UNIVERSITY OF MACEDONIA

DEPARTMENT OF APPLIED INFORMATICS

INTRUSION DETECTION SYSTEMS WITH THE USE OF FEDERATED LEARNING

UNDERGRADUATE THESIS

Andronikos Giachanatzis Grammatikopoulos (dai18122)

Supervising Professor: Ioannis Mavridis

Table of Contents

Index of Tables

Illustration Index

1. INTRODUCTION

Along with the creation of the first modern computer and especially of the Internet, two inventions that revolutionized the everyday lives of every individual and organization in the world, a new problem arose. While the communication distances were eradicated and people could communicate almost instantly with whomever they wished, cyber crime was created. Almost all online functions performed today need information from the users, in order to perform correctly, that are sometimes sensitive and should not be exposed to the public. From online banking, or a simple visit in a website, to the prediction of the next word typed in a sentence, an enormous amount of information can be used in order to provide accurate, reliable and interesting services to the user. That, of course, leaves the subject of the data exposed to malicious users that seek to steal their data and gain profit from their use, if the organization that is in charge of the processing of the data does not take the appropriate measures to protect the their integrity and confidentiality.

Over the years inventions that aim to enhance the security of organizations have been created and refined, such as devices like firewalls, Intrusion Detection and Prevention Systems, Secure by Design principles and security policies. But, it has been proven that even these products are flawed and still enable attackers to successfully inflict harm. While they limit the attack surface, there is still enough room for the attackers to perform their malicious actions. There has been an outstanding number of data leaks and breaches over the years, with the “Yahoo!” data breach that was revealed by the Washington Post in 2016 [1], being perhaps one of the most notable data breaches in history. This breach affected approximately three billion users (McMillan and Knutson 2017) [2] and allowed hackers to take possession of the users’ names, email addresses, telephone numbers, encrypted and unencrypted security questions and answers, dates of births and hashed passwords. Of course, one can imagine the magnitude of this attack and the potential harm it can do both to its users and Yahoo!. Certainly, the COVID-19 pandemic and the very fast transition to remote working and generally online activity, has enabled many attacks to take

place harming critical infrastructures, and has surfaced many unsafe products and organizations that violate fundamental security policies.

Regardless, many countries and unions have been conducting and enforcing laws and regulations that help strengthen data privacy and security. In particular, the General Data Protection Regulation (GDPR) (2016) [3] focuses on data protection and privacy in the European Union (EU) and the European Economic Area (EEA). It aims to give individuals control over their personal data via the "right to be forgotten" (that is users can choose to have their personal data deleted or withdrawn) and requires businesses to use clear and plain language for their user agreement. In addition, similar acts have been passed in the United States of America and China. For example, China's Cyber Security Law and the General Principles of Civil Law, enacted in 2017, require that Internet businesses must not leak or tamper with the personal information that they collect and that, when conducting data transactions with third parties, they need to ensure that proposed contracts follow legal data protection obligations. The establishment and the application of these regulations will help build a stronger security foundations to reduce potential data breaches and attacks.

In parallel with the advancement of the Internet and the computational power, another field has been evolving with an exploding rate, and that is Artificial Intelligence (AI). Artificial Intelligence nowadays is used in almost every device or software that we use. It has helped us minimize human effort, make predictions and describe complex phenomena. Applications like weather forecasting, assistive bots that help us solve issues in websites, self-driving cars and applications in the healthcare industry for designing drugs and recognizing malignant tumors all use Artificial Intelligence techniques, in order to create accurate and reliable results.

As technology advances, previous benchmarks that define artificial intelligence have become outdated. For example, machines that calculate basic functions or recognize text through optical character recognition (OCR) are no longer considered revolutionary as these functions are taken for granted as an inherent computer function. But 2016 is the year that AI came of age, as AlphaGo defeated the top human Go players using a deep neural network and tree search. In this complex process the huge potential of artificial intelligence was demonstrated and we have began to expect more complex and cutting-edge AI technologies. As a matter of fact, one of the most widely used technologies of Artificial Intelligence today, are Neural Networks, which aim to mimic the way that the human brain processes and learns information, and uses this knowledge in order to make decisions in the future.

