Chapter 16
Security and Robustness in Federated Learning

Ambrish Rawat, Giulio Zizzo, Muhammad Zaid Hameed, and Luis Muñoz-González

Abstract Federated learning (FL) has emerged as a powerful approach to decentralize the training of machine learning algorithms, allowing the training of collaborative models while preserving the privacy of the datasets provided by different parties. Despite the benefits, FL is also vulnerable to adversaries, similar to other machine learning (ML) algorithms in centralized settings. For example, just a single malicious or faulty participant in an FL task can entirely compromise the performance of the model when using unsecure implementations. In this chapter, we provide a comprehensive analysis of the vulnerabilities of FL algorithms to different attacks that can compromise their performance. We describe a taxonomy of attacks comparing the similarities and differences with respect to centralized ML algorithms. Then, we describe and analyze different families of existing defenses that can be applied to mitigate these threats. Finally, we review a set of comprehensive attacks that aim to compromise the performance and convergence of FL.

16.1 Introduction

Artificial intelligence (AI) and especially machine learning (ML) are at the core of the fourth industrial revolution. ML has become one of the main components of many systems and applications with success stories across different sectors, including healthcare [37], financial markets [10], or Internet of Things (IoT) [19]. The benefits of ML technologies are clear, as they allow the efficient automation of many processes and tasks by leveraging their capability to analyze a huge amount of data.

A. Rawat (✉) · G. Zizzo
IBM Research Europe, Dublin, Ireland
e-mail: ambrish.rawat@ie.ibm.com; giulio.zizzo2@ie.ibm.com

M. Z. Hameed · L. Muñoz-González
Imperial College, London, UK
e-mail: muhammad.hameed13@imperial.ac.uk; l.munoz-gonzalez@imperial.ac.uk

H. Ludwig, N. Baracaldo (eds.), *Federated Learning*,
https://doi.org/10.1007/978-3-030-96896-0_16

Recently, federated learning (FL) has emerged as a promising approach for the development of distributed ML systems, allowing us to resolve challenges in some application domains. FL allows us to train a shared ML model from a federation of participants who use their own datasets to locally train a machine learning model while preserving the privacy of their datasets within the federation. In this approach, there is a central aggregator (server) that combines the information that controls the learning process and aggregates the information from the parties (clients) during the training of the ML model. These parties train the models locally using their own dataset and send the model updates back to the central aggregator in an iterative manner. In this way, during training, the data always remains with the party, keeping their datasets private.

Given current laws and privacy regulations such as the *General Data Protection Regulation* (GDPR) in the European Union, or the *Health Insurance Portability and Accountability Act* (HIPAA) in the US, FL offers an appealing alternative to build collaborative models across different institutions or companies in sensitive domains, such as healthcare or financial markets, by preserving the privacy of the party data. On the other hand, with the increasing computational capabilities of edge devices, including smartphones, sensors, and other IoT devices, FL also allows us to decentralize the training of the ML models and push the computation to edge devices. Thus, the data does not need to be collected and centralized, but edge devices contribute toward the shared FL model performing local computations using their own data.

Despite the benefits and the advantages of ML and FL technologies, there are still challenges and risks that need to be analyzed, understood, and mitigated. ML algorithms are known to be vulnerable to attackers. At training time, ML algorithms can be subject to poisoning attacks, where attackers can influence the training of the learning algorithm to manipulate and degrade its performance. This can be achieved, for example, by manipulating the data that is used to train the ML model. Attackers can also introduce *backdoors* during the training of the learning algorithm, so that the performance of the model is not altered for regular inputs, but a specific and unexpected behavior of the model is observed for inputs containing a *trigger* that activates the backdoor [12]. During deployment, ML algorithms are particularly vulnerable to *adversarial examples*, inputs specifically crafted by the attacker that contain a very small perturbation with respect to the original sample, that are designed to produce errors in the system [16].

These vulnerabilities of the learning algorithms are also present in FL. However, the mechanisms that attackers can leverage to compromise the learning algorithms are, in some cases, different to those where ML is applied to centralized data sources, requiring special consideration. In this chapter, we provide a comprehensive description of the different attacks that can be performed to compromise FL algorithms, including an extended taxonomy to model the attack surface compared to the taxonomies typically used for centralized learning algorithms. Using this taxonomy, we categorize these different sets of attacks as well as defenses that can be applied to mitigate them both at training and test time. This includes data and model poisoning attacks aiming to compromise the performance or the convergence

of the FL algorithms, backdoors, and evasion attacks as exemplified by adversarial examples.

The rest of the chapter is organized as follows: In Sect. 16.2 we describe the threat model and present a taxonomy of attacks that can be performed against FL algorithms. Section 16.3, explains different defensive schemes capable of mitigating these threats. Section 16.4 provides a comprehensive description of different attack strategies that have been proposed to compromise FL algorithms, both at training and test time. Finally, Sect. 16.5 concludes the chapter.

16.1.1 Notation

In the rest of the chapter, we take classification models as the guiding example, but many principles transfer to other types of machine learning tasks. In a federated learning process, C parties with their individual data source, $\{D_i\}_{i=1}^{C}$, composed of (x, y) data–label pairs, seek to learn a common ML model f_w. During each training round, N parties participate by sending update vectors $\{v_i\}_{i=1}^{N}$ to the central aggregator. They obtain this update vector by optimizing for a common objective L with respect to their respective private data partition. The aggregator combines these updates, often by averaging, and broadcasts the corresponding vector to all C parties. For most of the discussions in the following section, we assume that all N parties participate in each training round.

16.2 Threats in Federated Learning

In this section, we present a threat model to describe the different threats and attacks possible against FL systems. This allows us to understand the vulnerabilities, providing a systematic framework to analyze the security aspects of FL.

For this, we rely on the frameworks originally proposed in [2, 18] and extended and revised in [25] for standard ML algorithms. Thus, we describe the threat model characterizing attacks according to the attacker's goal, capabilities to manipulate the data and influence the learning system, knowledge of the target system and data, as well as the attacker's strategy. Although some of these aspects are similar to those for standard learning algorithms, there are certain aspects of the threat model that are unique to FL scenarios which we discuss in the following sections.

16.2.1 Types of Attackers

Before contextualizing the threat model in similar terms to those in non-distributed ML algorithms, we need to define the specific types of attackers that are possible

in FL scenarios. This differs from centralized ML algorithms, where the attacker is typically considered external to the system and aims to compromise or degrade the system's performance, produce errors in the system, or leak information about the target system or the data used to train the model. In other cases, the attackers can also manipulate the software or code implementations used to train the machine learning models. In addition to this, in FL, some of the parties (users) within the system can also behave maliciously.

Thus, in FL systems, we can categorize attackers as:

- **Outsiders:** similar to the case of centralized learning algorithms, outsiders are attackers that are not users (parties) of the platform. They can compromise the FL system at training time by poisoning the training datasets of benign parties to perform poisoning or backdoor attacks. At test time, they can exploit the weaknesses and blind spots of the resulting models to produce errors, e.g., with adversarial examples [16], or to extract some knowledge from the target model, e.g., membership inference attacks [28]. This category can also include attackers that are capable of intercepting and tampering with the communications between the central node and some of the parties of the FL platform.
- **Insiders:** this includes cases where one or several users (parties) of the FL platform are malicious. These attackers can also manipulate and degrade the performance of the system to gain some advantage with respect to other parties but have more freedom than outsiders to do so. For example, for poisoning the federated learning model, insiders can directly manipulate the parameters of the model sent to the aggregator. Insiders can also aim to leak information from the datasets used by the other users, e.g., with property inference attacks [17, 23, 28]. In cases where there are several insiders in the FL platform, as shown in Fig. 16.1, there are different possible scenarios depending on whether the attackers collude toward the same malicious objective.

16.2.2 Attacker's Capabilities

The capabilities of the attacker to compromise an FL system can be categorized in terms of the attacker's influence on the data, the model, and any additional constraints which limit the attacker such as the presence of defensive algorithms.

