H2020 – ICT-13-2018-2019

Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

Grant No 824988

D5.1 Threat analysis for federated machine learning algorithms

July 19

Imprint

Contractual Date of Delivery to the EC:	**31 July 2019**

Author(s):	**Luis Muñoz-González (Imperial College)**
Participant(s):	**Mathieu Sinn (IBM); Ángel Navia-Vázquez (UC3M)**
Reviewer(s):	**Lucrezia Morabito (COMAU); Stephanie Rossello (KUL)**

Project:	**Machine learning to augment shared knowledge in federated privacy-preserving scenarios (MUSKETEER)**
Work package:	**WP5**
Dissemination level:	Public
Version:	**3**
Contact:	**Luis Muñoz-González – l.munoz@imperial.ac.uk**
Website:	**www.MUSKETEER.eu**

Legal disclaimer

The project Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824988. The sole responsibility for the content of this publication lies with the authors.

Copyright

Executive Summary

This deliverable D5.1 – Threat analysis for federated machine learning algorithms – is the first outcome of WP5 under task T5.1. It includes a technical report describing the threat model and the vulnerabilities of federated machine learning algorithms both at training and test time. It also contains the analysis of the threats across the different Privacy Operation Modes (POMs) to be implemented for MUSKETEER platform.

Document History

Version	Date	Status	Author	Comment
1	30 Jun 2019	For internal review	Luis Muñoz	First draft
2	5 Jul 2019	For internal review	Luis Muñoz	Second draft
3	16 Jul 2019	Final version	Luis Muñoz	Updated after int. review

Table of Contents

LIST OF FIGURES 5

LIST OF TABLES 5

LIST OF ACRONYMS AND ABBREVIATIONS 5

1 INTRODUCTION 6

1.1 Purpose 6

1.2 Related Documents 6

1.3 Document Structure 7

2 TAXONOMY OF THREATS AND ATTACKS 7

2.1 Types of Attackers 8

2.2 Attacker's Capabilities 8

2.3 Attacker's Goal 10

2.4 Attacker's Knowledge 12

2.5 Attack Strategy 13

3 ATTACKS AT RUN-TIME 15

3.1 Attack Scenarios 16

3.2 Intriguing Properties of Adversarial Examples 18

3.3 Attacks at Run-time in MUSKETEER 19

4 POISONING ATTACKS 19

4.1 Attack Scenarios 21

4.2 Poisoning Attacks in MUSKETEER 23

5 BACKDOORS 24

5.1 Triggering Backdoors 26

5.2 Backdoor Attack Scenarios 28

5.3 Backdoor Attacks in MUSKETEER 29

6 USERS' COLLUDING ATTACKS 30

6.1 Attack Scenarios 31

6.2 Users' Colluding Attacks in MUSKETEER 32

7 THREAT MODEL ACROSS MUSKETEER'S POMS 33

7.1 Federated Collaborative Privacy Operation Modes 34

7.2 Privacy Operation Modes in a Semi-Honest Scenario 36

7.3 Unrestricted Data-Sharing Privacy Operation Modes 38

8 CONCLUSION 40

9 REFERENCES 41

List of Figures

Figure 1 Adversarial example in a computer vision problem 15
Figure 2 Noise patterns incorrectly identified by a machine learning system 17
Figure 3 Explaining adversarial examples 18
Figure 4 Example of data poisoning in a binary classification task 20
Figure 5 Example of a backdoor attack 26
Figure 6 Two different triggers to generate a backdoor 27
Figure 7 Colluding vs non-colluding attacks 30
Figure 8 Training in federated machine learning 33
Figure 9 Communication scheme for POM 1 (Aramis) 35
Figure 10 Communication scheme for POM 4 (Rochefort) 37
Figure 11 Communication scheme for POM 7 (Planchet) 39
Figure 12 Communication scheme for POM 8 (Dartagnan) 39

List of Tables

Table 1 Threat Model 14

List of Acronyms and Abbreviations

Abbreviation	Definition
CNN	Convolutional Neural Network
DNN	Deep Neural Network
IoT	Internet of Things
MLaaS	Machine Learning as a Service
POM	Privacy Operating Mode
SVM	Support Vector Machine

1 Introduction

1.1 Purpose

Machine learning systems are vulnerable and can be the objective of attackers who can exploit these vulnerabilities to conduct illicit and highly profitable activities [Huang et al. 2011], [Muñoz-González, Lupu 2019]. The main objective in WP5 is to provide mechanisms to analyse and enhance the security of the machine learning algorithms used in MUSKETEER under the different POMs.

This deliverable, D5.1, includes a technical report describing the taxonomy and the threat model to characterise the possible vulnerabilities of machine learning algorithms in federated environments, considering both attacks at training and run-time. This threat model is useful to define requirements for the design, deployment and testing of federated machine learning algorithms. In this report we consider the security threats that involve the machine learning models in MUSKETEER, both at training and test time, and the manipulation of the data or the information used to train the machine learning algorithms in the platform. In some of these scenarios, the attacker may require exploiting software vulnerabilities in the system, which is out of the scope of this report. The aspects related to software security in MUSKETEER are covered in WP3 (especially in task T3.4) and WP7. Nevertheless, in our report we also describe those scenarios where attacks against the machine learning models are possible by exploiting these software vulnerabilities.

1.2 Related Documents

This deliverable will serve as a reference framework for the rest of the deliverables in WP5: D5.2- D5.6, where the taxonomy and attacker’s models defined here can be used to define different attack strategies to test the security of the algorithms used in MUSKETEER, as well as to define and prioritise defensive strategies capable of mitigating the effect of such attacks.

D5.1 will also serve as a reference in tasks T6.1 and T6.3 in WP6. Thus, D5.1 will provide a reference to design an assessment framework of the algorithms used in the platform taking their security into account, which is relevant to deliverables D6.1 and D6.3.

Some of the threats described in this document require the attacker to first exploit software vulnerability in the platform. Thus, this document can also help to characterise the implications and the software security requirements of MUSKETEER in WP3 (deliverables D3.3 and D3.4).

1.3 Document Structure

The rest of the document is organised as follows: In Section 2 we provide the taxonomy of the threats and possible attacks against machine learning systems. In Section 3 we introduce evasion attacks, those where the attacker aims to produce errors at test time, when the algorithm is deployed. Section 4 describes poisoning attacks, i.e. those produced at training time. In Section 5 we describe backdoor attacks, where the attacker aims to manipulate the model to produce misbehaviour for specific data points. In Section 6 we will describe coordinated attacks where a group of malicious users aim to manipulate the behaviour of the algorithms used in the platform. In Section 7 we analyse the different threats for the different POMs in MUSKETEER. Finally, Section 8 concludes the report.

2 Taxonomy of Threats and Attacks

Machine learning is at the core of many modern applications, extracting valuable information from the data gathered from many different sources and allowing the automation of many tasks. Machine learning has been successfully applied in many different application domains, including health, manufacturing, industrial control systems, or computer and system security, to cite some.

Despite their benefits, machine learning algorithms can be abused, providing new opportunities to cyber-criminals to compromise systems by exploiting the vulnerabilities of machine learning algorithms. In fact, machine learning itself can be the weakest link in the security chain, as often, the learning algorithms are not design with security in mind. Far from a theoretical hypothesis, these attacks have been already reported in the wild against anti-virus engines, spam filters, or fake news and profile detection, among others [Muñoz-González, Lupu 2018].

As in any other traditional security context, for understanding the vulnerabilities and to provide a systematic framework to analyse the security aspects of machine learning, we need to define an appropriate threat model. In this section we propose a threat model based on the frameworks originally proposed in [Barreno et al. 2010], [Huang et al. 2011] and extended in [Biggio et al. 2014], [Muñoz-González et al. 2017], [Muñoz-González, Lupu 2019], taking special consideration to the federated machine learning scenarios we have in MUSKETEER.

The threat model characterises the attacks according to the attacker's goal, capabilities to manipulate the data and to influence the learning system, knowledge of the system and the data used to train the algorithms, and the attacker's strategy.

This framework encompasses the different attack scenarios described in Sections 3-6, i.e. poisoning attacks (or attacks at training time), evasion attacks at run-time, backdoor attacks or coordinated attacks with users' collusion.

2.1 Types of Attackers

Before introducing the threat model according to the attacker's goal, capabilities, knowledge, and strategy, we need to define the types of attackers that we can consider in federated machine learning scenarios. In non-distributed machine learning algorithms, the attacker is considered as someone external to the system that aims to degrade system's performance, to produce some intentional error or to leak information from the targeted system. However, in federated machine learning algorithms, some of the users of the platform can also behave maliciously.

For the federated machine learning algorithms that we are using in MUSKETEER we can categorise the attackers as:

- **Outsiders**: this includes attackers that are not users of the platform. These attackers can compromise the performance of the system at run-time, by exploiting the weaknesses and blind spots of the algorithms, or at training time, where attackers can compromise the datasets of some of the users of the platform. This could include also cases where the attacker is capable of intercepting and tampering with the communications between the central node and some of the users.
- **Insiders**: this includes cases where one of some of the users of the platform are malicious and aim to degrade the system performance to take some advantage with respect to other users or aim to leak information from the datasets used by the other users. In the case of several insiders we can consider cases where each attacker works in isolation or scenarios where several malicious users collude towards the same malicious objective.

2.2 Attacker's Capabilities

We can define the capabilities of the attacker to compromise a machine learning system from the attacker's influence on the data used by the learning algorithms and on the constraints to manipulate the data.

Attack influence

According to this aspect, the attack can be classified as:

- **Causative**: if the attacker can influence the learning algorithm by injecting or manipulating the data used to train the learning algorithms or providing malicious information to manipulate the parameters of the system. These attacks are commonly referred to as **poisoning attacks**.

- **Exploratory**: the attacker cannot influence the training process but can attempt to exploit the weaknesses or blind spots in the system at test time. These attacks are usually known as **evasion attacks**.

Poisoning attacks are possible in scenarios where the data collected to train the learning algorithms is untrusted. For example, this can happen if the data is collected from sensors that can be compromised, people that can lie or can deliberately label the data incorrectly, or other devices whose integrity can be at risk. In the case of MUSKETEER, in this sense, we can also consider cases where some of the users of the platform are malicious and aim to degrade the system's performance, providing malicious data or sending wrong information to update the model's parameters.