One of the most fundamental driving forces of AI lately, is the explosion of Big Data, where the availability of immense amounts of data has enabled applications like AlphaGo to be developed. But one thing that limits the development of Big Data-driven Artificial Intelligence is most importantly the necessity to protect the data under process as they may contain personal and sensitive information that must not be disclosed to an untrusted party. In addition, real world scenarios have proved that with the exception of a few industries, most fields have only limited data or heavily distributed data that are of poor quality. One possible solution to that problem that was used predominantly in the last years, is the collection and process of all the data in a central location. Unfortunately, this quick solution has been proven either difficult (or even impossible in some cases) or too expensive due to the growing privacy regulations, industry competition and complex administrative processes. Additionally, distributed artificial intelligence

techniques have also been criticized for the unnecessary exposure of personal and sensitive data, which enables attackers to steal these sensitive information.

For those reasons an alternative technology was proposed as a possible solution to these problems, in order to combine the benefits of Artificial Intelligence and the big amounts of scattered data. Federated Learning centers its attention around the privacy of the data. It works by connecting the nodes that hold distributed data and tries to collaboratively train a model that doesn't expose the data outside of the device in which they are stored. While Federated Learning has an incredible potential to strengthen data privacy and security it has yet to make its impact in AI applications. In fact, while Artificial Intelligence has already been used to develop security products such as Intrusion Detection Systems, Federated Learning has yet to make its appearance in such products.

In this work I emphasize on the use of Federated Learning in the Cybersecurity industry and I have taken the initiative to develop and propose an Intrusion Detection System with the use of Federated Learning based on realistic data, in order to demonstrate its potential in real world security applications. By promoting Federated Learning I hope to enhance the ongoing reinforcement of data privacy and especially in the Cybersecurity product development process.

This thesis is structured as follows: I initially introduce the Intrusion Detection Systems, their objectives, their different classes and the ways that they accomplish their targets. Later I describe the Federated Learning technology, its goals, its categories, an alternative implementation, the ways that it enables applications to be reliable while protecting the privacy of the data used, its architecture, the participating entities and possible vulnerabilities that can sabotage its use. Moreover, I outline the process that I followed in order to develop the Intrusion Detection System with the use of Federated Learning, the used dataset and its characteristics and the experimental environment in which this application was developed. Finally, I close this work by presenting the results of the developed Intrusion Detection System, its capacity to detect threats alone and in comparison to a traditionally trained Intrusion Detection System, so that its benefits are comprehensible.

1.1 INTRUSION DETECTION SYSTEMS

Over the past decades, due to the explosive use of the Internet, there has been an overwhelming increase in network attacks that have greatly impacted negatively not only organizations but also individuals. As attacks have greatly increased also in severity, modern networked organizations require a high level of security to ensure safe and trusted communication of information, therefore Intrusion Detection Systems have become a necessary addition to any security infrastructure, in order to act as an adaptable safeguard technology for system security after traditional technologies fail. In this first section of this work, I describe briefly what Intrusion Detection Systems are, the different classes of Intrusion Detection Systems, the methodologies followed by Intrusion Detection Systems in order to accomplish their objectives, the various implementation approaches and I also provide a short comparison with Firewalls, another security technology that is widely used by organizations.

1.1.1 Overview of Intrusion Detection Systems

An *Intrusion Detection System* (or IDS) is a device or software application that automates the process of monitoring a network or a computer system and analyzing behaviors for malicious activity or policy violations. Generally, an *intrusion* is defined by NIST (Base and Mell, 2001) [5] as an attempt to compromise the Confidentiality, Integrity, Availability (CIA), or bypass the security mechanisms of a computer or network and can give rise to serious disasters to the affected party. A behavior is characterized as an intrusion when an attacker accesses the systems from the Internet, when authorized users of the systems attempt to gain additional privileges for which they are not authorized and when authorized users misuse the privileges given to them, but of course, in order to reach the conclusion that an activity is indeed an intrusion, often a large size of data must be analyzed. Therefore an Intrusion Detection System automates this monitoring and analysis process, which frequently results in quick and reliable results.