16.2.2.1 Attack Influence

According to the capabilities of the attacker to influence or compromise the ML model, attacks can be classified as:

- **Causative:** if the attacker can influence the learning algorithm by injecting or manipulating data used to train the learning algorithms or providing malicious

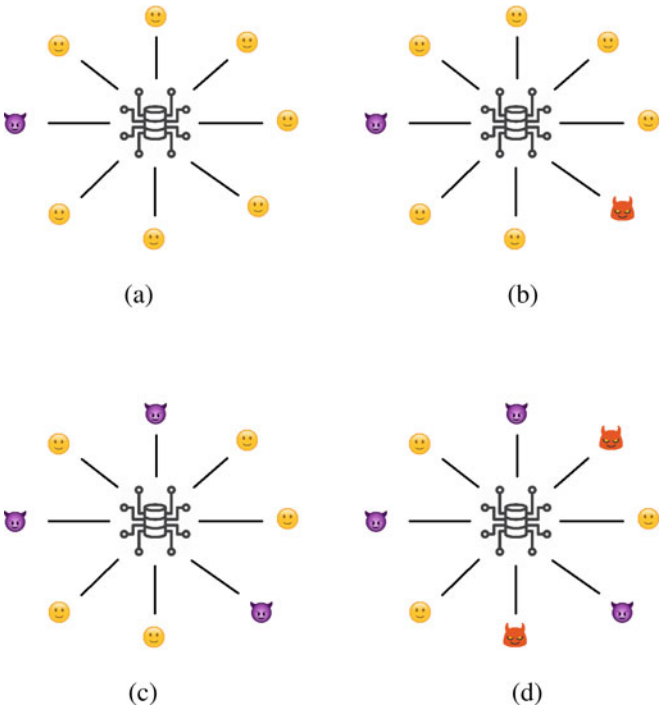

Fig. 16.1 Different scenarios of insider attackers in FL: (**a**) single attacker. (**b**) Several non-colluding attackers, i.e., the attackers have different objectives. (**c**) Group of colluding attackers. (**d**) Different groups of colluding attackers

information to manipulate the parameters of the system. These attacks are commonly referred to as *poisoning attacks*. *Byzantine* attacks [8] which send arbitrary updates to compromise model performance also qualify as causative within this categorization.

- **Exploratory:** the attacker cannot influence the training process but can attempt to exploit the weaknesses and blind spots of the system at test time or to extract information from the target system. Scenarios where attackers aim to produce errors in the target system are usually referred to as *evasion attacks*.

Poisoning attacks are an important threat in scenarios where the data collected to train the learning algorithms is untrusted. This is common in applications that collect data from humans who can act dishonestly or devices whose integrity can be at risk. In FL systems, poisoning attacks can be performed by both insiders and outsiders. In the case of outsiders, poisoning attacks can be achieved by injecting malicious data in the training datasets used by the participants, by compromising the integrity of software used by the participant or compromising communications between the participants and the central node. In this latter case, the attackers can perform stronger poisoning attacks via model poisoning [5].

At run-time, in evasion attacks, even if the data used for training the FL model is trusted and all the participants are honest, the attackers can probe the resulting model to produce intentional errors, for example, by crafting adversarial examples. On the other hand, there are other exploratory attacks that aim to compromise the privacy of the model, extracting or leaking information about the model and its training data. In this sense, similar to non-distributed ML settings, FL models can be vulnerable to *membership inference attacks*, where the attacker tries to assert if a data point has been used for training the learning algorithm, as described in Chap.. In this case, the difference in FL is that the attacker may not know which participant provided that data point. FL on the other hand leverages data from numerous participants, and the model is often trained with more data points compared to the centralized case, which increases the effort for the attacker to perform membership inference attacks [28]. In some settings, insider attackers can also perform *property inference attacks*, aiming to infer properties from the training data used by other participants [17, 23]. This can be achieved by examining the model updates during training. However, these attacks can only achieve a certain degree of success under very particular conditions: a limited number of participants and highly differentiated properties across the datasets of the participants.

16.2.2.2 Data Manipulation Constraints

The attacker's capabilities may be limited by the presence of constraints for the manipulation of the data or the parameters of the model in the case of model poisoning attacks. The attacker can also self-impose some constraints to remain undetected and perform stealthy attacks by, for example, crafting attack points that do not differ too much from benign data points. The manipulation constraints are also strongly related to the particular application domain. For example, in ML-based malware detection, the attacker's aim is to evade detection by manipulating the malware code, but those manipulations need to preserve the malicious functionality of the program [11, 32]. In contrast, in some computer vision applications, it is reasonable to assume that the attackers can manipulate every pixel in an image or every frame in a video.

In data poisoning attacks, the adversary can have different degrees of freedom to manipulate the data. For certain cases, the attacker may be in control of part of the labeling process (e.g., when using crowdsourcing). These are known as label flipping attacks. In other scenarios, even if the attacker is not in control of the labels assigned to the poisoning points, they can reliably estimate the label that will be assigned to the injected malicious points. For example, in a spam detection application, the attackers can assume that most of the malicious emails injected in the system will be labeled as spam.

When assessing the robustness of ML and FL algorithms to attacks, it is important to model realistic data constraints to better characterize worst-case scenarios, for example, through optimal attack strategies, where the attacker aims to maximize the damage on the target algorithm. However, it is important to consider

appropriate detectability constraints; otherwise, the attacks can be trivial and can be easily detected with orthogonal methods, such as data pre-filtering or outlier detection [30, 31].

16.2.3 Attacker's Goal

The goal of the adversary can be categorized based on the type of security violation that the attacker seeks to achieve and the specificity of the attack, which can be described in terms of the number of data points affected by the attack, or on the type of errors to be produced in the system.

16.2.3.1 Security Violation

We can differentiate three different security violations against ML and FL systems:

- **Integrity violation:** when the attack evades detection without compromising the system's normal operation.
- **Availability violation:** when the attacker aims to compromise the functionality of the system.
- **Privacy violation:** when the adversary obtains private information about the target system, the data used for training, or the users of the system.

Integrity and availability violations depend upon the application to be deployed and the attacker's capabilities to influence the training of the learning algorithm. In this sense, in FL, for insider threats, the attackers can not only poison the learning algorithm but also prevent the algorithm to converge during its training. On the privacy side, as mentioned previously, FL models can be vulnerable to membership and property inference attacks.

16.2.3.2 Attack Specificity

This characteristic is defined by a continuum spectrum that describes the specificity of the attacker's intention ranging from targeted to indiscriminate attack scenarios:

- **Targeted Attacks:** where the attacker aims to degrade the performance of the system or to produce errors for a reduce set of target data points.
- **Indiscriminate Attacks:** where the attacker aims to degrade the system's performance or to produce errors in an indiscriminate fashion, i.e., affecting a broad set of cases or data points.

Different from the taxonomy originally proposed in [2, 18], in the research literature on adversarial examples, i.e., evasion attacks for specific inputs, the term *targeted attack* usually refers to the case where the attacker aims to evade the target model producing a specific type of error, whereas *untargeted attacks* refer to those attacks that just aim to produce errors regardless of the nature of the error. However, the related work on poisoning attacks follows the original taxonomy [2, 18], so that *indiscriminate poisoning attacks* are those that produce errors for a large set of inputs, and *targeted poisoning attacks* are those that produce errors on a reduced set of target inputs. However, the taxonomy in [2, 18] is limited to describe attacks depending on the nature of the errors. This limitation was addressed by Muñoz-González et al. [26], extending the taxonomy to categorize attacks according to the type of errors that the attacker wants to produce.

16.2.3.3 Error Specificity

As described in [26], in some cases, such as multi-class classification, depending on the nature of the errors that the attacker seeks to produce in the system, we can categorize the attacks as:

- **Error-generic:** when the adversary wants to produce errors in the target system regardless of the type of error to be produced.
- **Error-specific:** when the attacker aims to produce a specific type of errors in the system. This can be application dependent. In fact, depending on their capabilities, the attackers can be constrained on the type of errors that can be produced in the system.