In evasion attacks, even if the data used for training the algorithm is trusted, attackers can probe the system to learn and exploit its weaknesses to produce intentional errors at run-time. It has been shown that learning algorithms are vulnerable to ***adversarial examples***, inputs that are indistinguishable to genuine data points but that, when used to test the learning algorithms, produce unexpected outcomes. Exploratory attacks can also include other scenarios where the objective of the attacker is to obtain information about the machine learning model deployed or the data used for training which constitutes a privacy violation. Recently, it has been shown that even federated machine learning algorithms can be vulnerable to information leakage [Melis et al. 2019]. Thus, attackers can infer properties from the training data used by other users in a federated machine learning platform by looking at the model updates during training. However, some of the POMs used in MUSKETEER can prevent or mitigate these attacks, as the users do not have access to the model's updates at training time.

Data Manipulation Constraints

The attacker's capabilities may also be limited by the possible presence of constraints for the manipulation of the data to craft the attacks. This is strongly related to the particular application domain. For example, in malware classification task where the attacker aims to evade detection, the manipulation of the attacker's malware to achieve her/his goal needs to preserve the malicious functionality of the program. In contrast, in a computer vision tasks, it is reasonable to assume that the attacker can manipulate every pixel in an image or every frame in a video.

In data poisoning scenarios, the attacker may be in control of the labelling process. Thus, she/he can control the labels assigned to the poisoning points injected. In other cases, even if the attacker is not in control of the labelling process, she/he can figure out the possible label that will be assigned to the injected malicious points. For example, in a spam filtering application, it is reasonable to assume for the attacker that her/his malicious email injected in the system will be labelled as spam. In other settings, the attacker can include self-imposed constraints to evade detection. For example, in poisoning attacks, if the attacker is sloppy and the malicious points injected in the training data are very different from the genuine data points, we can use pre-filtering techniques, such as outlier detection [Paudice et al. 2018a] or label sanitisation [Paudice et al. 2018b], to mitigate the effect of such attacks.

Modelling realistic data constraints is necessary to characterise reasonable worst-case scenarios through optimal attack strategies, where the attacker aims to maximise the damage on the system but remaining undetected. In MUSKETEER these analyses will be part of tasks T5.2, T5.3, and T5.4.

MUSKETEER provides some data manipulation constraints defined by the task owner, who can define a set of valid values for the different features that describe the data points to be used in the learning task. However, these constraints can be loose to limit the effect of possible attacks against MUSKETEER's algorithms. However, in WP5 we will investigate and propose mechanisms to limit the attacker's capabilities to manipulate the data through data pre-filtering and outlier detection.

In MUSKETEER, in scenarios where some of the users are malicious, the attacker's constraints can be loose for some of the POMs. Thus, in these cases, the malicious users can manipulate directly the information sent to the central node to update the model's parameters (for example by sending gradients). Then, data manipulation constraints are not really applicable here, as the user can have more degrees of freedom to send (malicious) updates to the central node. In contrast, in other POMs the users are not involved in this exchange of information, as their data is securely stored in a trusted server. Then, in these cases, the attackers can only manipulate the model through manipulation of their datasets.

2.3 Attacker's Goal

The goal of the attacker can be categorised according to the intended security violation and the specificity of the attack. Additionally, in some tasks, such as in multi-class classification, the attacker's goal can also be described in terms of the specificity of the errors to be produced in the system.

Security Violation

According to this aspect, we can distinguish three different security violations against machine learning systems:

- **Integrity violation**: this happens when the attack evades detection without compromising system's normal operation.
- **Availability violation**: this includes scenarios where the attacker aims to compromise the functionality of the system.
- **Privacy violation**: this occurs when the attacker obtains private information about the machine learning 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 or not the training of the learning algorithm. On the privacy side, MUSKETEER provides different POMs that can prevent some privacy violations, as in cases where some users aim to infer properties of the datasets provided by the other users. However, when the model is deployed, if privacy-preserving algorithms are not used, attackers can infer some properties of the data used for training by probing the system.

Attack Specificity

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

- **Targeted attacks**: if the attacker aims to degrade the performance of the system or to produce errors for a reduced set of data points / cases.
- **Indiscriminate attacks**: if the attacker aims to produce errors or degrade system's performance for a broad set of cases or data points, i.e. in an indiscriminate fashion.

The federated algorithms used in MUSKETEER can be vulnerable to both, indiscriminate and targeted attacks.

Error Specificity

Depending on the nature of the errors the attacker wants to produce in the system, we can categorise the attacks as:

- **Error-generic**: when the attacker aims to produce errors in the system regardless of the type of error to be produced. For example, in a facial recognition system the attacker wants subject A not to be recognised by the system, regardless if the system misrecognised this subject as subject B, C, etc.
- **Error-specific**: when the attacker wants to produce specific errors in the system. Following the previous example, in an error-specific attack the attacker may want subject A to be recognised as subject B.

Error-specificity is application dependant. Actually, depending on the attacker's capabilities the attacker can be constrained on the kind of errors to be produced in the system. In MUSKETEER, the error-specificity of possible attacks will depend on the definition of the task.

2.4 Attacker's Knowledge

The knowledge of the attacker about the target machine learning system includes the following aspects:

- The **dataset** used for training the learning algorithm.
- The **features** used to train the learning algorithm and their range of valid values.
- The **learning algorithm** and the **objective function** to be optimised by the learning algorithm.
- The **parameters** of the machine learning algorithm.

Considering these aspects, typically we can consider two different scenarios: perfect and limited knowledge attacks.

Perfect Knowledge Attacks

Although unrealistic in most cases, perfect knowledge attack scenarios assume that the attacker knows everything about the targeted system. However, perfect knowledge attacks allow to evaluate the security of machine learning algorithms in worst-case scenarios. These attacks can help to estimate the degradation of the system when it is under attack, which can be useful for model selection, comparing the robustness and performance of different machine learning algorithms when tested against perfect knowledge attacks.

Limited Knowledge Attacks

Although these scenarios include a broad range of possibilities, typically, in the research literature, two main cases 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, or the objective function optimised by the learning algorithm. However, the attacker does not have access to the training data, although she/he has access to a surrogate dataset with similar characteristics to the dataset used in the targeted system. Then, the attacker can estimate the parameters of the targeted model by using this surrogate dataset, which can enable quite successful attacks (depending on the similarity between the two datasets).
- **Limited knowledge attacks with surrogate models**: this includes scenarios where the attacker have access to the dataset and the feature representation used in the targeted system, but she/he does not have access to the machine learning model and the objective function to be optimised by the learning algorithm. In these cases, the attacker can train a surrogate model to estimate the behaviour of the targeted system, crafting attacks against this surrogate model. Then, the resulting malicious data points targeting this surrogate model are tested against the real model. It has been shown that this strategy can be effective to achieve successful attacks, especially if the targeted and the surrogate models are similar. This is known as attack transferability [Papernot et al. 2016a].

Although perfect knowledge attacks can allow to model worst-case scenarios, in the case of federated machine learning algorithms, this may not be the best strategy to test the robustness of the algorithms against different attacks. Thus, it can be more interesting to test federated machine learning algorithms using limited knowledge attacks where we assume that the attacker (or attackers) are in control of part of the dataset used in to train the algorithms, i.e. one or some of the users of the platform are malicious or their data have been compromised. This strategy can be especially relevant to test the security of the algorithms used in MUSKETEER against coordinated attacks with users' collusion.

2.5 Attack Strategy

Attack strategies can be formulated as an optimisation problem that captures different aspects of the threat model. Then, given the attacker's knowledge and the set of samples that the attacker aims to inject in the training dataset or for which the attacker aims to produce some (specific or non-specific) errors, the attacker's goal can be characterised as an objective function evaluated in this specific set of samples. This objective function helps to assess the effectiveness of the attack strategy with respect to this set of malicious points. This formulation is valid for both attacks at training and test time (i.e. poisoning and evasion attacks).

For poisoning attacks, in MUSKETEER we also need to consider attack strategies where the attacker is capable of manipulating the parameters of the model directly (for example by sending wrong parameter updates to the central node). In this case, the attacker needs a test set to evaluate the effectiveness of the attack, as the attacker is not necessarily manipulating a subset of the dataset used to train the learning algorithms.

In Table 1 we summarise the threat model described in this section.

Table 1 Threat Model

Attacker's Capability	**Attack Influence:** 1) *Causative attacks*: the attacker can influence the learning algorithm. 2) *Exploratory attacks*: attacker can only manipulate data at run-time.
	Data Manipulation Constraints: application dependant. Attackers may be limited to manipulate the features or the labels of the data. Additional constraints can be self-imposed by the attacker to evade detection.
Attacker's Goal	**Security Violation**: *1) Integrity attacks* *2) Availability attacks* *3) Privacy violation*
	Attack Specificity: 1) Targeted attacks: focused on specific data points. 2) Indiscriminate attacks: targeted on a broad set of data points.
	Error Specificity: 1) *Error-generic* attacks: the attacker does not care about the type of errors to be produced in the system. 2) *Error-specific* attacks: the attacker aims to produce specific types of errors.
Attacker's Knowledge	**Perfect Knowledge**: the attacker knows both the dataset and the model used in the targeted system.
	Limited Knowledge: 1) *Surrogate data*: the attacker knows the model of the targeted system but not the training dataset. 2) *Surrogate model*: the attacker knows the training dataset but not the model.

3 Attacks at Run-time

For some tasks, machine learning systems can outperform humans when tested on naturally occurring data. However, it has been shown that, at run-time, these systems fail considerably when considering the presence of an attacker [Szegedy et al. 2013]. In other words, machine learning works well when things go *as expected*, but the algorithms are brittle and can be easily broken by smart adversaries [Papernot, Goodfellow 2016].

At run-time, when the machine learning system has already been trained and deployed, attackers can look for the blind spots or the weaknesses of the system to produce intentional errors. As defined in Section 2, these attacks are often referred to as **evasion attacks**. It has been shown that many learning algorithms, especially deep networks, are vulnerable to adversarial examples [Szegedy et al. 2013], [Biggio et al. 2013], i.e. inputs indistinguishable from genuine data points that are designed to produce errors at test-time. As the perturbation that the attacker needs to introduce to create successful adversarial examples is very small, it is very difficult to automatically distinguish between malicious and benign examples.

