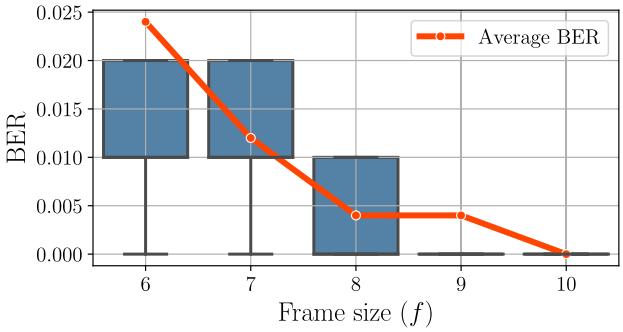

FIGURE 19. BER for CIFAR-10 dataset, 100 bits and different frame sizes.

covert channels. Further investigation is necessary to find techniques to maximize the channel performance.

B. IMPACT ON CLASSIFICATION ACCURACY

Although the goal of the attacker is not to compromise the accuracy of the classifier, the covert channel might degrade the performance of the FL system. We compared the accuracy measured by an honest client when Sender is not transmitting ($k = 0$) against that measured during transmissions with 1, 5 and, 10 channels. The results are shown in Figures 20 and 21.

Note that the honest client accuracy when $k = 0$ is stable. This happens because the network is well-trained by using the entire MNIST dataset. In real FL systems, this could not happen, since fresh training examples can appear over time.

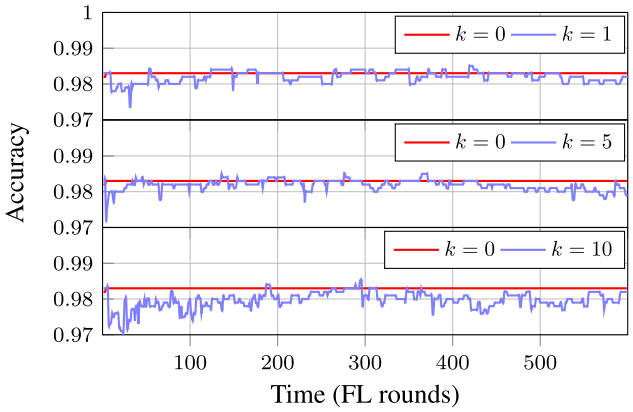

FIGURE 20. Accuracy for $k \in \{1, 5, 10\}$, τ = NN, $n_c = 10$, $p_c = 0.5$.

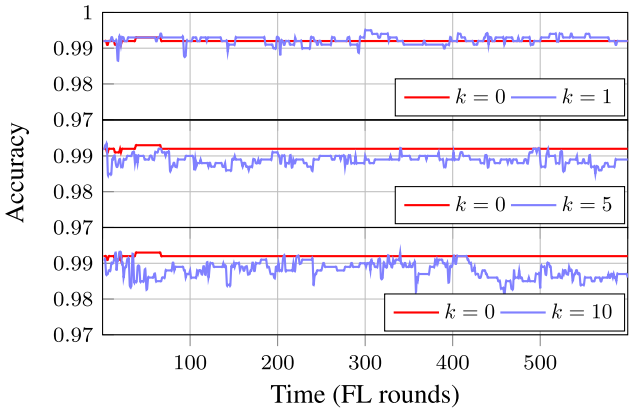

FIGURE 21. Accuracy for $k \in \{1, 5, 10\}$, τ = CNN, $n_c = 10$, $p_c = 0.5$.

For NN, when $k = 0$ the average accuracy is 0.983. When transmitting, the average accuracy slightly degrades, from 0.982 ($k = 1$) to 0.980 ($k = 10$). Similarly for CNN, the accuracy goes from 0.992 ($k = 1$) to 0.988 ($k = 10$). These results confirm that classification accuracy is minimally affected by transmissions. In larger FL systems, trained with fresh examples, accuracy is likely to be affected even less.