Furthermore, although Intrusion Detection Systems monitor networks for potentially malicious activity, they are also prone to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. As a normal behavior is subjective and is almost always different for every network, the security administrators of an organization must first properly set up the Intrusion Detection Systems to recognize what normal traffic looks like in that particular network as compared to malicious activity.

One important thing to note when discussing Intrusion Detection Systems, is that they are capable of only discerning a malicious behavior from a normal one, and informing the administrator or a security information and event management system. In this way, after an intrusion has occurred and the IDS has successfully recognized it, it cannot take the appropriate measures to prevent the attack. This work is performed instead by another similar technology, an *Intrusion Prevention System* (or IPS), which has all the capabilities of an IDS and could attempt to stop the ongoing attack. Although, these two technologies are very alike, it important to say that in this work I focus only on Intrusion Detection Systems, with the aim to provide an agile way to classify intrusions using Federated Learning.

1.1.2 Comparison of Intrusion Detection Systems with Firewalls

In the previous section I noted one key difference between an Intrusion Detection System and an Intrusion Prevention System. At this section I provide a brief comparison of Intrusion Detection Systems with another technology that is vital to the security infrastructure of an organization, Firewalls. Both Intrusion Detection Systems and firewalls are Cybersecurity solutions that can be deployed to enhance the security of an endpoint or an entire network. However, they differ significantly in their purposes and the way that they operate.

An Intrusion Detection System is a *passive* monitoring device (or software) that detects potential threats, describes a suspected intrusion once it has taken place and generates alerts, enabling Security Operation Center (SOC) analysts or incident responders to investigate the activity and respond to the potential incident. An IDS also is able to identify attacks that originate from within a system or a network. This is ordinarily achieved by examining network communications, by intercepting and analyzing the network

packets flowing through, identifying heuristics and patterns, which are often called signatures of common attacks (trails that are related to the attacks and are used to identify them) and taking action to only alert operators.

In contrast, a firewall is an *active* protective device (or software). It performs analysis on the metadata of network packets and allows or blocks traffic based upon predefined and static rules. It implicitly prevents intrusions, assuming an appropriate set of rules have been defined. Essentially, firewalls limit access between networks to prevent intrusions and do not signal an alert if an attack has originated from the interior of a network. Firewalls are more similar to Intrusion Prevention Systems than an Intrusion Detection System. Therefore, many Next-Generation Firewalls (NGFWs) have integreated IDS/IPS functionality, so that they can both enforce predefined filtering rules (firewalls) and detect and respond to more sophisticated threats (IDS/IPS).

1.1.3 Classification of Intrusion Detection Systems

Nowadays, there is a wide array of IDS types, ranging from antivirus software to tiered monitoring systems that follow the traffic of an entire network. The most common technologies are categorized into four major distinct classes according to where they are deployed to inspect suspicious activities and what event types they recognize. The most common categories according to Stavroulakis and Stamp 2010 [6] are *Host-based IDS* (HIDS), *Network-based IDS* (NIDS) *Wireless-based IDS* (WIDS), *Network Behavior Analysis* (NBA) and *Mixed IDS* (MIDS).

A *Host-based IDS* is most commonly deployed on a particular critical host, such as publicly accessible servers containing sensitive information and is designed to detect both internal and external threats. IDSs of this type have the ability to monitor network packets on its network interfaces, observe running processes, inspect the system logs and monitor important operating files for any suspicious activity. A host-based IDS can detect intrusions only in its host machine, decreasing the available context for decision-making , but has deep visibility into the host computer's internals. In fact, only a HIDS can analyze an end-to-end encrypted communication activity. One major disadvantage though of HIDSs is that in order for a HIDS to operate it must consume host resources and thus has significant delays in alert generation and centralized reporting.

A *Network-based IDS* is deployed often at a boundary between networks, such as in proximity to border firewalls or routers and is designed to monitor an entire protected network. It captures traffic at specific network locations and analyzes the activities of applications and protocols to recognize suspicious incidents. It has visibility into all traffic flowing through the network and makes decisions based upon packet metadata and contents. This wider viewpoint provides more context and the ability to detect a wide variety of threats, but these systems lack visibility into the internals of the endpoints that they protect. Network-based IDSs are limited to wired monitoring and cannot detect wireless protocols. Also, they have high false positive (non-intrusive activities classified as intrusive) and false negative (intrusive activities classified as non-intrusive) rates, which many times makes them untrustworthy.