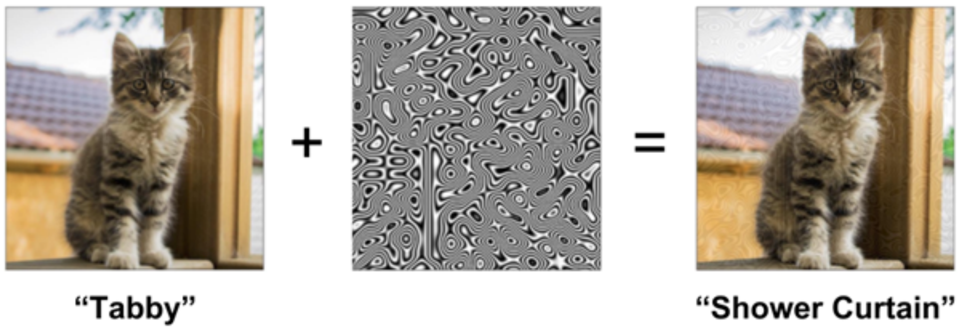

Figure 1 Adversarial example in a computer vision problem

In Figure 1 we show an adversarial example in a computer vision application [Co et al. 2019], where a deep neural network is designed to classify images amongst 1,000 different categories. The left picture shows a *tabby cat* that is correctly classified by the system. However, after adding the malicious perturbation shown in the centre of the figure, the resulting adversarial example on the right is classified as a *shower curtain* by the machine learning system. For the human eye the cat is clearly visible in the adversarial example and there is no element that suggests that there is a shower curtain in the image. This example is analogous to optical illusions, designed to deceive the human brain. However, from this simple picture we can observe that it is easier to fool a machine learning algorithm than a human.

Other types of attacks at run-time are those that aim at extracting private, confidential or proprietary information about the training data from a trained model. For example, [Fredrikson et al. 2015] demonstrate a model inversion attack which allows them to detect whether the image of a particular individual was included in the training data set for a face recognition system. [Carlini et al. 2019] demonstrate a similar attack strategy against generative sequence models, allowing them to extract sensitive personal information, such as Social Security Numbers, from training data used to train text classifiers. Recently, [Melis et al. 2019] showed that such type of information leakage can also occur for machine learning models trained in a federated fashion.

3.1 Attack Scenarios

Broadly we can differentiate two different scenarios where attackers can exploit vulnerabilities at test time:

- On one side, attackers can leverage regions of the feature space that are not supported by the training data, i.e. the resulting attack points are quite different from the points used to train the machine learning algorithm. Thus, in these regions, machine learning systems can produce quite unexpected predictions. However, these attacks can be easier to detect and mitigate by adequate data pre-filtering or by using outlier detection. Thus, points that are suspicious can be rejected by the system, for example. To illustrate these scenarios, in Figure 2 we show and example from [Nguyen et al. 2015] using a state-of-the-art deep neural network for an image classification task. All the patterns showed in Figure 2 are recognised by the machine learning system as concrete objects (see the labels below each picture) with more than 99% confidence. However, none of the patterns has any resemblance with objects from the category assigned by the machine learning system.
- On the other side, smart adversaries can leverage regions of the feature space for which the learned model differs from the true model that we would learn if we could characterise completely the underlying data distribution or, in other words, if we had an infinite number of data points for training the machine learning algorithm. This vulnerability occurs because the number of data points used for training the algorithms is limited and/or the learning algorithms have limited capacity to solve the tasks their aiming to solve (for example, when using

a linear classifier to solve a non-linear classification problem or when using a small neural network with limited expressive power). Apart from this, the presence of noise in the data can make impossible to solve perfectly a specific task, i.e. the system will naturally produce errors. Then, attackers can also leverage regions where these errors are more frequent. The adversarial example depicted in Figure 1 belongs to this category. In Figure 3 we show an example to explain this scenario for a binary classification problem, where a learning algorithm aims to classify the red dots and the yellow stars. The blue line depicts the true model we would learn if we had an infinite number of training points, and the red line represent the model learned based on the stars and dots represented in the figure. The grey areas are the regions for which the true and the learned model differ. These regions can be leveraged by attackers to craft successful attacks by slightly modifying the features of genuine data points (as shown in the example).

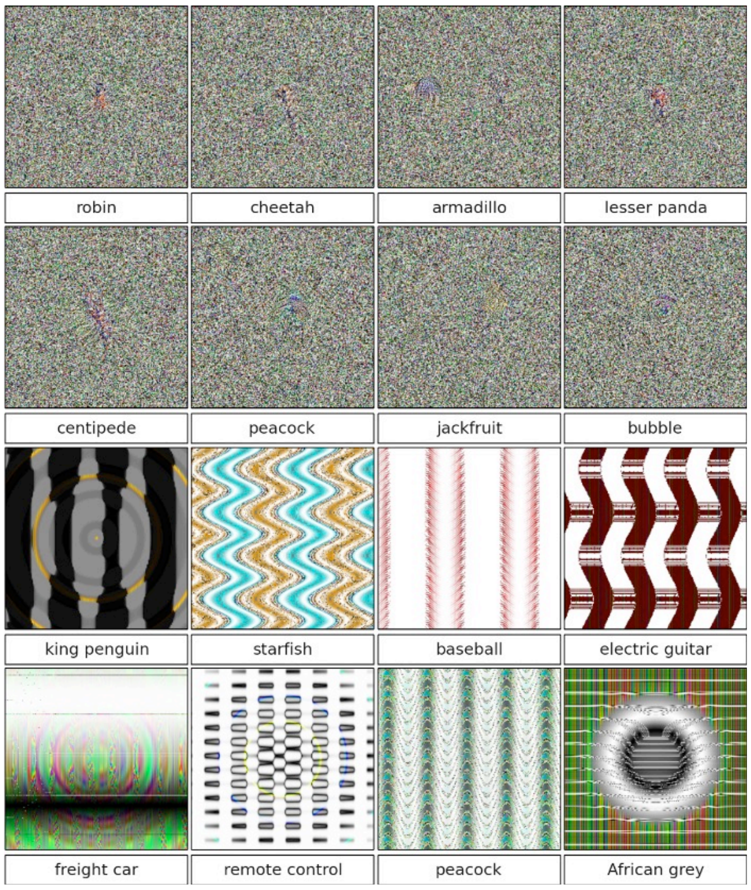

Figure 2 Noise patterns incorrectly identified by a machine learning system

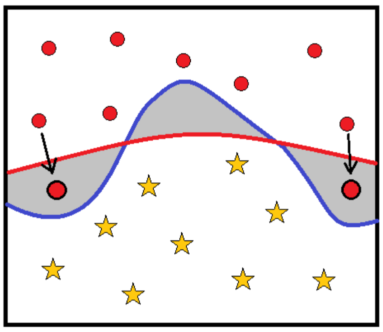

Figure 3 Explaining adversarial examples

Different white-box (such as [Szegedy et al. 2013], [Goodfellow et al. 2014] [Papernot et al. 2016b], [Carlini, Wagner 2017]) and black-box (such as [Chen et al. 2017], [Ilyias et al. 2019], [Co et al. 2019]) attacks have been proposed in the research literature, showing that machine learning algorithms are very sensitive to this threat, especially in some application domains.

3.2 Intriguing Properties of Adversarial Examples

There are two interesting and intriguing properties that can help to understand the underlying weaknesses of learning algorithms against adversarial examples. First, we have **attacks transferability** [Papernot et al., 2016]: vulnerabilities are shared across different learning algorithms. In other words, adversarial examples that are successful against a particular learning algorithm are often successful to deceive other learning algorithms, especially if they are similar. This enables black-box attacks, as the attacker can build a surrogate model to craft adversarial examples using white-box attack strategies, and then perform quite successful black-box attacks in the target system with these examples.

The second property of interest for the analysis of these vulnerabilities is the **universal** character of some adversarial perturbations. For example, [Moosavi-Dezfooli et al. 2017] showed that deep networks are vulnerable to universal adversarial perturbations, such that one malicious perturbation added to a large set of genuine examples is capable of producing errors in the learning algorithm for a large fraction of these examples. Thus, the attacker does not need to craft specific adversarial perturbations for each input; the same perturbation produces adversarial examples with a high probability. Universal adversarial perturbations suggest the existence of systemic vulnerabilities in the learning algorithms.

3.3 Attacks at Run-time in MUSKETEER

In federated machine learning scenarios, once the system is deployed the vulnerabilities at run-time are the same as in a standard (non-distributed) machine learning algorithms, i.e. there will exists the same weaknesses and blind spots that can be leveraged by the attacker to produce errors in the system by crafting adversarial examples.

Depending on the POM and the availability of the model after training, white-box attack may not be possible against MUSKETEER trained models, but there still exists the vulnerability of the system against black-box attacks, who can be perform both, by querying the target system or by crafting attack points through a surrogate model.

In MUSKETEER project, in task T5.3 (WP5) we will investigate and develop techniques to mitigate this threat by, first, testing the algorithms' robustness against different evasion attack strategies (deliverable D5.3) and, second, by developing mechanisms to enhance the robustness of the algorithms used in MUSKETEER platform against these attacks (deliverable D5.5).

4 Poisoning Attacks

Many machine learning systems rely on untrusted data collected from different data sources that may not be reliable or the integrity of the data can be compromised, such as humans, machines, sensors or IoT devices, to cite some. In many cases, data curation or cleaning is not always possible, and then, learning algorithms are trained using untrusted data. It is clear that this offers cyber criminals an opportunity to compromise the integrity of machine learning systems by performing poisoning attacks. Thus, attackers can subvert the learning process to manipulate and damage the system by injecting malicious data into the training set used by the learning algorithms. Data poisoning is one of the emerging and most relevant threats for data-driven technologies [Joseph et al., 2013]. Some of the attacker's goals in these scenarios can include:

- Reducing the overall system's performance.
- Produce specific types of errors over particular sets of instances (targeted attacks).
- Facilitate subsequent evasion attacks.
- Create backdoors (see Section 5).

For some applications, the learning algorithms are regularly re-trained to adapt to changes in the data distribution or to offer a more personalised service to new users. In these cases, the stream of new data collected is used to update the parameters of the system. Malicious users can take advantage of this to gradually poison and manipulate the system while still evading detection. In many cases, even if the malicious data injected is correctly identified by the system, its performance is still degraded when this malicious data is used to train or re-train the algorithms. For example, in spam filtering applications, learning algorithms typically aim to classify emails as *spam* or *ham* (good emails) based on the words contained in the header and the body of the emails (among other features). Attackers can poison these systems by sending spam emails that contain words typical of both, good and spam emails. Then, when the system is re-trained, to learn new forms of spam or to personalise the service, including these malicious emails in the training dataset, some of the words that were previously considered by the system as indicative of good emails will now be considered as typical of spam. Hence, after re-training possibly some good emails containing those words will be incorrectly classified as spam.