C. POSSIBLE MITIGATION MECHANISMS

Although the attacker model is new, some existing techniques might mitigate its effectiveness. For instance, anomaly detection techniques might identify the sender. Although designed for countering byzantine adversaries, *Krum* [10] aggregation function skips outlier models and could block or degrade the transmission. In our setting, the FL server can measure the distance between two models by using matrix norm. With L2-norm, the distance of a local model m_c w.r.t. the global model m is given by $d_c = \|m_c - m\|_2$.[9] Figure 22 shows the maximum d_c, i.e., the outlier, detected by the server during 200 FL rounds for both NN (top) and CNN (bottom).

[9]More precisely, this amounts to computing the L2-norm of the difference between the vectors of parameters of m_c and m.

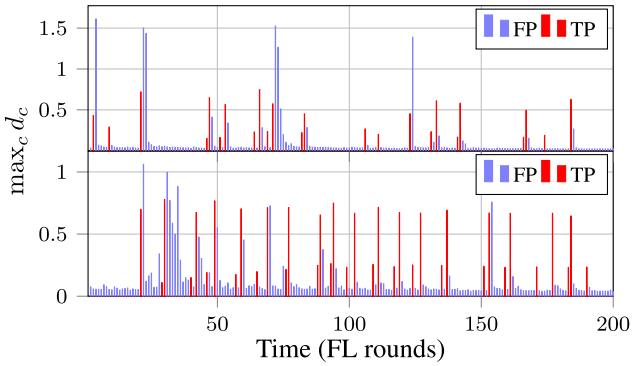

FIGURE 22. Anomalous local models detected over 200 rounds for τ = NN (top) and τ = CNN (bottom), with $k = 1$, $n_c = 10$, $p_c = 0.5$ and $f = 6$.

Red bars correspond to true positives (TP), i.e., rounds where Sender's model was detected. Instead, blue bars denote false positives (FP), i.e., when an honest client was tagged. Detection precision is 12% and 17% for NN and CNN, respectively. These values are quite low in a 10-client network since 10% is the accuracy of a random strategy. Moreover, to remain undetected, the attacker may try to minimize d_c, e.g., using poisoning data rate control [57].

Another countering approach consists in adopting *differential privacy* (DP) [76]. Briefly, in DP mechanisms for FL both the honest clients and the server inject random noise in their models to prevent a malicious client from leaking private information. DP typically does not work against collusion, i.e., Sender *wants* to leak information. Nevertheless, at the price of degrading model accuracy, noise added by the server may reduce the channel quality.

IX. CONCLUSION

In this paper, we introduced a new covert channel leveraging FL systems. Our attack allows a malicious agent to establish stealth communications between FL clients that should rather stay isolated. We discuss a prototype implementation and we empirically assess its performance. Our experiments confirm that the attack is feasible and the covert channel supports good-quality communications. Since the attack is new, no specific countermeasures exist, however some existing countermeasures for poisoning attack, e.g., [10], [47], and [57], might affect our covert channel and we plan to study them in the future. Further research directions include those related to the actual exploitability in real-world systems and to the costs for an attacker to implement our channel.