A *Wireless-based IDS* is similar to an NIDS, but it captures wireless network traffic, such as ad hoc networks, wireless sensor networks and wireless mesh networks. A WIDS is susceptible to physical jamming attacks, in order to obscure any intrusive activity and cannot avoid evasion techniques, which are

ways that an attacker can evade the detection of a malicious activity by an IDS. It is deployed within range of an organization's wireless range but can also be deployed where unauthorized wireless networking could be occurring.

A Network Behavior Analysis system inspects network traffic to recognize attacks with unexpected traffic flows and are superior at detecting reconnaissance attacks, Denial of Service attacks and more. The major limitation of NBAs is that there is a significant delay in the detection of attacks, as flow data are transferred to NBA in batches and not in real time. It is most commonly deployed in a place where they can monitor flows between an organization's networks and external networks, such as the Internet and a business partner's networks.

Last but not least, a *Mixed IDS* is made by combining two or more approaches of the Intrusion Detection System. The four primary types of IDS technologies – host-based, network-based, wireless-based and network behavior analysis systems – each offer fundamentally different capabilities and each technology offers benefits over the others, such as detecting some attacks that others cannot, or some attacks with greater accuracy, or functioning without significantly impacting the hosts' performance. Therefore, a combination of some of these technologies is can be used in order to widen the spectrum of attacks that can be detected. For example, NBA products can be deployed along with a Wireless-based product, in order to achieve strong detection capabilities for DoS attacks, worms and other threats that cause anomalous network flows (performed normally by the NBA system) but also be to be able to detect rogue WLANs simultaneously (performed by the WIDS). Nevertheless, in most environments a combination of Host-based IDSs and Network-based IDSs is needed at the minimum, where HIDSs are deployed on critical hosts and NIDs are deployed at the boundaries between the organization and the outer networks.

1.1.4 Detection Methodologies

Now the logical question arises, how do Intrusion Detection Systems recognize an intrusion. Intrusion detection methodologies generally are divided into three major categories (Hung-Jen Liao et al., 2013) [7]: *Signature-based Detection* (SD), *Anomaly-based Detection* (AD) and *Stateful Protocol Analysis* (SPA).

Signature-based Detection, or as it is often called *Knowledge-based Detection* or *Misuse Detection*, compares captured packets with known patterns or strings that correspond to a *known* attack or threat. These patterns that are linked to *known* attacks are called *signatures*, and they may be byte sequences in network traffic or known malicious instruction sequences used by malware. This terminology originates from antivirus software, which refers to these detected patterns as signatures. Although this type of detection methodology is simple and effective against known attacks based on detailed contextual analysis, it is ineffective against unknown attacks, evasion attacks and even variants of known attacks, as it does now have knowledge on these types of threats. Also, it has little understanding of states and protocols and it is hard to keep signatures updated.

Anomaly-based, or *Behavior-based Detection* is a newer technology that is designed to detect and adapt to unknown attacks due to the explosive appearance of malwares. This detection method detects deviations from a normal and known behavior and recognizes these deviations as intrusions. Profiles that represent normal or expected activities are derived from monitoring regular activities, network

connections, hosts and users over a period of time and frequently machine learning is used, in order to create a model that creates and generalizes a trustful activity model. Then, all future behavior is compared to this model and any anomalies are labeled as potential threats and generate alerts. While systems based on Anomaly detection are effective against novel and zero-day vulnerabilities, are less dependent on Operating Systems and facilitate detections of privilege abuse, it has been proven extremely difficult to create a model of "normal" behavior as these systems must balance false positives (non-intrusive activities labeled as intrusive) with false negatives (intrusive activities labeled as non-intrusive) and it is also difficult to trigger alerts on time, so delay between the intrusion and the notification of the staff can facilitate the attacker.

Stateful Protocol Analysis or *Specification-based Detection* indicates that an IDS could know and trace the protocol states (for example pairing requests with replies), by depending on knowledge of specific protocol states and state transitions, which knowledge is based on protocol standards from international standard organizations, such as IETF. One major advantage of this method is that It can distinguish unexpected sequences of commands, but on the opposite side it consumes a big amount of resources in order to trace and examine the protocol states and is unable to detect attacks that look like benign protocol behaviors.