While the categorization of targeted and indiscriminate attacks is based on specificity with respect to data samples, the error specificity characterizes the orthogonal dimension of quality of error—like an error-specific attacker could seek misclassification while an error-generic attacker might pursue more general objectives for system compromise. For example, in the context of data poisoning, an error-specific indiscriminate poisoning attack aims at maximizing the performance of the model over a large set of test inputs producing specific type of errors (e.g., classifying all the samples from all classes as samples from class "0"), whereas in the case of an error-generic indiscriminate attack, the adversary does not care about the nature of the errors produced in the system and just aims at maximizing the overall error of the model for a large set of inputs.

16.2.4 Attacker's Knowledge

The attacker's knowledge of the target FL system includes the following aspects:

- The datasets used by one or more participants
- The features used to train the learning algorithm and their range of valid values

- The learning algorithm, the objective function to be optimized, and the aggregation method used by the central node
- The parameters of the FL algorithm and the resulting model

Depending on how much the attacker knows about the previous points, we can differentiate two main scenarios: *perfect* and *limited* knowledge attacks.

16.2.4.1 Perfect Knowledge Attacks

These are scenarios where we assume that the attacker knows everything about the target system. Although this assumption can be unrealistic in most cases, perfect knowledge attacks are useful to assess the robustness and security of ML and FL algorithms in worst-case scenarios, helping to provide lower bounds in the performance of the algorithm for different attack's strength. Furthermore, they can be useful for model selection, by comparing the performance and robustness of different algorithms and architectures tested against these type of attacks.

16.2.4.2 Limited Knowledge Attacks

There is a broad range of possibilities to model attacks with limited knowledge. Typically, in the research literature, two main categories are considered:

- **Limited knowledge attacks with surrogate data:** this includes scenarios where the attacker knows the model used for the learning algorithm, the feature representation, the objective function, and the aggregation scheme used by the aggregator. However, the attackers do not have access to the training data, although they can have access to a surrogate dataset with similar characteristics to the dataset used to train the target learning algorithm. Then, the attacker can estimate the parameters of the targeted model by using this surrogate dataset, which can enable successful attacks depending on the quality of the surrogate dataset. In the case of FL, this is a reasonable assumption to model insider attackers. Such an adversary has access to the model information and their own dataset, but not to the datasets of the rest of the participants.
- **Limited knowledge attacks with surrogate models:** this category includes scenarios where the attackers have access to the dataset and the feature representation used by the target system, but they do not have access to the ML model, the objective function to be optimized, or the aggregation method used by the central node. In these cases, the attackers can train a surrogate model to estimate the behavior of the system. By crafting attacks against this surrogate model, the resulting malicious points are used to attack the real model. This strategy can be effective to achieve successful attacks, especially if the surrogate models are similar, as the vulnerabilities of different model architectures and learning algorithms are similar in some cases. This is commonly referred to as *attack transferability* and has been shown for both evasion [29] and poisoning attacks [26].

Although perfect knowledge attacks can be helpful to model worst-case scenarios for testing the robustness of many FL systems, a balance between realistic and worst-case scenarios should be considered in practical deployments. For example, in most cases, both insider and outsider attackers will not have access to the datasets from all the participants. Therefore, asserting robustness of FL algorithms against weaker adversaries can be a useful and well motivated threat model to investigate.

16.2.5 Attack Strategy

Attack strategies against both standard ML and FL systems can be formulated as an optimization problem capturing different aspects from the threat model. The attacker's goal can be characterized by an objective function evaluated on a set of predefined data points, which can be a specific set of target points or, for indiscriminate attacks, a representative set of the underlying data distribution used by the target system. This objective function typically helps the attacker to assess the effectiveness of an attack strategy. The objective function can also include specific constraints to prevent being detected by the defender. In Sect. 16.4, we will show a comprehensive set of attack strategies that can be used to compromise FL algorithms.

Finally, Table 16.1 summarizes the threat model presented in this section.

16.3 Defense Strategies

We now look at different defense strategies that have been devised to counter the different types of attacks described in the previous section. Designing a defense method incurs several challenges. To take one, it is essential that defense mechanisms preserve the model performance in the absence of malicious parties. The FL model assumptions may also affect the design strategy for defenses. The aggregator, for instance, may not have the ability to inspect model updates [9]. In this section we distill some broad themes that have been used for designing defenses for FL systems. First, we look at defenses developed for convergence attacks. Broadly speaking, these methods inspect the set of updates across all parties during each training round and use a filtering criterion with the aggregation. We then describe an alternate line of defenses which incorporate the update history for this process. A third category of defenses are based on redundancy between party data partitions.

It is worth noting that a large class of defenses developed for centralized systems naturally apply to federated settings. However, FL-specific scenarios do require specialized approaches which we discuss in the following section.

Table 16.1 Threat model in federated learning

Types of attackers	• **Outsiders:** attackers external to the platform • **Insiders:** attackers that are participating in the FL task
Attacker's capabilities	**Attack influence** • *Causative attacks:* the attacker can influence the learning algorithm (e.g., poisoning or backdoor attacks) • *Exploratory attacks:* the attacker can only manipulate data at test time (e.g., adversarial examples)
Attacker's goal	**Security violation** • *Integrity attacks* (e.g., backdoor attacks) • *Availability attacks* (e.g., poisoning attacks) • *Privacy violation* (e.g., property inference attacks) **Attack specificity** • *Targeted attacks:* focused on a specific set of cases or data points • *Indiscriminate Attacks:* target a broader set of cases or data points **Error specificity:** • *Error-generic attacks:* the attacker just aims to produce errors in the system, regardless of their nature • *Error-specific attacks:* the attacker aims to produce specific types of errors in the target system
Attacker's knowledge	• **Perfect knowledge:** the attacker knows everything about the target system • **Limited knowledge:** – *Surrogate dataset:* the attacker knows the target model but not the training dataset (or has partial knowledge of it) – *Surrogate model:* the attacker knows the training dataset but not the model (e.g., transfer attacks)

Backdoor attacks often include a subtask for which the adversary seeks high performance. The first lines of defenses against such attacks are implemented at the aggregator and assume that the updates for backdoor tasks would be outside the natural spread of benign updates. Two strategies that handle this perspective include *norm clipping* and *weak differential privacy*. For norm clipping, the central aggregator inspects the difference between the broadcasted global model and the received updates from the selected parties and clips the updates that exceed a pre-specified norm threshold [39]. On the other side, weak differential privacy

approaches, as in [39, 43], add Gaussian noise with a small standard deviation to the model updates prior to the aggregation. Weak backdoor attacks can be easily countered with such defenses as the addition of Gaussian noise can neutralize the backdoor update.

Defenses that function against evasion attacks can similarly be employed in a federated setting. Adversarial training, [22] in which the defender trains against adversarial examples, is one such popular defense, and however this is a challenging training task and its difficulty increases in federated learning settings. For example, Shah et al. [36] observed that the performance against adversarial examples was strongly influenced by the amount of local computation conducted by the party, and Zizzo et al. [50] noted that the proportion of adversarial examples to clean data in a batch has a significant impact. The work in [41] also shows that robustness against affine distribution shifts (which can occur between parties in federated learning) can offer protection against adversarial examples. Effectively conducting adversarial training in a federated context remains an open problem, not only due to the underlying optimization difficulties but also attackers can interfere with the training process and create brittle models with misleading performance metrics for a defender [50].

16.3.1 Defending Against Convergence Attacks

For convergence attacks, we need to protect against adversaries who aim to degrade model performance in an unrestricted manner. A common attacker model to defend against in this scenario is a *Byzantine* attacker. This corresponds to a strong adversary who can send arbitrary model updates. Typically, in these attacks, the malicious model updates differ significantly from those sent by the benign parties and aim to produce a completely useless machine learning model, i.e., the performance of the resulting model is very poor. This can be achieved by, for example, sending random model updates adding noise with a very large variance to all the model's parameters. Blanchard et al. [8] showed that a single Byzantine adversary is enough to completely compromise a federated learning model when using standard aggregation methods, such as federated averaging.

This can be easily shown: for a set of party updates $\{v_k\}_{k=1}^{N}$, if the attacker aims for the global model to have a specific set of parameters w, and they control the party $k = N$, then the update required can be exactly computed as

$$w = \frac{1}{N} \sum_{k=1}^{N-1} v_k + \frac{1}{N} v_N \tag{16.1}$$

$$v_N = Nw - \sum_{k=1}^{N-1} v_k. \tag{16.2}$$

Even without the knowledge of benign party updates, an attacker can trivially compromise the system. An attacker controlled party can send arbitrarily large updates which, when averaged with the benign parties, will break the model.