REFERENCES

[1] H. Aghakhani, T. Eisenhofer, L. Schönherr, D. Kolossa, T. Holz, C. Kruegel, and G. Vigna, "VenoMave: Targeted poisoning against speech recognition," 2020, *arXiv:2010.10682*.
[2] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *Int. J. Secur. Netw.*, vol. 10, no. 3, pp. 137–150, 2015.
[3] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, S. Chiappa and R. Calandra, Eds. vol. 108, Aug. 2020, pp. 2938–2948.
[4] M. Barreno, B. Nelson, R. Sears, D. A. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proc. ACM Symp. Inf., Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, 2006, pp. 16–25.
[5] M. Bauer, "New covert channels in HTTP: Adding unwitting web browsers to anonymity sets," in *Proc. ACM Workshop Privacy Electron. Soc. (WPES)*. New York, NY, USA: Association for Computing Machinery, 2003, pp. 72–78.
[6] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, Jun. 2019, pp. 634–643.
[7] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, F. Železný, Eds. Berlin, Germany: Springer, 2013, pp. 387–402.
[8] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. 29th Int. Conf. Int. Conf. Mach. Learn.*, Madison, WI, USA: Omnipress, 2012, pp. 1467–1474.
[9] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognit.*, vol. 84, pp. 317–331, Dec. 2018.
[10] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2017, pp. 118–128.
[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA: IEEE Computer Society, May 2017, pp. 39–57.
[12] C. Daniel Castro, J. Tan, B. Kainz, E. Konukoglu, and B. Glocker, "Morpho-MNIST: Quantitative assessment and diagnostics for representation learning," *J. Mach. Learn. Res.*, vol. 20, no. 178, 2019.
[13] S. Chandra, Z. Lin, A. Kundu, and L. Khan, "Towards a systematic study of the covert channel attacks in smartphones," in *Proc. Int. Conf. Secur. Privacy Commun. Netw.*, J. Tian, J. Jing, and M. Srivatsa, Eds. Cham, Switzerland: Springer, 2015, pp. 427–435.
[14] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," 2017, *arXiv:1712.05526*.
[15] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "FedHealth: A federated transfer learning framework for wearable healthcare," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 83–93, Jul. 2020.
[16] G. Costa, F. Pinelli, S. Soderi, and G. Tolomei, "Covert channel attack to federated learning systems," 2021, *arXiv:2104.10561*.
[17] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma, "Adversarial classification," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 99–108.
[18] J. Dumford and W. Scheirer, "Backdooring convolutional neural networks via targeted weight perturbations," 2018, *arXiv:1812.03128*.
[19] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proc. 29th USENIX Secur. Symp. (USENIX Security)*, S. Capkun and F. Roesner, Eds. Berkeley, CA, USA: USENIX Association, Aug. 2020, pp. 1605–1622.
[20] M. Fang, G. Yang, N. Z. Gong, and J. Liu, "Poisoning attacks to graph-based recommender systems," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.* New York, NY, USA: Association for Computing Machinery, Dec. 2018, pp. 381–392.
[21] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 1322–1333.
[22] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *Proc. USENIX Secur. Symp.*, San Diego, CA, USA: USENIX Association, Aug. 2014, pp. 17–32.
[23] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," 2018, *arXiv:1808.04866*.
[24] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 619–633.
[25] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," 2020, *arXiv:2012.10544*.