In Figure 4 we show a synthetic example with a poisoning attack on a binary classification task, where the learning algorithm aims to classify the red and the blue dots. The blue solid line depicts the decision boundary that the model would learn in the absence of attack. But, if the attacker injects a few poisoning red points (depicted as a red star) in the training set, the decision boundary learned, represented by a red solid line, is significantly different from the previous one. Comparing this scenario with the one in Figure 3, we can observe that after injecting a few poisoning points the attacker can facilitate evasion attacks at test time in this case, leveraging regions where the two models differ.

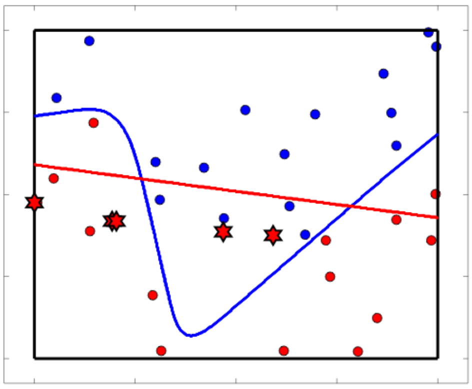

Figure 4 Example of data poisoning in a binary classification task

Different poisoning attacks have been shown in the research literature targeting different machine learning algorithms. First reported attacks relied on simple heuristics capable of compromising spam filtering applications trained with Naïve Bayes [Lowd, Meek 2005], [Barreno et al. 2010]. More systematic approaches have been described later in the literature targeting well-known machine learning algorithms, including SVMs [Biggio et al. 2012], linear classifiers [Mei, Zhu 2015], and neural networks and deep learning systems [Koh et al. 2017], [Muñoz-González et al. 2017], to cite some. Although most of these attacks are white-box, i.e. the attacker is assumed to know the details and dataset used in the targeted system, black-box attacks are also possible by using a surrogate model or a surrogate dataset. In this sense, [Muñoz-González et al. 2017] showed empirically that, as in the case of evasion, attacks are transferable, i.e. attacks that are effective against one model are often effective when tested against similar models. Recent work in [Demontis et al. 2019] provided further evidence of attack's transferability.

4.1 Attack Scenarios

Different poisoning attack scenarios can be considered according to the capabilities of the attacker to manipulate the data and the ultimate objective of the attacks. It is clear that the attacker's capabilities and the specific application settings can limit the goal of the attack.

Manipulation of labels

Some applications rely on labelled datasets, where the labels need to be manually annotated. Typically, a set of (untrusted) annotators labels the data, so that each data point is labelled by more than one annotator. Then, ***crowdsourcing*** techniques are usually applied to automatically aggregate the information by considering the annotators' skills, which are also learned by the crowdsourcing algorithm. Although some crowdsourcing algorithms, such as [Raykar, Yu 2012], allow for the detection of *spammers* (annotators that label data at random) or biased annotators, attackers can perform ***crowdturfing*** attacks, where malicious users collude to deceive the crowdsourcing algorithm [Yao et al. 2017]. These attacks are also referred to as label flipping attacks [Xiao et al. 2012], [Paudice et al. 2018b], where the features of the data remain intact and only their labels are altered. This can affect supervised and semi-supervised learning tasks, such as classification or regression. Other forms of crowdturfing are also possible in unsupervised learning algorithms, as for example, in recommender systems or social networks, where the users' reputation or items' ratings can be manipulated from the feedback provided by malicious colluding users.

Manipulation of features

In other applications it is also reasonable to assume that the attacker can manipulate different features of the data used to train the learning algorithms. Although in some cases, the attacker may not be in control of the label assigned to the malicious data points, they can infer the labels that will be possibly assigned to them. For example, in the spam filtering application described before, the attacker can reasonably assume that poisoning emails will be labelled as spam.

Depending on the attacker's capabilities we can consider three scenarios or models for the attacker:

- **Insertion model**: the attacker can add malicious samples to the training dataset but cannot modify genuine data.
- **Edition model**: the attacker can edit/manipulate the features and labels of genuine data points.
- **Deletion model**: the attacker can remove genuine data points from the training dataset that can be relevant to achieve the attacker's goal.

Manipulation of model updates

In federated machine learning scenarios there is a different possibility for performing poisoning attacks in the case of insider attackers. In these cases, the users do not share the data, but compute internally some operations that allow the central node, which coordinates the learning process, to update the parameters of the learning algorithm. For example, each user can send to the central node the gradients of the model's parameters computed on a subset (batch) of their training data. Thus, the central node (and the other users) is not aware of the training data from each user. Insider attackers can, then, manipulate the behaviour of the learning algorithm by sending malicious updates to the central node, which does not necessarily imply to modify the training dataset. In fact, these attacks provide more flexibility to the adversary to achieve her/his goal as it is possible to target specific components of the learning algorithm. For example, in deep networks, the attacker may perform attacks aiming to influence the latter layers in the network. It is clear that the effectiveness of the attack will be determined by the number of users in the platform. For example, if the number of malicious users is reduced compared to the total number of users, attacks aiming to degrade the overall system's performance will have a limited effect. However, even in these situations, successful targeted attacks are still possible.

If the communication link between the users and the central node is vulnerable, outsiders can also perform these poisoning attacks by intercepting the communication between some of the users and the central nodes. Then, the attacker can send malicious model updates to the central node on behalf of legitimate users. This can be seen as a *man-in-the-middle* attack.

4.2 Poisoning Attacks in MUSKETEER

In MUSKETEER, we need to differentiate two scenarios for data poisoning depending on the type of the attacker: insider or outsider.

Outsider attackers

In this case, the federated machine learning algorithms in MUSKETEER can be vulnerable to data poisoning in cases or applications where the data used to train the algorithms is untrusted or can be compromised. The strength of the attack will be determined by the fraction of users whose data has been compromised. For example, if only a small fraction of users has been compromised (i.e. their datasets have been compromised), indiscriminate attacks aiming to degrade the overall performance of the system will have a very limited effect. Moreover, in this case, attacks can be easier to detect, as the model updates provided by the compromised users can be significantly different compared to the ones from the non-compromised users. However, subtler (targeted) attacks are still possible and can be more difficult to detect by analysing and comparing the model updates provided by all the users in the platform.

Although the man-in-the-middle attacks described before are possible in federated machine learning scenarios, MUSKETEER platform provides security mechanisms to protect and guarantee the confidentiality in the communications between the central node and the platform users (these aspects are covered in WP3 and WP7). Thus, unless these security mechanisms are bypassed, these attacks will not be possible in MUSKETEER. Moreover, in some of the POMs, there is no communication between the users and the central node at training time (see Section 7), which removes completely the possibility of a man-in-the-middle attack.

Insider attackers

According to the number and the coordination between the insider attackers we can consider different scenarios:

- **A single malicious user**: in this case the attacker can either manipulate its own training dataset or the model updates it sends to the central node. This also depends on the MUSKETEER's POM used to perform the task, as for some of the POMs, only data manipulation is possible (see Section 7). The effectiveness of some attacks can be limited, although targeted attacks can be successful (see for example backdoor attacks in Section 5). If the attacker is sloppy and does not self-impose appropriate detectability constraints to perform the attack, in this scenario, attacks can be easier to detect by, for example, analysing the statistical properties of the model updates sent by all the users. Some of these attacks can be filtered-out, for example, with the data alignment techniques proposed and implemented in WP4 (see for example deliverable D4.2).

- **A subset of uncoordinated malicious users**: this is an extension of the previous case, where, in this case, several attackers aim to manipulate the performance of the system, but each attacker may have a different objective and the attack is not coordinated. Depending on the fraction of attackers compared to the total number of users and the alignment of the different attacker's goals, it may be more or less difficult to detect attacks based on the statistical analysis of the model updates. As before, depending on the POMs attackers can manipulate their datasets or the model updates sent to the central node.
- **Groups of coordinated malicious users**: this scenario will be analysed more carefully in Section 7. This can include two possibilities: a subset of malicious users that perform a coordinated attack or different groups of coordinated malicious users with aligned or competing objectives.

In MUSKETEER project, in task T5.2 we will investigate the vulnerabilities of the algorithms used in the platform against poisoning attacks (deliverable D5.2), as well as defensive strategies that can help to detect and mitigate the effect of such attacks (deliverable D5.4).

5 Backdoors

Similar to other settings in traditional system's security, machine learning algorithms can also be vulnerable to backdoors or *trojan attacks* that compromise the integrity of the learning algorithm. This can happen in scenarios where the data used to train the learning algorithms is untrusted (as in poisoning attacks) or where the machine learning model deployed cannot be trusted.

Training large machine learning algorithms, such as DNNs or CNNs, can be computationally very intensive, as they require using a large amount of training data and millions of parameters to be tuned to achieve good performance. This has opened different possibilities to train and deploy machine learning systems at a reduced cost, such as:

- **Machine Learning as a Service (MLaaS)**: some computing providers like Google, Microsoft or Amazon offer services to outsource the training of machine learning models to the cloud. Thus, if the service provider is compromised, the integrity of the model can also be compromised.
- **Federated machine learning systems**: if some of several users are malicious or the integrity of their data is compromised, then the machine learning system can be compromised (as explained in previous sections for the case of poisoning and evasion attacks).
- **Transfer learning**: machine learning developers can use pre-trained models designed to solve some specific tasks. These pre-trained models can be fine-tuned to solve a different (but similar) task requiring less training data and less computation. If the integrity of the pre-trained model is compromised, then, performance of the final model can also be affected.
- **Model outsourcing**: some companies or public organisations rely on proprietary models developed by external companies. In some cases, access to the training datasets used by these external companies may not be possible due to intellectual property limitations. This can hinder the detection of malicious behaviour in the learning algorithms, for example, if the dataset used to train the system is compromised. On the other side, if the external company is dishonest, the outsourced model can also be manipulated to create backdoors.

All these scenarios introduce new security risks and models can be compromised by 1) poisoning the datasets used for training or 2) directly manipulating the models. Backdoor attacks can be performed by exploiting any of these two possibilities.