Thus, for practical FL deployments, it is essential to include mechanisms to filter out malicious (or faulty) model updates that can compromise the overall system's performance. This vulnerability has fostered research on robust aggregation methods aiming to detect and mitigate different types of poisoning attacks, including Byzantine adversaries.

16.3.1.1 Krum

Krum is one of the first algorithms proposed to defend against convergence attacks in FL [8]. A naive defender could try and filter out attackers by computing a score based on the squared distance between update i and all other received updates, to then select the update with the lowest score. This mechanism will however only tolerate a single Byzantine party. As soon as two Byzantine parties collude, then one Byzantine party can propose an update which shifts the barycenter of the benign party updates toward the other Byzantine update.

Krum solves this problem by being more selective in computing distance measures. Given N party updates $\{v_k\}_{k=1}^{N}$, Krum selects the update u which has the lowest squared Euclidean distance with respect to its $N - F - 2$ neighbors, where F is the allowable number of malicious parties in the system. We can express this as

$$s(i) = \sum_{i \to j} ||v_i - v_j||^2, \tag{16.3}$$

where we only sum over the $N - F - 2$ parties with the lowest squared distance. Krum requires that the number of malicious workers satisfies $2F + 2 < N$. We can see an example of Krum acting on a 2D set of updates in Fig. 16.2 where we only sum over the $N - F - 2$ parties with the lowest squared distance.

Although Krum can be effective to mitigate some attacks, especially Byzantine adversaries, it has been shown that this defense is not effective to mitigate other type of attacks, like label flipping attacks [27], or can be brittle against adaptive attacks targeting Krum [39]. Apart from this, the use of Krum slows the convergence of the FL algorithm, requiring more training rounds to achieve a good level of performance [8]. On the other side, Krum requires to compute the Euclidean distance between the model updates sent by all the parties participating at each training round, which can be computationally very demanding for scenarios where the number of parties is large.

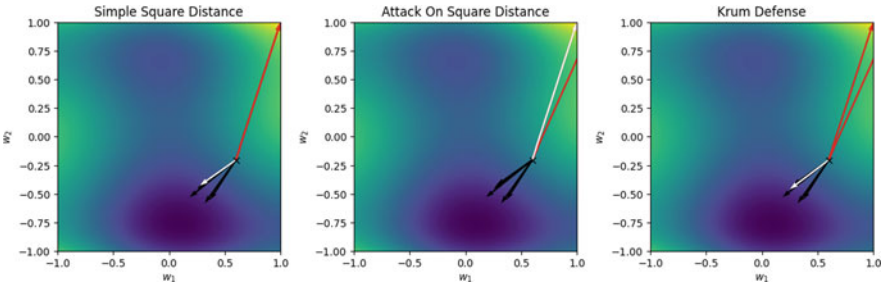

Fig. 16.2 Illustration of the Krum defense in a synthetic example with two parameters for the FL model. If we score each update based on its square distance to all other updates, then, by selecting the update which has the minimum score (white vector), we can successfully handle one Byzantine party. However, two malicious parties can collude, and now one update shifts the barycenter of all the supplied updates so that the original attacker update is selected. Note that this new Byzantine update (red line in the middle plot) is extremely large as it needs to counter the effect of all the benign parties. In fact, as we can see from the middle plot, it extends well beyond the w_1 and w_2 ranges we visualize. However, from the rightmost plot, if we apply Krum (which only considers the closest $N - F - 2$ parties), then many benign parties never have their score influenced by the malicious parties, and we can see that Krum reselects a benign party

The problems on the slow convergence of Krum can be partially mitigated with **Multi-Krum**, a straightforward variant of the algorithm where, instead of selecting a single update, we select the lowest scoring M updates so that the final update is given by

$$\frac{1}{M}\sum_{i} v_i^*, \tag{16.4}$$

where v^* is the set of the M lowest scoring party updates. This intuitively is interpolating between Krum and federated averaging, with M acting as a tunable parameter that a defender can set to prioritize convergence speed or robustness.

16.3.1.2 Median-Based Defenses

Methods based on the median form a broad family of defenses. If the number of malicious parties, F, is less than half of the total number of parties N, $F \leq \lceil \frac{N}{2} \rceil - 1$, then the median of a particular parameter must come from a benign party. Therefore, with this group of defenses, we are computing the median independently for every *dimension* of a parameter update. This is in contrast to Krum, which by using the squared Euclidean distance between two updates does not distinguish between the cases where updates differ significantly on only a few components, compared to when updates differ slightly on many components.

In its simplest form, we independently compute the median along every dimension and apply it to the global model as an update. However, there are a group of defenses that perform filtering around the median and then average the resulting parties. These are broadly referred to as *Trimmed Mean*-based defenses [24, 45, 49]. Concretely, the median for jth dimension in the N party updates $\{v_k\}_{k=1}^{N}$ is computed and a filtering operation is conducted. The remaining updates on each dimension are then averaged resulting in the final update vector. This is expressed as

$$w^{(j)} = \frac{1}{|U_j|} \sum_{i \in U_j} v_i^{(j)}, \tag{16.5}$$

where $|U_j|$ is the cardinality of the selected updates on dimension j. It is in the filtering step that the different Trimmed Mean algorithms differ. In particular,

- In [45], with $F \leq \lceil \frac{N}{2} \rceil - 1$, select the closest $N - F$ values to the median to average.
- In [24], with $N - 2F \geq 3$, only pick the nearest $N - 2F$ updates.
- Finally, for [49], with $F \leq \lceil \frac{N}{2} \rceil - 1$, remove the largest and smallest F updates on each dimension.

16.3.1.3 Bulyan

The Bulyan [24] defense seeks to combine the strengths of the previously discussed defenses. Krum has a shortcoming as it analyzes party updates based on the Euclidean distances of the local models across parties. Thus, adversaries can propose model updates which differ significantly on only a few parameters which will have little effect on the overall distance with respect to model updates from benign parties, but that can have a significant impact on the model performance. Bulyan thus computes a two-step process, in which Krum first produces a set of *likely* benign parties and then Trimmed Mean acts on this set derived from Krum. To be more precise,

- On the set of received party updates $V = \{v_i\}_{i=1}^{N}$, apply Krum which will select a single update.
- Add the update selected by Krum to a selection set S and remove the update from V.
- Apply the above two steps $N - 2F$ times. Thus, we are shrinking V and growing S by one update every iteration.
- Finally, apply Trimmed Mean on the resulting selection set S.

The Bulyan defense has robustness up to $N \geq 4F + 3$.

A different route is to directly limit the influence of the absolute value of any party's updates on the overall aggregation. One method for achieving this is to consider the sign of an update [4, 20]. In addition to limiting the influence of individual

parties, it makes the communication between the parties and the aggregator much more efficient as only one-bit update is needed for every dimension in the update vector. Sign-based methods have been shown, under the assumption that updates are unimodal and symmetric about the mean, to be able to converge. Sign methods can also be viewed, as was done in [20], as a form of L_1 regularization. However, simple sign-based methods can be vulnerable to adaptive adversaries. Consider the algorithm in [4] in the following attack:

Example: Consider a system of 9 parties. The benign updates are modelled as coming from a Gaussian distribution $\mathcal{N}(0.2, 0.15)$. *We model 5 benign workers which generate updates* $v_b = \{0.037, 0.4, 0.24, -0.026, 0.11\}$, *which when signed have* $v_b = \{1, 1, 1, -1, 1\}$. *The attacker breaks the unimodal requirement and submits updates from 4 malicious parties of* $v_m = \{-1, -1, -1, -1\}$ *with a negative sign. Although the true update direction should be positive, the sum over all signed updates is now −1.*

16.3.1.4 Zeno

Should the aggregator have additional capabilities with access to the data itself, then further analysis can be conducted by examining the effect of the update on the model's performance on the aggregator data. This was examined in [48] which proposed the *Zeno* defense. Zeno produces a score s for every supplied gradient update v which indicates its reliability. The key idea here is to use the validation data to estimate the descent of the loss function value after a party update is applied. The score, s, is defined as

$$s = L(w, X) - L(w - \gamma v, X) - \rho||v||^2, \tag{16.6}$$

where w is the current parameter vector, γ is the learning rate at the aggregator, X represents samples of data drawn from the data distribution, and L is the loss function of the underlying machine learning task. The updates with the highest scoring s are averaged and used to update w. This can offer very strong defensive performance, and however the existence of an aggregator side dataset introduces additional requirements for the FL system.