[26] I. Goodfellow, Y. Bengio, and A. Courville," *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[27] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," 2017, *arXiv:1708.06733*.
[28] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
[29] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics). New York, NY, USA: Springer, 2001.
[30] D. Hitaj, G. Pagnotta, B. Hitaj, F. Perez-Cruz, and V. Luigi Mancini, "FedComm: Federated learning as a medium for covert communication," 2022, *arXiv:2201.08786*.
[31] L. Huang, D. A. Joseph, B. Nelson, I. P. B. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proc. 4th ACM Workshop Secur. Artif. Intell.* New York, NY, USA: Association for Computing Machinery, 2011, pp. 43–58.
[32] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*. Washington, DC, USA: IEEE Computer Society, May 2018, pp. 19–35.
[33] M. S. Jere, T. Farnan, and F. Koushanfar, "A taxonomy of attacks on federated learning," *IEEE Secur. Privacy*, vol. 19, no. 2, pp. 20–28, Mar. 2021.
[34] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 349–363.
[35] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, 2021, pp. 1–210.
[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009.
[37] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.
[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[40] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2016, pp. 1893–1901.
[41] B. Liang, M. Su, W. You, W. Shi, and G. Yang, "Cracking classifiers for evasion: A case study on the Google's phishing pages filter," in *Proc. 25th Int. Conf. World Wide Web*, Apr. 2016, pp. 345–356.
[42] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.* San Diego, CA, USA: Internet Society, 2018, pp. 1–17.
[43] Y. Lu and L. Fan, "An efficient and robust aggregation algorithm for learning federated CNN," in *Proc. 3rd Int. Conf. Signal Process. Mach. Learn.* New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–7.
[44] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," 2020, *arXiv:2003.02133*.
[45] S. Mahloujifar, M. Mahmoody, and A. Mohammed, "Universal multi-party poisoning attacks," in *Proc. 36th Int. Conf. Mach. Learn.*, K. Chaudhuri and R. Salakhutdinov, Eds. vol. 97, Jun. 2019, pp. 4274–4283.
[46] Y. Mao, X. Yuan, X. Zhao, and S. Zhong, "Romoa: Robust model aggregation for the resistance of federated learning to model poisoning attacks," in *Proc. Eur. Symp. Res. Comput. Secur.* Heidelberg, Germany: Springer, 2021, pp. 476–496.
[47] P. M. Salehi and J. Proakis, *Digital Communications*, 5th ed. New York, NY, USA: McGraw-Hill, 2007.
[48] B. McMahan, E. Moore, D. Ramage, S. Hampson, and A. Y. B. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[49] B. McMahan and D. Ramage. (2017). *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. Accessed: Apr. 6, 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
[50] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 691–706.
[51] A. Mileva and B. Panajotov, "Covert channels in TCP/IP protocol stack–extended version-," *Open Comput. Sci.*, vol. 4, no. 2, pp. 45–66, Jan. 2014.
[52] P. K. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[53] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Proc. IEEE Symp. Security Privacy (SP)*, San Francisco, CA, USA, May 2019, pp. 739–753.
[54] B. Nelson, M. Barreno, F. J. Chi, D. A. Joseph, I. P. B. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter," in *Proc. 1st USENIX Workshop Large-Scale Exploits Emergent Threats*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–9.
[55] J. Ngiam, Z. Chen, D. Chia, P. Koh, Q. Le, and A. Ng, "Tiled convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 23, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Red Hook, NY, USA: Curran Associates, 2010.
[56] T. D. Nguyen, P. Rieger, H. Fereidooni, and S. Marchal, "FLAME: Taming backdoors in federated learning," in *Proc. 31st USENIX Secur. Symp. (USENIX Security)*, 2022, pp. 1415–1432.
[57] T. D. Nguyen, P. Rieger, M. Miettinen, and A.-R. Sadeghi, "Poisoning attacks on federated learning-based IoT intrusion detection system," in *Proc. Workshop Decentralized IoT Syst. Secur.* San Diego, CA, USA: Internet Society, 2020, pp. 1–7.
[58] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 506–519.
[59] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, H. Wallach, Eds. Red Hook, NY, USA: Curran Associates, 2019, pp. 8024–8035.
[60] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Comput. Sci. Rev.*, vol. 34, Nov. 2019, Art. no. 100199.
[61] S. R. Pokhrel and J. Choi, "A decentralized federated learning approach for connected autonomous vehicles," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2020, pp. 1–6.
[62] A. Pyrgelis, C. Troncoso, and E. D. Cristofaro, "Knock knock, who's there? Membership inference on aggregate location data," in *Proc. Netw. Distrib. Syst. Secur. Symp.* San Diego, CA, USA: Internet Society, 2018.
[63] I. P. B. Rubinstein, B. Nelson, L. Huang, D. A. Joseph, S.-H. Lau, S. Rao, N. Taft, and J. D. Tygar, "Antidote: Understanding and defending against poisoning of anomaly detectors," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas.* New York, NY, USA: Association for Computing Machinery, 2009, pp. 1–14.
[64] J. Selvi, "Covert channels over social networks," SANS Inst., Bethesda, MD, USA, Tech. Rep. 33960, Jun. 2012. [Online]. Available: https://www.sans.org/reading-room/whitepapers/engineering/paper/33960
[65] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! Targeted clean-label poisoning attacks on neural networks," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2018, pp. 6106–6116.
[66] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 3–18.
[67] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds. San Diego, CA, USA, 2015, pp. 1–14.
[68] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2019, pp. 8–14.
[69] O. Suciu, R. Mărginean, Y. Kaya, H. Daumé, and T. Dumitraș, "When does machine learning fail? Generalized transferability for evasion and poisoning attacks," in *Proc. 27th USENIX Conf. Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2018, pp. 1299–1316.
[70] G. Sun, Y. Cong, J. Dong, Q. Wang, and J. Liu, "Data poisoning attacks on federated machine learning," 2020, *arXiv:2004.10020*.
[71] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, J. I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Represent. (ICLR)*, Banff, AB, Canada, Y. Bengio and Y. LeCun, Eds. Apr. 2014, pp. 1–10.