One thing to note, is that these methodologies of detection are not mutually exclusive and can even be complementary. In particular, Signature-based detection and Anomaly-based detection can be used together, because SD focuses on known and certain attacks or threats and AD centers its attention around unknown attacks.

1.1.5 Implementation Approaches

While these methodologies outline the most important ways that Intrusion Detection Systems classify an intrusion, a further classification to subdivide these approaches has been proposed by Stavroulakis and Stamp 2010 [6], in order to highlight their practical use. Some approaches to implement IDSs are based on artificial intelligence technologies such as neural networks and expert systems, others are computationally-based such as special purpose languages and Bayesian and others are based on biological concepts as immune systems and genetics.

In *Neural-based* Intrusion Detection Systems an Artificial Neural Network is trained and refined using a predefined set of examples of benign behaviors and attacks, in order to get the experience necessary to recognize an intrusion. These examples (or samples) are composed by a number parameters that are connected to and are able to describe the desired output (label, in the case of supervised learning). Neural networks in general are able to process data from a plethora of sources, predict events and accept nonlinear signals as input. They can also make fast predictions and can perform supervised learning by mapping input signals to desired responses, such as intrusions and normal behavior. However, the mandatory training of the neural network demands often complex hardware and software and if the conditions change from those used when the neural network was trained, data must once again be collected, analyzed and used for retraining the system.

Bayesian-based Intrusion Detection Systems are an example of *statistics-based* IDSs and they use Bayesian Logic, which is a branch of logic that is applied to decision making and inferential statistics that deals with probability inference. These systems use the knowledge of prior events to predict future events, by fitting an optimal statistical model to experimental data. These systems are considered very robust as they can extract complex patterns from sizable amounts of information that contain a significant level of noise, and small alterations in the model do not affect the performance of the system dramatically. Nevertheless, uncertainty may arise especially when the input parameters to an IDS are independent from one another and the number of parameters needed for defining the model is too high.

Rule-based IDSs, including *Expert system-based* IDSs, provide consistent answers for repetitive decisions, processes and tasks. The implementation of these IDSs involves the use of specific rules for identifying known penetrations, penetrations that would exploit known weaknesses and suspicious behavior. These rules are expressed in the forms of "If-Then" and "If-Then-Else" statements, in order to construct the model and the profile of known intrusions. However, similar to the description of systems based on signatures, the rules need frequent updates to remain current and the acquisition of these rules is a tedious and error-prone process.

Immune-based IDSs mimic the ability of the innate immune system to detect intrusions. These systems can imitate the adaptive immune system to detect new types of intrusions that have not been observed before and do not require a human expert to indicate that the intrusion is actually true, by taking into consideration the activation of the T-cells and the B-cells. They have a faster response to previously seen intrusions and are distributed, requiring no local coordination, which means that there is no single point of failure. However these models are extremely simple since the actual human immune system is still under study and it lacks a theoretical foundation.