16.3.2 Defenses Based on Parties' Temporal Consistency

In the previous section, we discussed defense aggregation methods that analyze a party's updates in each training round independently of their behavior during earlier rounds. This means a party's update in one training round does not affect

its participation in the overall aggregation at later stages. Parties under the influence of attacks are likely to exhibit consistent malicious behavior across different training rounds and defense schemes can benefit from this knowledge by monitoring a party's temporal behavior during the training process. This insight can be employed for efficient and more accurate detection of malicious parties. Robust aggregation schemes based on these observations have been proposed in [27, 41], which either directly model the party's behavior during the training process or use a detection scheme to identify the parties sending malicious updates during the course of training. Furthermore, once the malicious parties are identified, they can be prevented from further participating in the training process which can result in reduced communication cost at the aggregator side.

16.3.2.1 Adaptive Model Averaging (AFA)

Muñoz-González et al. [27] propose an algorithm that relies on two components: (1) a robust aggregation rule to detect malicious model updates and (2) a probabilistic model using a hidden Markov model (HMM) that learns the quality of the model updates provided by each party during training and models their behavior.

The parameters of HMM are updated during each training round and implicitly incorporate the quality of update history for each party. They further use the HMM to detect a malicious party and then subsequently bar the malicious party from further participating in the training process. The proposed robust scheme aggregates the update at iteration $t + 1$ as

$$\sum_{k \in \mathcal{K}_t^g} \frac{p_{k_t}\, n_k}{P} v_k, \tag{16.7}$$

where p_{k_t} is the probability of party k providing a useful model update at iteration t and $P = \sum_{k \in \mathcal{K}_t^g} p_{k_t} n_k$, where n_k is the size of the dataset owned by party k. The set $\mathcal{K}_t^g \subset \mathcal{K}_t$ contains the parties that provide a good update according to the robust aggregation algorithm proposed in this chapter. For this, at each training round, AFA aims to detect malicious model updates iteratively using a distance-based algorithm (using cosine similarity or Euclidean distance). This detection algorithm is independent from the past contributions of the parties, to avoid situations where attackers are silent for some training rounds. At the start of the training process, updates from all parties are in the set of good updates. The aggregated model is estimated from Eq. 16.7 for given probabilities of the parties and the number of training data points provided. Then, the similarity of each party with respect to the aggregated model is calculated, and finally the mean, $\hat{\mu}$, and the median, $\bar{\mu}$, of all these similarity measures are calculated as shown in Fig. 16.3. Thereafter, each party's similarity score is compared to a threshold based on median score $\bar{\mu}$ and the standard deviation of the similarities. All model updates that are beyond that threshold (below or above depending on the position of the mean with respect

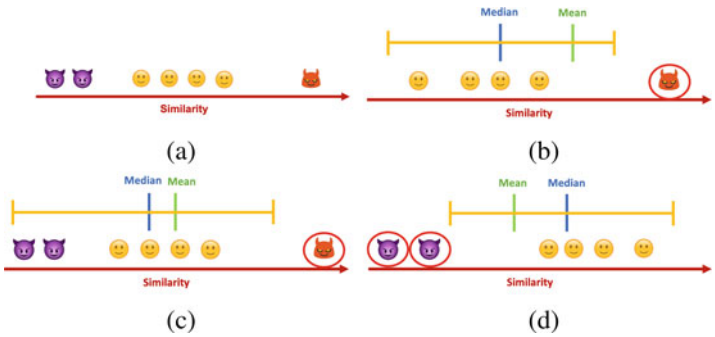

Fig. 16.3 (**a**) Considering that the benign parties are in majority during the training, they tend to send "similar" updates. (**b**) AFA calculates the median and mean of estimated similarity values of parties updates with the aggregated update. Parties whose similarity values are at a distance greater than a threshold based on these mean and median values of similarity can be identified easily in case of a single attacker. (**c**–**d**) AFA allows to detect different groups of attackers with different objectives by iteratively removing bad model updates at each training round

to the median) are considered as malicious. Then, the global model is recomputed and the procedure repeated until no model updates are considered as malicious. This iterative process allows to identify different types of attacks that can occur simultaneously, as described in Fig. 16.3. Finally, the probability p_{k_t} of each party is updated using the HMM accordingly for each party, depending on whether the model update at current training iteration was considered as malicious or not. If a party consistently sends malicious model updates, AFA includes a mechanism to block the user based on the beta posterior probability distribution used to model the parties' behavior.

Compared to Krum, AFA is more scalable, as it only needs to compute the similarity for each party model update with respect to the aggregated model, whereas Krum requires to compute the similarities among the model updates from all the parties. On the other side, compared to Krum and median-based aggregation rules, AFA enables the detection of the malicious parties and improves the communication efficiency by blocking parties that consistently send malicious model updates.

16.3.2.2 PCA

An alternative use of history was proposed in [41] to specifically combat against label flipping attacks. Given an update, for each output class, change (or delta) in the corresponding row in the final neural network layer is extracted and a history over many communication rounds is stored for each output class. From this history, the deltas are projected down to 2D via PCA, and the authors show that malicious and benign parties form well separated clusters.

16.3.2.3 FoolsGold

Along the same line, Fung et al. [15] devise a defense strategy based on comparison of historical updates between multiple parties. The algorithm works under the assumption that update from malicious parties tend to have similar and less diverse updates than those of honest parties. Cosine similarity is used to compare the histories of different participants and the party updates are rescaled to reflect the confidence before the subsequent aggregation.

16.3.2.4 LEGATO

Varma et al. [42] propose a fusion algorithm that can mitigate the effect of malicious gradients in various Byzantine attacks setting to train neural networks in FL. In particular, it analyzes the change of the norm of the gradient per layer and employs a dynamic gradient reweighing scheme based on layer-specific robustness computed based on the gradient analysis. Details about LEGATO can be found in Chap. 17.

16.3.3 Redundancy-Based Defenses

Thus far, the defenses we have discussed rely on improving the aggregation mechanism. However, an alternative line of proposed defensive methods function based on redundancy [13, 34, 38]. These defenses function by replicating data across several devices, so that each data partition is seen by at least R parties. If $R \geq 2S + 1$, where S is the number of malicious parties, then, by simple majority vote, the true update can be recovered. An example of this is illustrated in Fig. 16.4. The difficulty is that naively replicating data across R parties and having each party send R gradient updates corresponding to each replicated portion of data are computationally expensive. Thus, approaches have considered encoding all the gradients computed at each party. Then, the encoded representation is sent, and the individual gradients at the aggregator are then decoded [13]. Or, in [34], a hierarchical scheme was considered when combined with robust aggregation. More precisely, parties are assigned into groups and parties within the same group all perform the same redundant computation. The results from different groups are then hierarchically combined into a final model.

In general, redundancy-based defenses can be extremely strong and come with rigorous guarantees of the robustness offered. However, they have several significant drawbacks for application in federated (as opposed to *distributed*) learning. First, there is an inescapable communication overhead as the data will need to be replicated across devices. Second, there are privacy concerns with sharing data in such a manner. Data could potentially be anonymized prior to transmission (either by employing differential privacy or by other privacy mechanisms), and however, the risk might still be higher than not sharing data altogether.