More concretely, in a backdoor or trojan attack against a machine learning system, an adversary can create a maliciously trained model which has a good performance when evaluated on regular inputs or datasets, but which behaves badly when tested on specific attacker-chosen inputs [Gu et al. 2017], [Liu et al. 2017]. To achieve this goal, attackers can:

- Perform a targeted **poisoning attack**, aiming to produce errors only for a reduced (and specific) set of inputs.
- Directly **manipulate the parameters** of the model to introduce backdoors with the desired behavior for a specific set of inputs.

5.1 Triggering Backdoors

Backdoors are usually activated through a ***trigger***, i.e. a specific pattern that, when added to a genuine data point, produces the (incorrect) behaviour desired by the attacker. For effective backdoor attacks, this trigger must contain a pattern that is rare amongst genuine data points, so that the backdoor is not activated when tested on genuine inputs. At the same time, the trigger should be a pattern that can be easily included by the attacker when the system is in operation to activate the backdoors. Attackers may also want to introduce subtle triggers to remain undetected.

At the moment, most of the research literature on backdoors has focused on computer vision problems [Gu et al. 2017], [Liu et al. 2017], [Wang et al. 2019], where the triggers proposed to generate backdoors are usually specific geometric forms that are added to genuine objects or specific patterns that are added in features that provide little information for the task to be solved.

In Figure 5 we show an example for a traffic sign recognition system. Here, the attacker uses a yellow sticker as a trigger to activate the backdoor. Then, when the yellow sticker is added to a stop sign, the system misclassifies it as a speed limit sign.

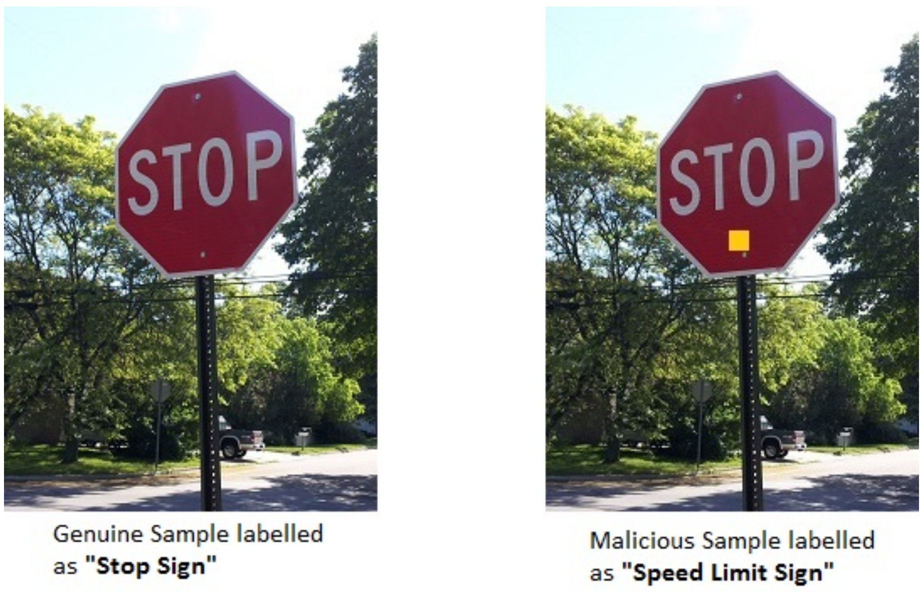
STOP
Genuine Sample labelled as "Stop Sign"
STOP
Malicious Sample labelled as "Speed Limit Sign"

Figure 5 Example of a backdoor attack

In Figure 6, we have two examples of possible triggers in a handwritten digit recognition task, using MNIST, a well-known benchmark in computer vision. In the centre of the figure the attacker just needs to manipulate a single pixel in the image to trigger the backdoor. This trigger is located in a region where, in most cases, the represented hand-written digit is not present (i.e. the selected pixel for the trigger is always black in the genuine dataset). However, as the trigger is very simple, there is a possibility that the backdoor can be activated unintendedly, for example if the image has some salt and pepper noise. To avoid this, more elaborated patterns can be preferred, as the one shown on the right plot in Figure 6. In this case, the trigger can be easily detected, as it also leverages a region of the image where there is usually no relevant information, but it is more difficult to trigger the backdoor accidentally.

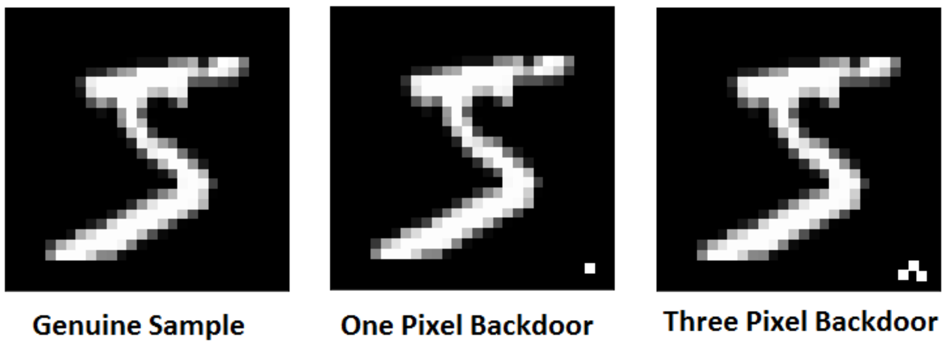

Figure 6 Two different triggers to generate a backdoor

In both Figures 5 and 6, the proposed patterns to trigger the backdoors only affect a reduced set of features, so that the objects or digits represented in the images are preserved. For example, humans can recognise the stop sign in the malicious example in Figure 5. However, the features modified by the trigger stand out when compared to genuine examples: it is not usual to have yellow stickers on stop signs.

The attacker uses these patterns to increase the effectiveness to activate the backdoor, however, these patterns can also be easier to detect. Hence, we can expect a trade-off for the attacker between the effectiveness of activating the backdoor and its detectability. It is reasonable to think that future sophisticated backdoor attacks will include subtler patterns that cannot be easily distinguished from genuine data.

5.2 Backdoor Attack Scenarios

On the attacker's side, we can consider two different mechanisms to create backdoors, which are intrinsically related to the attacker's capabilities to influence or manipulate the model:

- On one side, attackers can create backdoors by performing targeted poisoning attacks, where the adversary injects malicious data in the training set. In this case, the attacker needs to be in control of the label assigned to the malicious examples and to be able to modify the features of these examples to include the desired trigger. In federated machine learning scenarios, insider attackers can also create backdoors by sending malicious model updates to the central node.
- On the other hand, in some cases attackers can manipulate directly the parameters of the model to create the backdoors. This can happen if the integrity of the model can be compromised by an attacker (someone illicitly access and manipulates the model) or in cases where machine learning developers behave in a malicious way (dishonest developers).

On the defender's side, we can also consider two different scenarios that can lead the defender to use different strategies to try to detect and *"close"* backdoors:

- The defender has access to both the machine learning model and the training dataset (or, at least, to a fraction of the training dataset). This includes scenarios where the user (defender) trains a machine learning model relying on untrusted datasets or in cases where both, the integrity of the model and the training dataset can be compromised (e.g. if the attacker can access the machines where the model is stored). This can also include federated machine learning deployments, where users can have access to the shared machine learning model and to a fraction of the training data. In other words, the users have access to their own training dataset but not to the training data from the other users.
- The defender has only access to the trained model. This includes cases where the user outsources the model to an external company that trains the learning algorithm using their own (proprietary) dataset, which is not shared with the user. This scenario also encompasses cases where the user relies on external pre-trained models (like in trans-

fer-learning approaches) to fine-tune or deploy, the user's machine learning system.

5.3 Backdoor Attacks in MUSKETEER

Backdoor attack scenarios on MUSKETEER platform relate to those cases where attackers can only create the backdoor by poisoning the machine learning model, assuming that the attacker has not access and the possibility to directly edit or manipulate the parameters of the model after training (which would require exploiting software vulnerabilities of the machines where the machine learning models are stored).

Outsider attackers

In the case of outsider attackers, the main mechanism to create backdoors is to compromise the dataset of some of the platform users, and then, perform a targeted poisoning attack. In some of the POMs, in the remote case the attacker performs a man-in-the-middle attack and compromises the communication between some of the users and the central node, there is a possibility to create backdoors by sending malicious model updates to the central node.

Insider attackers

For insider attackers, similar to poisoning attacks, there are also two possibilities to perform backdoor attacks: 1) to manipulate the training dataset or 2) to send malicious model updates to the central node. For some of the MUSKETEER's POMs (as we will discuss in Section 7) backdoors can only be created by manipulating the training dataset, as the user has not access to the training process.

On the other side, we can also consider different scenarios according to the number of attackers and their intentions:

- Single attacker.
- Uncoordinated set of attackers where each attacker aims to create her/his own backdoor.
- Coordinated groups of attackers.

As described before, the ways to create backdoors in MUSKETEER are very similar to those required to perform poisoning attacks. Thus, in task T5.2 we will also include the investigation of the mechanisms that can allow sophisticated attackers to create backdoors (deliverable D5.2) and defensive mechanisms to detect and close them (deliverable D5.4).