[72] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2020, pp. 480–501.
[73] F. Tramèr, F. Zhang, A. Juels, K. M. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Conf. Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2016, pp. 601–618.
[74] V. Vapnik, "Principles of risk minimization for learning theory," in *Proc. 4th Int. Conf. Neural Inf. Process. Syst.*, San Francisco, CA, USA: Morgan Kaufmann, 1991, pp. 831–838.
[75] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*. San Francisco, CA, USA: IEEE Computer Society, May 2018, pp. 36–52.
[76] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, Q. S. T. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.
[77] Q. Wu, K. He, and X. Chen, "Personalized federated learning for intelligent IoT applications: A cloud-edge based framework," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 35–44, 2020.
[78] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?" in *Proc. 32nd Int. Conf. Mach. Learn.*, Lille, France, F. Bach and D. Blei, Eds. vol. 37, Jul. 2015, pp. 1689–1698.
[79] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed backdoor attacks against federated learning," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–19.
[80] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems," in *Proc. Netw. Distrib. Syst. Secur. Symp.* San Diego, CA, USA: Internet Society, 2017.
[81] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving Google keyboard query suggestions," 2018, *arXiv:1812.02903*.
[82] S. Zander, G. Armitage, and P. Branch, "Covert channels and countermeasures in computer network protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, pp. 44–57, Sep. 2007.
[83] J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu, "Poisoning attack in federated learning using generative adversarial nets," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Rotorua, New Zealand, Aug. 2019, pp. 374–380.
[84] C. Zhou, A. Fu, S. Yu, W. Yang, H. Wang, and Y. Zhang, "Privacy-preserving federated learning in fog computing," *IEEE Internet Things J.*, vol. 7, no. 11, pp. 10782–10793, Apr. 2020.
[85] C. Zhu, W. Ronny Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein, "Transferable clean-label poisoning attacks on deep neural nets," in *Proc. 36th Int. Conf. Mach. Learn.*, K. Chaudhuri and R. Salakhutdinov, Eds. vol. 97, Jun. 2019, pp. 7614–7623.

GABRIELE COSTA (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science, in 2007 and 2011, respectively. He was a member of the Cybersecurity Group, Istituto di Informatica e Telematica (IIT) of the CNR. He is currently an Associate Professor at the SySMA Group, IMT School for Advanced Studies. His appointments include a period as a Visiting Researcher at ETH Zurich, in 2016 and 2017. He is also the Co-Founder and the CRO of Talos, a spin-off of DIBRIS focused on cybersecurity. He focuses on studying and applying formal methods for the automatic verification and testing of mobile and modular systems.

FABIO PINELLI received the M.Sc. degree in computer science and the Ph.D. degree in information engineering from the University of Pisa, in 2005 and 2010, respectively. He is currently an Assistant Professor at the SySMA Group, IMT School for Advanced Studies. He was a Research Scientist at IBM-Research Ireland and a Senior Data Scientist at Vodafone Italia. His research interests include data mining and machine learning methods applied to different domains, from economics to urban environments.

SIMONE SODERI (Senior Member, IEEE) received the M.Sc. degree from the University of Florence, Italy, in 2002, and the Dr.Sc. degree from the University of Oulu, Finland, in 2016. His skills range from cybersecurity and wireless communications to software engineering. Currently, he is an Assistant Professor at the IMT School for Advanced Studies Lucca. He has published journals and conference papers and chapters in a book. He holds five patents regarding wireless communications and positioning. His research interests include cybersecurity for critical infrastructure systems, 6G, covert channels, network security, physical layer security, electromagnetic emissions security, VLC, and UWB.

GABRIELE TOLOMEI received the M.Sc. degree in computer science from the University of Pisa, in 2005, and the Ph.D. degree in computer science from the Ca' Foscari University of Venice, in 2011. He is currently an Associate Professor at the Department of Computer Science, Sapienza University of Rome. He was a Research Scientist at the Yahoo Laboratories, London, and an Assistant Professor at the Department of Mathematics, University of Padua. His research interests include (human-)explainable, robust, and collaborative machine learning. He has authored more than 40 papers in peer-reviewed international conferences and journals and he is the inventor of four U.S. patents.

Open Access funding provided by 'Scuola IMT Alti Studi Lucca' within the CRUI CARE Agreement