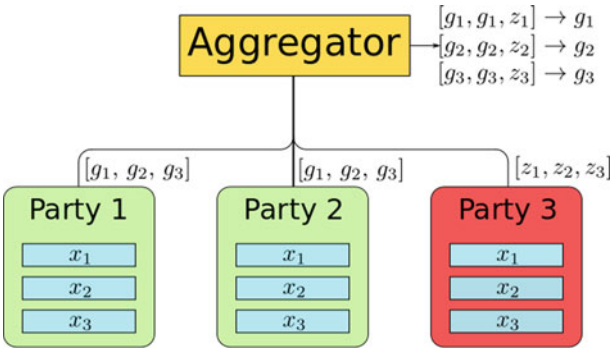

Fig. 16.4 Example of a simple redundancy-based defense. The two benign parties 1 and 2 each compute gradients g_{1-3} on data points x_{1-3}. Party 3 supplies arbitrary updates z_{1-3}. By a majority vote, the correct gradients g_{1-3} are used

16.4 Attacks

With a broad understanding of different threat models and defense strategies, we are now in a position for closer examination of specific attacks. The ability to supply arbitrary updates during model training allows an attacker to pursue a wide range of goals for data and model poisoning. Here, we categorize the types of attacks into three broad categories. First, convergence attacks which seek to indiscriminately degrade the global model performance on the underlying task. Second, in targeted attacks an adversary aims to produce errors for specific target data points or to introduce backdoors. For instance, a backdoor attacker can introduce a *key* (or trigger) into the data which will cause a machine learning model to always output an attacker chosen class when presented with the key, or alternatively backdoor task might consist in targeted misclassification for a subset of samples. Compared to targeted poisoning attacks, backdoors do not compromise the normal operation of the system, i.e., the performance of the resulting model is not affected for regular examples, and it only produces "unexpected" outputs for inputs that contain the key. Finally, we briefly discuss other attack strategies from centralized settings which naturally extend to federated setups.

Many of these attacks are specifically designed to counter certain defensive strategies. For scenarios where a defender is not employing any defense, it can be trivial to subvert a model undergoing federated learning [8]. An important dimension to consider for attack strategies is the amount of system compromise required in order to achieve the attack objective. Backdoor attacks, for instance, often require significantly lower compromise, with successful attacks needing as little as one malicious party. For cross-device setups, the frequency of attacks also affects their success rates. An attacker might control a fixed level of compromise for every selected quorum or might control a number of devices, a portion of which is selected every round of federated learning.

An alternative attack is for an adversary who crafts samples which expose the vulnerabilities of a deployed model at run-time. Machine learning models are known to be vulnerable to such *adversarial examples*. They represent indistinguishably perturbed inputs from a human standpoint that are misclassified with high confidence by a trained machine learning model [7, 16, 40]. Such attack vectors can be computed for both white-box and black-box scenarios and are even known to transfer across different models. Communication channels for model update sharing and broadcasting in federated learning could potentially expose additional surfaces for some of these white-box attacks, especially for insiders.

16.4.1 Convergence Attacks

For convergence attacks, an adversary seeks maximum damage to the model performance within the limits imposed by defensive aggregation schemes. According to the taxonomy in Sect. 16.2, these correspond to indiscriminate causative attacks, where the attackers can manipulate the parameters of the aggregated model by providing malicious local model updates, aiming to compromise the overall model's performance.

In this line, the simplest attacks that could be performed are Byzantine attacks, as the one proposed in [8], where malicious parties send model updates with very large values, which is enough to compromise vanilla aggregation methods, such as federated averaging. Another effective way to accomplish a convergence attack is to aim for the aggregation schemes to select an update with the sign that is opposite to the true update direction. This line of research has been examined in [14, 47]. If the secure aggregation scheme being targeted is using a median-based defense, the developed attacks in [14, 47] are similar. The strategy exploits benign parties that may supply updates with the opposite sign to the mean update of the benign parties. The attacker can force their selection either by supplying updates that are larger than any benign client, thereby trying to force the selection of a positive update, or by supplying malicious updates that are less than any of the benign parties, thus trying to select a negative direction.

Example: If the benign updates $V = \{-0.2, 0.2, 0.5\}$, *then the true update mean* $\mu = 0.167$. *A simple median-based aggregation on this set would yield 0.2. However, if the attacker supplies updates smaller than min(V), the selection of a negative gradient can be obtained. The attacker submits* $V_{\text{attacker}} = \{-1, -1\}$; *now with the combined update set being* $V = \{-1, -1, -0.2, 0.2, 0.5\}$, *the median selects* -0.2.

For Krum, the two methodologies [14, 47] differ more substantially. Both methods try to deviate the chosen update vector toward the opposite sign of the true update mean μ. In [47], the formulation is similar to an attack on median-based defenses with the malicious update being

$$v_{\text{attacker}} = -\epsilon\mu \tag{16.8}$$

while in [14] the malicious update was formed via

$$v_{\text{attacker}} = w - \lambda s \tag{16.9}$$

where w is the global model, s is the sign of the direction the parameters should change by with only benign parties, and λ is our perturbation parameter.

In both cases, we would like to maximize the deviation ϵ or λ while still being selected by Krum. With [47], the deviation was manually set to a reasonably small number as to have a high selection chance. Conversely, in [14], the maximum value of λ was determined by running a binary search.

Example: With the benign updates of $V = \{0.0, 0.1, 0.25, 0.35, 0.5, 0.65\}$, *we want a negative update that is selected by Krum. If the attacker controls 3 malicious parties, we search over* $-\lambda$ *in a simple grid search and see that* $\lambda = 0.21$ *is selected. Therefore, the attacker supplies* $V_{\text{attacker}} = \{-0.21, -0.21, -0.21\}$. *The updates as seen by the aggregator are* $\{-0.21, -0.21, -0.21, 0.0, 0.1, 0.25, 0.35, 0.5, 0.65\}$ *with the minimum Krum score belonging to a party that supplied* -0.21.

Neither of those attacks considered Bulyan as a defensive method, which was instead tackled in [3]. The key observation that the authors exploited in their attack is that malicious updates can still cause significant harm by hiding in the natural spread of benign updates. The benign updates are modelled following a normal distribution with mean μ and variance σ. Then, the attacker submits updates of the form $\mu + k\sigma$. By setting k to the appropriate value, we can ensure that there are benign party updates that lie further away from the mean then the malicious updates. These parties support the selection of the malicious updates, which are selected with a high degree of probability by a robust aggregation algorithm. We can see an example for this attack in a 2D case in Fig. 16.5.

16.4.2 Targeted Model Poisoning

Having examined convergence-based attacks, we now turn our attention to model poisoning attacks which aim to be more specific in their objective. In particular, this

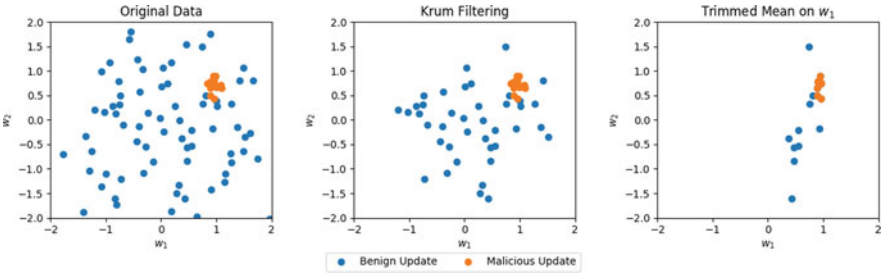

Fig. 16.5 Example illustration for the attack in [3] targeting Bulyan. Going from the plots left to right: we initially start with the distribution of two model parameters, w_1 and w_2, submitted by the benign parties (blue dots). The attackers submit model updates (orange dots) offset to the true mean, but still within the update variance. Then, in the middle plot, we apply the initial Krum filtering and we can see many malicious updates are still present. In the final plot, we apply Trimmed Mean on w_1 and a large amount of adversarial updates are included in the final averaging for w_1. An equivalent operation is also then done for w_2

involves attacks-based training a model on data that has had its feature manipulated through the insertion of backdoors or targeted label flipping attacks where particular data points are mislabelled. By performing one of these attacks, an adversary can force a model to learn attacker chosen correlations and therefore misclassify certain points at test time. We should note that this type of misclassification differs from the case of *adversarial examples*, as this results from explicit manipulations during the training process.