6 Users' Colluding Attacks

In this section we introduce users' colluding attacks, which can be seen as a particular case of data poisoning where a group (or different groups) of malicious user aim to manipulate the model in a coordinated way. In Figure 7 we show different attack scenarios to clarify the difference between *standard* data poisoning and users' colluding attacks:

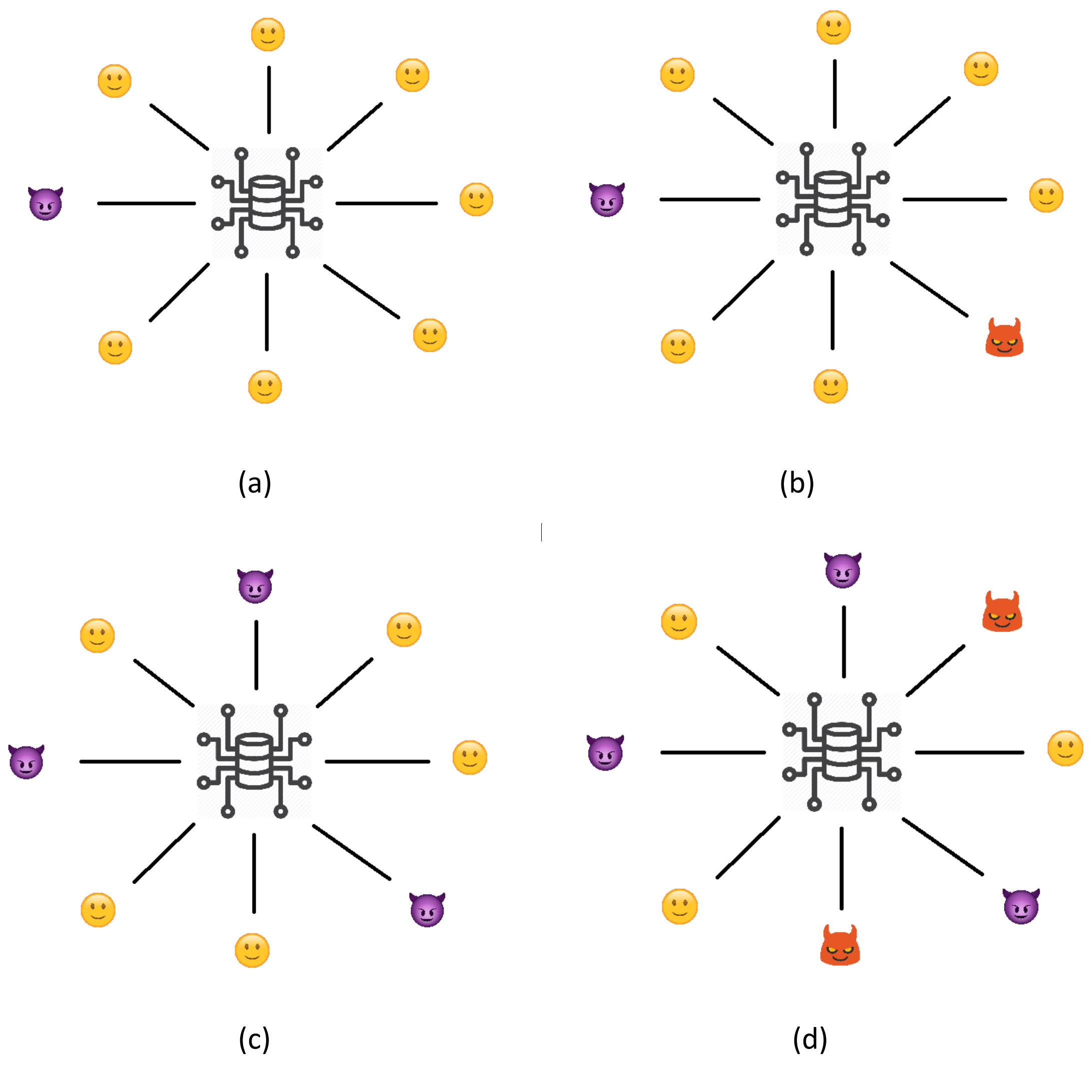

Figure 7 Colluding vs non-colluding attacks

- In Figure 7(a) we show a typical scenario for a poisoning attack where one malicious user aims to compromise the performance of the system (or the data from that user has been manipulated by an external adversary).
- Figure 7(b) represent a poisoning attack scenario with two uncoordinated attackers, i.e. each attacker has her/his own goal.
- A coordinated attack is shown in Figure 7(c), where a group of 3 malicious users collude to manipulate the model, i.e. they craft a poisoning attack with the same goal.
- A more complex scenario is depicted in Figure 7(d) where two different groups of adversaries perform poisoning attacks with separate goals.

Standard poisoning attacks with uncoordinated attackers can be easier to detect, especially if the attacker is sloppy and does not consider appropriate detectability constraints to craft the poisoning points. On the other hand, the attacker's capabilities to influence the model can be limited (e.g. in the case of indiscriminate poisoning attacks) and inversely proportional to the number of users in the platform. On the other hand, colluding attacks involving a reasonable number of malicious users can achieve a more significant damage in the target system and, at the same time, they can be more difficult to detect.

6.1 Attack Scenarios

The goal of users' colluding attacks comprises a set of different scenarios, including:

- *Indiscriminate attacks* aiming at reducing the overall performance of the machine learning system.
- *Targeted attacks* (both error-generic and error-specific) against specific subset of inputs, e.g. possible inputs from genuine users of the platform.
- Creation of *backdoors* (see Section 5).

In the case of indiscriminate and targeted attacks, we can have the same attack scenarios as those described in Section 4.1 for data poisoning. In these cases attackers can also (at least partially) reverse engineer the malicious changes they introduced in the model to recover a model with a better performance whereas the rest of the (benign) users get a degraded model.

For creation of backdoors through colluding attacks we can have the same scenarios as those described in Section 5.2.

In the research literature some approaches have already been proposed for Byzantine tolerant federated machine learning algorithms [Blanchard et al. 2017], [El Mhamdi et al. 2018], describing robust mechanisms to aggregate model updates in scenarios where some of the users send faulty or malicious updates. However, [Bhagoji et al. 2019] have shown that these defensive mechanisms can be bypassed even by a single malicious user at least in the case of targeted poisoning attacks. Then, it is reasonable to think that smart colluding attacks can also deceive these defences for more greedy goals (such as indiscriminate attacks).

6.2 Users' Colluding Attacks in MUSKETEER

In MUSKETEER users' colluding attacks follow similar considerations to those described both for backdoors and poisoning attacks. Although both insider and outsider attackers are possible, in this case it is more reasonable (and plausible) to consider insider attackers aiming to degrade or manipulate the performance of the system for the rest of the non-colluding users.

Outsider attackers may be possible, but in this case the attackers would require compromising the datasets of several users and/or intercepting the communication between the central node and these target users. Compared to standard poisoning, evasion or backdoor attacks this could require a more significant effort on the attacker's side.

For insider attackers, there are two mechanisms to craft these attacks:

- Manipulating directly the model updates sent to the central node. This option could only be possible for some of the POMs where the malicious users interchange directly information with the central node.
- Manipulating their own training datasets to perform coordinated poisoning attacks. This option enables to perform attacks against all the POMs in the platform.

In MUSKETEER, in Task T5.4 we will investigate thoroughly how colluding users can manipulate the federated machine learning algorithms used in the platform. In T5.5 we will investigate and develop mechanisms to detect and mitigate these coordinated attacks.

7 Threat Model across MUSKETEER's POMs

In this section we describe the different attack scenarios that are possible across the different POMs in MUSKETEER platform. For the sake of clarity, first, we will describe briefly how federated machine learning systems are trained.

In Figure 8 we show an example of a typical setting on a federated machine learning platform, where several users want to build a shared machine learning model without sharing explicitly their datasets. For this, there is a central node responsible for aggregating the information sent by all the users to build the model. The procedure to do this is as follows:

- The central node sends the current parameters of the model to all the users in the platform.
- The users compute locally model updates to improve the performance of the system based on their own datasets. Then, these model updates are sent to the central node. Note that the users do not need to exchange their data, but they just send information about the parameters of the model, which helps to preserve the privacy or their data.
- The central node updates the parameters of the shared model using the information sent by (all or part of) the users.
- The process is repeated until the training of the machine learning model is completed.
- When training is finished, the model can be shared across the users.

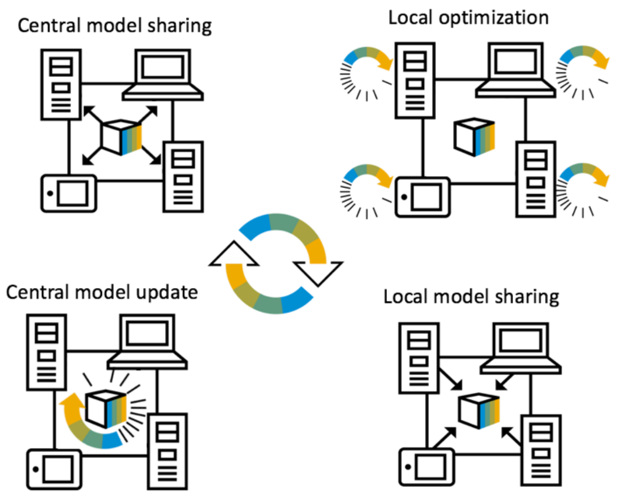

Figure 8 Training in federated machine learning

In MUSKETEER we provide variations of this standard approach to cope with different scenarios with different privacy-preserving demands. We refer to them as Privacy Operating Modes (POMs). In the remainder of this section, we will describe the possible attacks for these POMs.

As mentioned in the introduction, here we describe all possible scenarios that can let an attacker compromise the integrity of the machine learning models. Some of these attacks may require exploiting previously software vulnerabilities. These aspects are out of the scope of WP5, but are covered in WP3 (see for example T3.4) and WP7.

7.1 Federated Collaborative Privacy Operation Modes

These privacy operation modes are similar to the standard scenario described previously, where the data never leaves the data owner's facilities. Then, the shared machine learning model is shared between the central node and the users, who locally compute the model updates to be sent to the central node update the model's parameters. Figure 9 shows the communication scheme in these settings (for POM 1) in a case with two users and the central node.

In MUSKETEER we have 3 different POMs following this paradigm:

- **POM 1 (Aramis)**: Here, data cannot leave the facilities of each data owner, and the predictive models are transferred without encryption. It is intended for partners who want to collaborate to create a predictive model that will be public.
- **POM 2 (Athos)**: The same schema as Aramis but using homomorphic encryption with a single private key in every client. The server can operate in the encrypted domain without having access to the unencrypted model. In this case the predictive model can be private.
- **POM 3 (Porthos)**: Extension of Athos, where different data owners use different private keys for homomorphic encryption. The central node can transform encrypted models among different private keys.

Attacks at Training Time

In these POMs the users participate in the training process, i.e. the computation of the model updates is done on the user's side. Then, the users can manipulate not only the training data, but also the model updates they send to the central node. Then, insider attackers have more flexibility to perform poisoning attacks. This also includes scenarios with coordinated attacks with colluding users.

In the cases where the model is made available to the final users, the attackers can possibly reverse engineer the malicious changes they have produced in the system and recover the *good* model once training has finished. Then, the benign users will get a degraded version of the model, whereas the malicious users can get a better model.