Both backdoor or label flipping attacks involve training models on manipulated data. One manner in which this can be achieved is if the adversary is able to tamper with the data collection process of the benign parties in a federated learning system. Thus, if manipulated data can be given to the benign parties, then the model learned through the federated learning process can be vulnerable. However, the more commonly modelled attack vector is that the adversary joins a federated learning system controlling one or more parties. The adversary then sends corrupted updates to the aggregator. This can be considered a stronger adversarial model compared to just poisoning the data that benign parties have access to, as the adversary has control over the update vector and its participation rate, and can even collude with other malicious parties to improve the attack success rate [5].

Should the adversary pursue their attack via label flipping, then the features of particular training data samples are left unchanged, but their associated labels are altered. An example of this in practice is changing the labels of all the green cars in a dataset to an attacker chosen class [1]. The model will then learn to associate green cars with the attacker class, rather than their original label. Label flipping attacks have been explored in a wide range of works [6, 15, 21, 30, 41, 44]. For label flipping attacks, it has been shown that attacking the model in the later part of training near convergence is more successfully compared to attacking the model during initial stage of training [41].

On the other hand, an adversary can mount backdoors by manipulating features, like certain pixels in the case of images, and also changing the label of the data point. Thus, a model will learn to associate the backdoor with a particular label and ignore the rest of the features in a data point if a backdoor is present. Although this is the most common attack method by which backdoors are inserted, clean label backdoor attacks in which the label is not altered are also possible [35].

Other nuances that can affect a backdoor attack depend also on the total proportion of samples in the training set that the attacker controls and wishes to affect. To continue our running example with misclassification of green cars to an attacker class, the attack will be easier if the attacker is able to control *all* the green cars in the dataset, rather than just a portion of them [39].

When attempting backdoor attacks, the challenge for an adversary depends on both:

1. The complexity of the target subtask, as an adversary might require varying numbers of malicious parties and potentially high participation frequency if the subtask has a high degree of complexity.
2. And the robust aggregation methods and anomaly detectors at the aggregator end, which need to be accounted for wen fabricating the malicious updates so as to circumvent such defenses.

In the simplest case where the aggregator uses federated averaging as the update rule, if the attacker sends their updates after training their local model on the backdoor task then as the number of parties that the attacker controls can be small in comparison to the total number of parties participating in an FL round then the backdoor updates can be cancelled out. To make these attacks effective, the work of [1] builds a strategy based on the observation that near convergence the updates sent by honest parties tend to be similar. An adversary can take advantage of this and rescale their update to ensure that the backdoor survives the eventual aggregation, thereby successfully replacing the global model with the malicious one. Specifically, with a global model w_t on round t, the attacker replaces it with their corrupted model w_{corrupt} by submitting v computed via

$$v_{\text{attacker}} \leftarrow \gamma \left(w_{\text{corrupt}} - w_t \right) + w_t \tag{16.10}$$

where γ is a scaling parameter. Independently, the work of [5] also arrives a similar rescaling strategy (explicit boosting) that accounts for the scaling at aggregation.

Example: Near convergence, a global model with parameter value of $\{4.03\}$, *might receive update from honest parties as* $0.08, 0.083, 0.09$. *The malicious model might seek to replace the parameter value with* $\{3.98\}$. *Assuming that the aggregator will combine the updates with a learning rate of* 0.015, *the adversary in this case supplies update as* $\frac{1}{0.015}(3.98 - 4.03) + 4.03 = 0.696$ *as opposed to* -0.05.

In order to supply updates that fall within the natural spread of updates received from non-malicious parties, an adversary can include additional constraints. For instance, Bhagoji et al. [5] proposes to include additional loss terms corresponding to benign training samples and regularizes the current update to be as close to the combined update from benign parties in the previous communication round. Similarly, Wang et al. [43] considers adversaries that employ projected gradient descent where the intermediate parameter states are periodically projected to an ϵ-ball around the previously received global update. Alternatively, rather than having the same backdoor key on all the data, in [46], the key is split between each malicious party according to a decomposition rule. Thus, each malicious party only inserts a part of the backdoor key into their data. The sum of all the key fragments is equal to the full backdoor key. Testing on LOAN and three image datasets shows that this approach yields better attack success rates as well as being more stealthy against FoolsGold [15] and RFA [33].

16.5 Conclusion

In this chapter, we have discussed the security of FL systems. In a similar fashion to standard, non-distributed ML systems, FL is vulnerable to attacks both at training and test time. For example, FL algorithms can be completely compromised during training just by the presence of one single malicious participant when using standard aggregation methods. Thus, the analysis of robust methods for FL is critical for the use of this technology in most practical settings.

In this chapter, we provide a comprehensive overview of different attack strategies and approaches to defend and mitigate them. However, some of the vulnerabilities of FL still need to be better understood and defending against some type of attacks remains an open research challenge. In this sense, it is also necessary to characterize and analyze further different trade-offs present in the design of FL systems, for example, a trade-off among performance, robustness and data heterogeneity or among performance, robustness, and privacy, just to cite some.

References

1. Bagdasaryan E, Veit A, Hua Y, Estrin D, Shmatikov V (2020) How to backdoor federated learning. In: Chiappa S, Calandra R (eds) The 23rd international conference on artificial intelligence and statistics, AISTATS 2020, 26–28 August 2020, Online [Palermo, Sicily, Italy], Proceedings of machine learning research. PMLR, vol 108, pp 2938–2948
2. Barreno M, Nelson B, Joseph AD, Tygar JD (2010) The security of machine learning. Mach Learn 81(2):121–148
3. Baruch G, Baruch M, Goldberg Y (2019) A little is enough: Circumventing defenses for distributed learning. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R (eds) Advances in neural information processing systems 32, pp 8635–8645. Curran

Associates. http://papers.nips.cc/paper/9069-a-little-is-enough-circumventing-defenses-for-distributed-learning.pdf
4. Bernstein J, Zhao J, Azizzadenesheli K, Anandkumar A (2018) signSGD with majority vote is communication efficient and fault tolerant. Preprint. arXiv:1810.05291
5. Bhagoji AN, Chakraborty S, Mittal P, Calo S (2019) Analyzing federated learning through an adversarial lens. In: International conference on machine learning. PMLR, pp 634–643
6. Biggio B, Nelson B, Laskov P (2012) Poisoning attacks against support vector machines. In: Proceedings of the 29th international conference on machine learning, ICML 2012, Edinburgh, Scotland, June 26–July 1, 2012. icml.cc/Omnipress. http://icml.cc/2012/papers/880.pdf
7. Biggio B, Corona I, Maiorca D, Nelson B, Srndic N, Laskov P, Giacinto G, Roli F (2013) Evasion attacks against machine learning at test time. In: Blockeel H, Kersting K, Nijssen S, Zelezný F (eds) Machine learning and knowledge discovery in databases - European conference, ECML PKDD 2013, Prague, September 23–27, 2013, Proceedings, Part III, Lecture notes in computer science, vol 8190. Springer, pp 387–402
8. Blanchard P, Guerraoui R, Stainer J et al (2017) Machine learning with adversaries: Byzantine tolerant gradient descent. In: Advances in neural information processing systems, pp 119–129
9. Bonawitz K, Ivanov V, Kreuter B, Marcedone A, McMahan HB, Patel S, Ramage D, Segal A, Seth K (2017) Practical secure aggregation for privacy-preserving machine learning. In: Thuraisingham BM, Evans D, Malkin T, Xu D (eds) Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, CCS 2017, Dallas, TX, October 30–November 03, 2017. ACM, pp 1175–1191
10. Buehler H, Gonon L, Teichmann J, Wood B (2019) Deep hedging. Quant Financ 19(8):1271–1291
11. Castro RL, Muñoz-González L, Pendlebury F, Rodosek GD, Pierazzi F, Cavallaro L (2021) Universal adversarial perturbations for malware. CoRR abs/2102.06747. https://arxiv.org/abs/2102.06747
12. Chen X, Liu C, Li B, Lu K, Song D (2017) Targeted backdoor attacks on deep learning systems using data poisoning. Preprint. arXiv:1712.05526
13. Chen L, Wang H, Charles Z, Papailiopoulos D (2018) Draco: Byzantine-resilient distributed training via redundant gradients. In: International conference on machine learning. PMLR, pp 903–912
14. Fang M, Cao X, Jia J, Gong N (2020) Local model poisoning attacks to byzantine-robust federated learning. In: 29th {USENIX} security symposium ({USENIX} Security 20), pp 1605–1622
15. Fung C, Yoon CJ, Beschastnikh I (2018) Mitigating sybils in federated learning poisoning. Preprint. arXiv:1808.04866
16. Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In: International conference on learning representations
17. Hitaj B, Ateniese G, Pérez-Cruz F (2017) Deep models under the GAN: information leakage from collaborative deep learning. In: Thuraisingham BM, Evans D, Malkin T, Xu D (eds) Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, CCS 2017, Dallas, TX, October 30–November 03, 2017. ACM, pp 603–618
18. Huang L, Joseph AD, Nelson B, Rubinstein BIP, Tygar JD (2011) Adversarial machine learning. In: Chen Y, Cárdenas AA, Greenstadt R, Rubinstein BIP (eds) Proceedings of the 4th ACM workshop on security and artificial intelligence, AISec 2011, Chicago, IL, October 21, 2011. ACM, pp 43–58
19. Hussain F, Hussain R, Hassan S.A, Hossain E (2020) Machine learning in IoT security: Current solutions and future challenges. IEEE Commun Surv Tutorials 22(3):1686–1721
20. Li L, Xu W, Chen T, Giannakis GB, Ling Q (2019) RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 1544–1551
21. Liu Y, Ma S, Aafer Y, Lee W, Zhai J, Wang W, Zhang X (2018) Trojaning attack on neural networks. In: 25th Annual network and distributed system security symposium, NDSS 2018, San Diego, California, February 18–21, 2018. The Internet Society

22. Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A (2018) Towards deep learning models resistant to adversarial attacks. In: International conference on learning representations. https://openreview.net/forum?id=rJzIBfZAb
23. Melis L, Song C, Cristofaro ED, Shmatikov V (2019) Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE symposium on security and privacy, SP 2019, San Francisco, CA, May 19–23, 2019. IEEE, pp 691–706
24. Mhamdi EME, Guerraoui R, Rouault S (2018) The hidden vulnerability of distributed learning in Byzantium. Preprint. arXiv:1802.07927
25. Muñoz-González L, Lupu EC (2019) The security of machine learning systems. In: AI in cybersecurity. Springer, pp 47–79
26. Muñoz-González L, Biggio B, Demontis A, Paudice A, Wongrassamee V, Lupu EC, Roli F (2017) Towards poisoning of deep learning algorithms with back-gradient optimization. In: Thuraisingham BM, Biggio B, Freeman DM, Miller B, Sinha A (eds) Proceedings of the 10th ACM workshop on artificial intelligence and security, AISec@CCS 2017, Dallas, TX, November 3, 2017. ACM, pp 27–38
27. Muñoz-González L, Co KT, Lupu EC (2019) Byzantine-robust federated machine learning through adaptive model averaging. Preprint. arXiv:1909.05125
28. Nasr M, Shokri R, Houmansadr A (2019) Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE symposium on security and privacy, SP 2019, San Francisco, CA, May 19–23, 2019. IEEE, pp 739–753
29. Papernot N, McDaniel PD, Goodfellow IJ (2016) Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. CoRR abs/1605.07277. http://arxiv.org/abs/1605.07277
30. Paudice A, Muñoz-González L, György A, Lupu EC (2018) Detection of adversarial training examples in poisoning attacks through anomaly detection. CoRR abs/1802.03041. http://arxiv.org/abs/1802.03041
31. Paudice A, Muñoz-González L, Lupu EC (2018) Label sanitization against label flipping poisoning attacks. In: Alzate C, Monreale A, Assem H, Bifet A, Buda TS, Caglayan B, Drury B, García-Martín E, Gavaldà R, Kramer S, Lavesson N, Madden M, Molloy I, Nicolae M, Sinn M (eds) ECML PKDD 2018 Workshops - Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, September 10–14, 2018, Proceedings, Lecture Notes in Computer Science, vol 11329. Springer, pp 5–15
32. Pierazzi, F, Pendlebury, F, Cortellazzi, J, Cavallaro, L (2020) Intriguing properties of adversarial ML attacks in the problem space. In: 2020 IEEE symposium on security and privacy, SP 2020, San Francisco, CA, May 18–21, 2020. IEEE, pp 1332–1349
33. Pillutla VK, Kakade SM, Harchaoui Z (2019) Robust aggregation for federated learning. CoRR abs/1912.13445. http://arxiv.org/abs/1912.13445
34. Rajput S, Wang H, Charles Z, Papailiopoulos D (2019) Detox: A redundancy-based framework for faster and more robust gradient aggregation. Preprint. arXiv:1907.12205
35. Shafahi A, Huang WR, Najibi M, Suciu O, Studer C, Dumitras T, Goldstein T (2018) Poison frogs! targeted clean-label poisoning attacks on neural networks. Preprint. arXiv:1804.00792
36. Shah D, Dube P, Chakraborty S, Verma A (2021) Adversarial training in communication constrained federated learning. Preprint. arXiv:2103.01319
37. Shen L, Margolies LR, Rothstein JH, Fluder E, McBride R, Sieh W (2019) Deep learning to improve breast cancer detection on screening mammography. Sci Rep 9(1):1–12
38. Sohn Jy, Han DJ, Choi B, Moon J (2019) Election coding for distributed learning: Protecting signSGD against byzantine attacks. Preprint. arXiv:1910.06093
39. Sun Z, Kairouz P, Suresh AT, McMahan HB (2019) Can you really backdoor federated learning? Preprint. arXiv:1911.07963
40. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow IJ, Fergus R (2014) Intriguing properties of neural networks. In: Bengio Y, LeCun Y (eds) 2nd International conference on learning representations, ICLR 2014, Banff, AB, April 14–16, 2014, Conference Track Proceedings. http://arxiv.org/abs/1312.6199

41. Tolpegin V, Truex S, Gursoy ME, Liu L (2020) Data poisoning attacks against federated learning systems. In: European symposium on research in computer security. Springer, pp 480–501
42. Varma K, Zhou Y, Baracaldo N, Anwar A (2021) Legato: A layerwise gradient aggregation algorithm for mitigating byzantine attacks in federated learning. In: 2021 IEEE 14th international conference on cloud computing (CLOUD)
43. Wang H, Sreenivasan K, Rajput S, Vishwakarma H, Agarwal S, Sohn Jy, Lee K, Papailiopoulos D (2020) Attack of the tails: Yes, you really can backdoor federated learning. Preprint. arXiv:2007.05084
44. Xiao H, Xiao H, Eckert C (2012) Adversarial label flips attack on support vector machines. In: Raedt LD, Bessiere C, Dubois D, Doherty P, Frasconi P, Heintz F, Lucas PJF (eds) ECAI 2012 - 20th European conference on artificial intelligence. Including prestigious applications of artificial intelligence (PAIS-2012) System demonstrations track, Montpellier, August 27–31, 2012, Frontiers in artificial intelligence and applications, vol 242. IOS Press, pp 870–875
45. Xie C, Koyejo O, Gupta I (2018) Generalized byzantine-tolerant SGD. Preprint. arXiv:1802.10116
46. Xie C, Huang K, Chen PY, Li B (2019) Dba: Distributed backdoor attacks against federated learning. In: International conference on learning representations
47. Xie C, Koyejo O, Gupta I (2019) Fall of empires: Breaking byzantine-tolerant SGD by inner product manipulation. In: Globerson A, Silva R (eds) Proceedings of the thirty-fifth conference on uncertainty in artificial intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019. AUAI Press, p 83. http://auai.org/uai2019/proceedings/papers/83.pdf
48. Xie C, Koyejo S, Gupta I (2019) Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In: International conference on machine learning. PMLR, pp 6893–6901
49. Yin D, Chen Y, Ramchandran K, Bartlett P (2018) Byzantine-robust distributed learning: Towards optimal statistical rates. Preprint. arXiv:1803.01498
50. Zizzo G, Rawat A, Sinn M, Buesser B (2020) Fat: Federated adversarial training. Preprint. arXiv:2012.01791