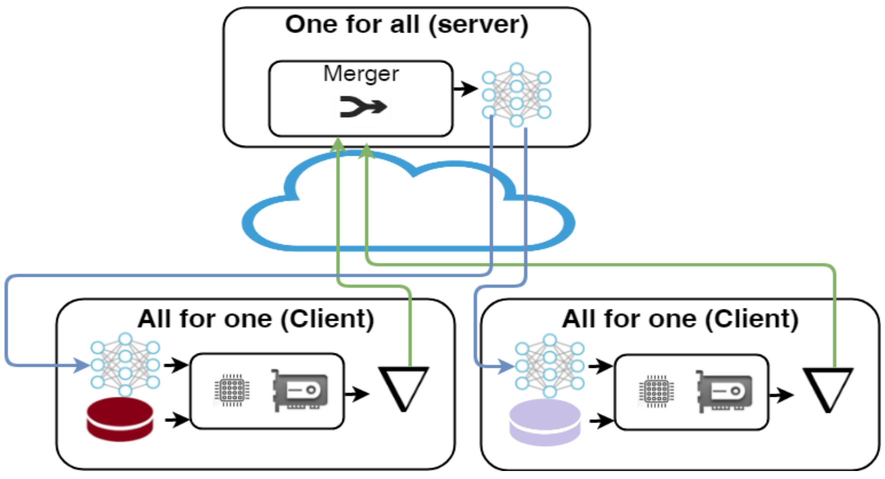

Figure 9 Communication scheme for POM 1 (Aramis)

Similar to standard poisoning attacks, backdoors are also possible both, by performing targeted poisoning attacks or by manipulating the model updates.

In the case of **insider attackers**, these POMs enable the possibility of having **dynamic attacks**, i.e. attackers can adapt their strategy and measure their success as they regularly receive the model updates from the central node, so they can monitor the training process.

Outsiders can perform attacks at training time by:

- Compromising the dataset of one or several users.
- Performing a man-in-the-middle attack, intercepting the communications between one or several users and the central node. In this case, the attacker can also adopt dynamic strategies, as she/he can monitor the training process.

Attacks at Run Time

Evasion attacks at run-time are always possible, although the ability of the attacker to succeed on her/his attack can be different depending on whether the attacker can have access to the final model or not.

If the attacker has access to the trained model, she/he can perform **white-box evasion attacks**, i.e. the attacker knows the parameters and the architecture of the final model. This can happen for insider attackers that can have access to this model (in the case it is disclosed) or for outsider attackers that steal the model by exploiting a software vulnerability in the system (in the central node or for some of the users).

If the attacker cannot access the final model, different **black-box evasion attacks** can also be possible. In these cases the attacker can:

- Build a surrogate model with a surrogate dataset and exploit attacks transferability to succeed on her/his goal.
- Query the model to look for the blind spots and craft successful adversarial examples.

Obviously the effectiveness of the black-box attacks is expected to be reduced compared to white-box settings. However, research works in adversarial machine learning have shown that even black box attacks can still be very effective, e.g. [Co et al. 2019].

7.2 Privacy Operation Modes in a Semi-Honest Scenario

In these settings, the training of the machine learning models takes place on the server side and the protection of the resulting model is at maximum. In some cases the datasets from the different users may need to leave users' facilities and be stored in a trusted external server (POM 4). The server may ask the users to compute some specific operations to complete the training model, but they never see the complete model during training. Users can only access the machine learning model when training is completed (if the model is disclosed).

In Figure 10 we show the communication schema for POM 4, where the users' data is encrypted and stored in a separate cloud server. Then, the server uses this encrypted data to train the model with no further interaction with the users.

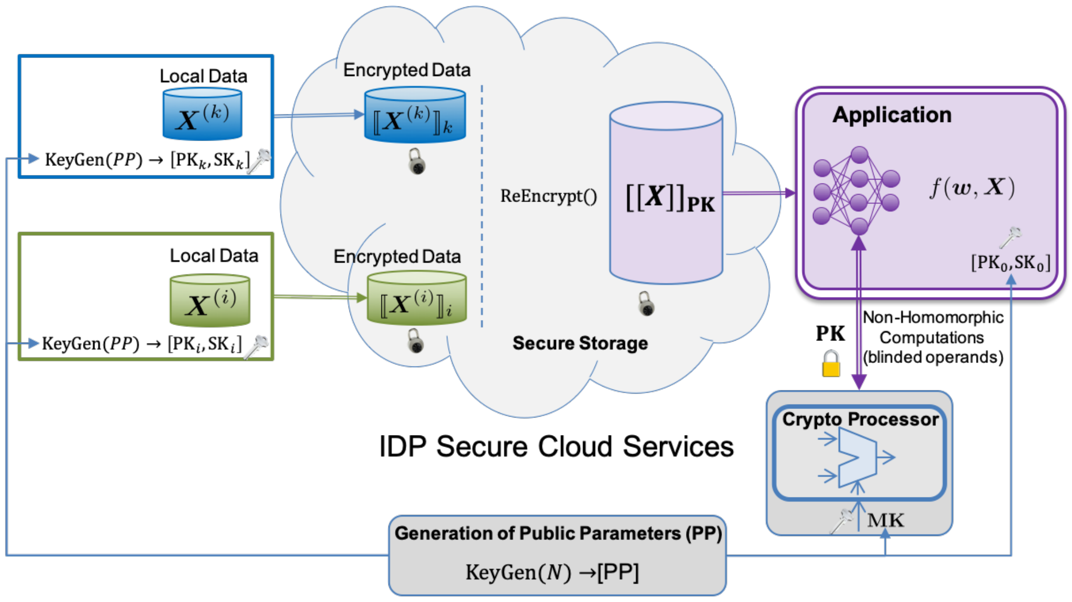

Figure 10 Communication scheme for POM 4 (Rochefort)

Following this scheme we have 3 different POMs in MUSKETEER:

- **POM 4 (Rochefort)**: In this case the platform acts as a trusted cryptographic service provider (or *master authority*) that issues the public parameters for the generation of multiple (public and secret) key. This POM uses a homomorphic cryptosystem with a double trapdoor decryption mechanism. The first decryption procedure allows a given user to decrypt cipher texts encrypted with a specific public key (local secret key), and the second one provides a master key for decrypting any cipher text, whatever its key is. To reduce the user involvement in the process, the platform also offers private cloud storage for users' encrypted data. Then, even if the data leaves users' facilities, it is securely encrypted in the cloud server.
- **POM 5 (de Winter)**: In this case a partially homomorphic cryptography method is also used, but data does not leave the users' local storage databases and data is never decrypted outside users' facilities. A proxy re-encryption process is used to avoid using the Crypto Processor described in POM 4, but now the users have to provide the result of some operations not supported by the homomorphic encryption. On the positive side, any training algorithm that can be decomposed into basic operations can be adapted to this scheme.

- **POM 6 (Richelieu)**: This setting does not require the use of encryption. Data never leaves the users' facilities, and the full model is never sent to users during training, therefore it provides a high degree of privacy. Data is processed on the client side to obtain a compact representation of information that preserves privacy according to *k-anonymity* concepts (privacy induced by averaging the operations of many users).

Attacks at Training Time

These POMs reduce the adversary's capabilities to perform (coordinated/uncoordinated) poisoning and backdoor attacks. Thus, both insider and outsider attackers can only manipulate the training data to manipulate the machine learning model. Manipulation of the model updates sent to the central node is not possible unless the cloud server is compromised.

As the attacker's cannot directly manipulate directly the model updates, the attackers cannot perform dynamic attacker, i.e. they cannot adapt their strategies during training time.

User collusion is also possible by manipulating the users' training dataset before training. As the users cannot monitor the training progress and only can access the machine learning model after training is completed (if the model is disclosed), reverse engineering the malicious changes in the final model may be more challenging than in the case for POMs 1-3, where the malicious users can get more information about the manipulation of the model during training.

Attacks at Run Time

For evasion attacks at run-time, for these POMs we have the same scenarios and considerations as in the case for POMs 1 - 3.

7.3 Unrestricted Data-Sharing Privacy Operation Modes

These POMs include more traditional approaches to train machine learning models for scenarios where the users do not have any privacy restriction concerning data sharing. This includes modes where data can leave the users' facilities using secure communication protocols (WP3) and remain unencrypted in the cloud and on the client side.

In MUSKETEER we have two POMs with unrestricted data-sharing operation:

- **POM 7 (Planchet)**: In this case the users store their datasets in an external cloud server and the machine learning models are trained on that external server without data encryption. The model can be shared with the users after training is completed. Figure 11 shows the communication scheme for this POM in a scenario with 3 users and the central node (server).

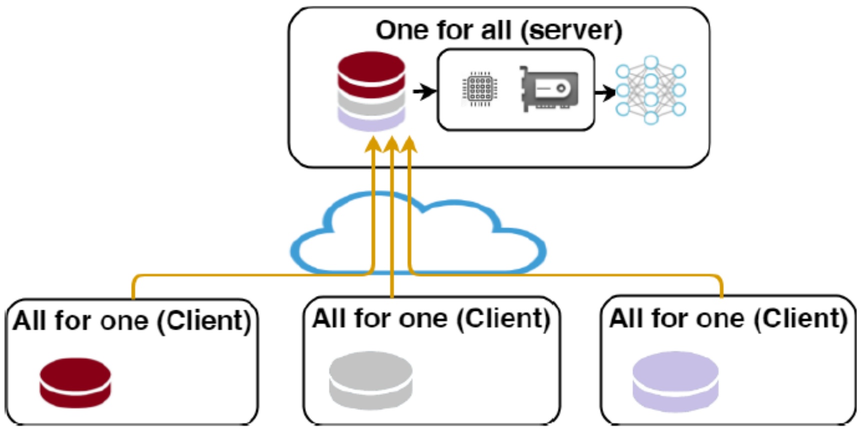

Figure 11 Communication scheme for POM 7 (Planchet)

- **POM 8 (Dartagnan)**: under this operation mode users can download different datasets and train their own machine learning models locally. In Figure 12 we show an example of this POM where 3 users share their datasets through the cloud server and one of them uses the shared dataset to build locally a machine learning model.

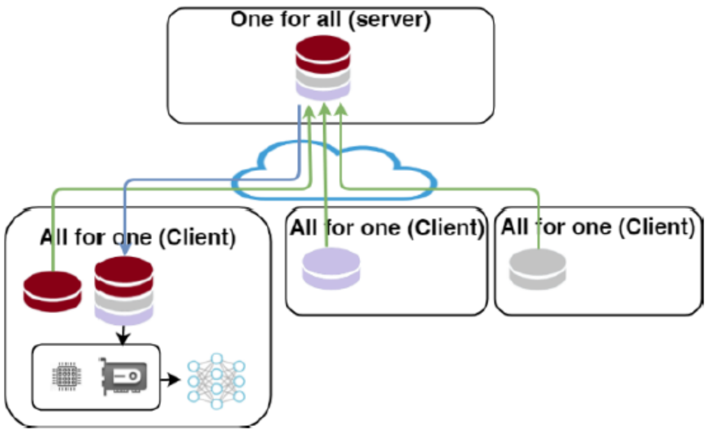

Figure 12 Communication scheme for POM 8 (Dartagnan)

Attacks at Training Time

- **POM 7 (Planchet):** As the machine learning model is computed on the external cloud and the users do not have access to the model until training is completed, manipulation of the model updates during training are not possible (unless the integrity of the cloud server is compromised and attackers can manipulate the model directly on the server). Then, poisoning and backdoor attacks are only possible by manipulating directly the training datasets provided by the users. In this case, both insider and outsider attackers are possible. As the users do not compute the model locally and cannot influence the training process (once started), then, dynamic/adaptive attack strategies are not possible. Similar restrictions apply for users' colluding attacks.

- **POM 8 (Dartagnan):**In this case only one user trains her/his own machine learning model, leveraging the data from other users. Insider attackers can include other users that may want to provide wrong or malicious data to degrade the performance of the system trained by the interested user (to perform both poisoning and backdoor attacks). Then, insiders can only influence the learning algorithm by manipulating the datasets they provide to the platform, but they cannot influence the model once the training starts, as the model is computed locally in the user's facilities. Similarly, users' colluding attacks are only possible through the manipulation of the datasets provided to the user building the machine learning model. Outsider attackers can perform both poisoning and backdoor attacks by manipulating the data or injecting poisoning points in the data sets for any of the platform users.

Attacks at Run Time

For evasion attacks at run-time, for these two POMs we have the same scenarios and considerations as in the previous cases.

8 Conclusion

We have shown that machine learning systems can be vulnerable to both attacks at training and test time. In this report, we have provided a comprehensive description of such threats including poisoning, evasion and backdoor attacks. We have also focused on specific cases that can compromise the security of federated machine learning algorithms, as it is the case of insider attackers that can collude to degrade or manipulate system's performance at training time. We have also introduced a threat model to formalise the different attack scenarios against machine learning systems, including the cases relevant to MUSKETEER. Finally, we have also described the different threats and vulnerabilities for the different POMs proposed in MUSKETEER.

This report will serve as a reference for the rest of the work in WP5, which aims to investigate and develop mechanisms to test the security of the learning algorithms to be used in MUSKETEER, as well as to propose defensive mechanisms to defend against possible attacks and mitigate the vulnerabilities of the learning algorithms.

9 References

[Barreno et al. 2010] Barreno, M., Nelson, B., Joseph, A.D., Tygar, J.D.: *The Security of Machine Learning*. Machine Learning 81(2), pp. 121–148, 2010.

[Bhagoji et al. 2019] Bhagoji, A.N., Chakraborty, S., Mittal, P., Calo, S.: *Analyzing Federated Learning through an Adversarial Lens*. International Conference on Machine Learning, pp. 634-643, 2019.

[Biggio et al. 2012] Biggio, B., Nelson, B., and Laskov, P.: *Poisoning Attacks against Support Vector Machines*. International Conference on Machine Learning, pp. 1807-1814, 2012.

[Biggio et al. 2013] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: *Evasion Attacks against Machine Learning at Test Time*. European Conference on Machine Learning (ECML/PKDD), pp. 387-402, 2013.

[Biggio et al. 2014] Biggio, B., Fumera, G., Roli, F.: *Security Evaluation of Pattern Classifiers under Attack*. IEEE Transactions on Knowledge and Data Engineering 26(4), pp. 984–996, 2014.

[Blanchard et al. 2017] Blanchard, El Mhamdi, E.M., P., Guerraoui, R., Stainer, J.: *Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent*. Advances in Neural Information Processing Systems, pp. 119-129, 2017.

[Carlini, Wagner 2017] Carlini, N., Wagner, D.: *Towards Evaluating the Robustness of Neural Networks*. Symposium on Security and Privacy, pp. 39-57, 2017.

[Carlini et al. 2019] Carlini, N., Liu, C., Kos, J., Erlingsson, U., Song, D.: *The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks.* USENIX Security (to appear), 2019.

[Chen et al. 2017] Chen, P.Y., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.J.: *Zoo: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models*. Workshop on Artificial Intelligence and Security, pp. 15-26, 2017.

[Co et al. 2019] Co, K.T., Muñoz-González, L., Lupu, E.C.: *Procedural Noise Adversarial Examples for Black-Box Attacks on Deep Neural Networks*. ACM Conference on Computer and Communications Security (CCS) (to appear), 2019.

[Demontis et al. 2019] Demontis, A., Melis, M., Pintor, M., Jagielski, M., Biggio, B., Oprea, A., Nita-Rotaru, C., Roli, F.: *Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks*. USENIX Security Symposium, vol. 28, 2019.

[El Mhamdi et al. 2018] El Mhamdi, E.M., Guerraoui, R., Rouault, S.: *The Hidden Vulnerability of Distributed Learning in Byzantium*. International Conference on Machine Learning, pp. 3518-3527, 2018.

[Fredrikson et al. 2015] Fredrikson, M., Jha, S., Ristenpart, T.: *Model inversion attacks that exploit confidence information and basic countermeasures*. ACM SIGSAC Conference on Computer and Communications Security, pp. 1322-1333, 2015.

[Goodfellow et al. 2014] Goodfellow, I.J., Shlens, J., Szegedy, C.: *Explaining and Harnessing Adversarial Examples*. arXiv preprint arXiv preprint arXiv:1412.6572, 2014.

[Gu et al. 2017] Gu, T., Dolan-Gavitt, B., and Garg, S.: *Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*. arXiv preprint arXiv:1708.06733, 2017.

[Huang et al. 2011] Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.: *Adversarial Machine Learning*. Workshop on Security and Artificial Intelligence, pp. 43–58, 2011.

[Ilyias et al. 2019] Ilyas, A., Engstrom, L., Madry, A.: *Prior Convictions: Black-box Adversarial Attacks with Bandits and Priors*. International Conference on Learning Representations, 2019.

[Joseph et al. 2013] Joseph, A.D., Laskov, P., Roli, F., Tygar, J.D., and Nelson, B.: *Machine Learning Methods for Computer Security*. Dagstuhl Perspectives Workshop 12371, Dagstuhl Manifestos, vol. 3, 2013.

[Koh et al. 2017] Koh, P.W., and Liang, P.: *Understanding Black-Box Predictions via Influence Functions*. International Conference on Machine Learning, pp. 1885-1894, 2017.

[Liu et al. 2017] Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., and Zhang, X.: *Trojaning Attack on Neural Networks*. Technical report, 2017.

[Lowd, Meek 2005] Lowd, D., and Meek, C.: *Good Word Attacks on Statistical Spam Filters*. Conference on Email and Anti-Spam, 2005.

[Mei, Zhu] Mei, S., and Zhu, X.: *Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners*. AAAI Conference on Artificial Intelligence, pp. 2871-2877, 2015.

[Melis et al. 2019] Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: *Exploiting Unintended Feature Leakage in Collaborative Learning*. IEEE Symposium on Security and Privacy, pp. 497-512, 2019.

[Moosavi-Dezfooli et al. 2017] Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., and Frossard, P.: *Universal Adversarial Perturbations*. Conference on Computer Vision and Pattern Recognition, pp. 1765-1773, 2017.

[Muñoz-González et al. 2017] Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E.C., Roli, F.: *Towards Poisoning of Deep Learning Algorithms with Back-Gradient Optimization*. Workshop on Artificial Intelligence and Security, pp. 27–38, 2017.

[Muñoz-González, Lupu 2018] Muñoz-González, L., Lupu, E.C.: *The Secret of Machine Learning*. ITNOW, 60(1), pp. 38-39, 2018.

[Muñoz-González, Lupu 2019] Muñoz-González, L., Lupu, E.C.: *The Security of Machine Learning Systems*. Book chapter in *AI in Cybersecurity*, pp. 47-79, Springer, 2019.

[Nguyen et al. 2015] Nguyen, A., Yosinski, J., Clune, J.: *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*. Conference on Computer Vision and Pattern Recognition, pp. 427-436, 2015.

[Papernot, Goodfellow 2016] Papernot, N., Goodfellow I.: *Breaking Things is Easy*. Cleverhans blog (http://www.cleverhans.io/), 2016.

[Papernot et al. 2016a] Papernot, N., McDaniel, P., Goodfellow, I.: *Transferability in Machine Learning: From Phenomena to Black-box Attacks using Adversarial Samples*. arXiv preprint: arXiv: 1605.07277, 2016.

[Papernot et al. 2016b] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., and Celik, Z.B., Swami, A.: *The Limitations of Deep Learning in Adversarial Settings*. European Symposium on Security and Privacy, pp. 372-387, 2016.

[Paudice et al. 2018a] Paudice, A., Muñoz-González, L., Gyorgy, A., Lupu, E.C.: *Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection*. arXiv preprint: arXiv:1802.03041, 2018.

[Paudice et al. 2018b] Paudice, A., Muñoz-González, L., Lupu, E.C.: *Label Sanitization Against Label Flipping Poisoning Attacks*. ECML PKDD 2018 Workshops, pp. 5-15, 2018.

[Raykar, Yu 2012] Raykar, V.C., Yu, S.: *Eliminating Spammers and Ranking Annotators for Crowdsourced Labeling Tasks*. Journal of Machine Learning Research, vol. 13, pp. 491-518, 2012.

[Szegedy et al. 2013] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R.: *Intriguing Properties of Neural Networks*. arXiv preprint arXiv:1312.6199, 2013.

[Wang et al. 2019] Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*. IEEE Symposium on Security and Privacy, pp. 530-546, 2019.

[Xiao et al. 2012] Xiao, H., Xiao, H., and Eckert, C.: *Adversarial Label Flips Attack on Support Vector Machines*. European Conference on Artificial Intelligence, pp. 870-875, 2012.

[Yao et al. 2017] Yao, Y., Viswanath, B., Cryan, J., Zheng, H., and Zhao, B.Y.: *Automated Crowdturfing Attacks and Defenses in Online Review Systems*. Conference on Computer and Communications Security (CCS), pp. 1143-1158, 2017.