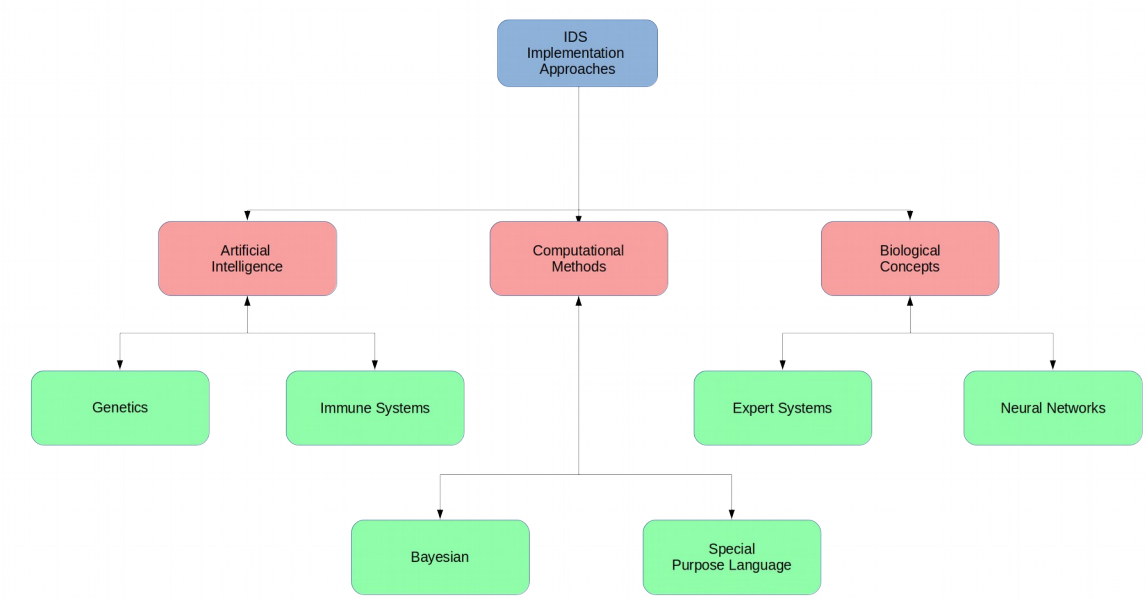

Illustration 1: Different implementation approaches of Intrusion Detection Systems

During this work, as will be presented in later sections, I chose to develop an *Anomaly-based* and more specifically a *Neural-based* Intrusion Detection System following the Network Behavior Analysis system

scheme. This choice was made, as it is one of the most common types of IDSs used today and developing an IDS of this type with the use of Federated Learning would highlight the importance and the applicability of Federated Learning in Intrusion Detection Systems in a real world scenario. In addition, as will be described below, Federated Learning is a machine learning technology that uses neural networks at its core, so a neural-based IDS is necessary when using Federated Learning.

After describing the Intrusion Detection Systems, their objectives, their types and the methods that they use in order to detect intrusions it is time to move on to the next part of this work and present the Federated Learning technology.

1.2 FEDERATED LEARNING

Federated Learning, first proposed by Google at 2016 [8] is a derivation of common machine learning, or better, distributed machine learning, which enables distributed devices to learn a common model collaboratively using their local datasets while simultaneously protecting the privacy of the device or user that provides the training data against leaks or even malicious users. Federated learning does not by itself completely protect the user, as it is susceptible to some sophisticated attacks which will be described below in section 1.2.6

The concept of Federated Learning as proposed by Google and later enhanced by other contributors targets scenarios where a lot of distributed devices collaboratively want to train a universal machine learning model using their local datasets, but focusing on the data privacy, security and other idiosyncrasies of those data. Today, traditional distributed machine learning is based on taking the decentralized data from the edge devices and transferring them to a central location, most commonly in the data center of the enterprise. Those data afterwards are fed in the node or the server that calculates the global model. After the server trains this model, it is transferred back to the devices in order to make the predictions that the application is aiming for.

As one can understand, most notably the first step of this cyclic procedure is vulnerable to data leaks and other kinds of attacks that expose the user's identity. Federated learning tackles this problem by never transferring the raw training data of the devices, but instead providing a "summary" of the data. In Federated Learning, every device trains a model strictly on its own data and sends the trained model to the server. That server aggregates the updates from all the devices in order to represent a joint model and pushes that model back to the devices.

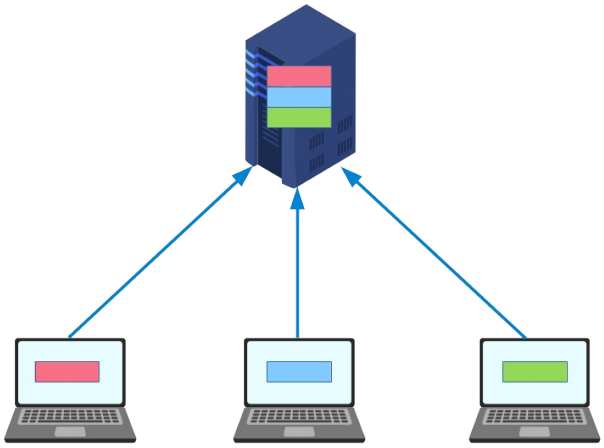

Illustration 2: Each client can recognize an attack using only its information (before training)

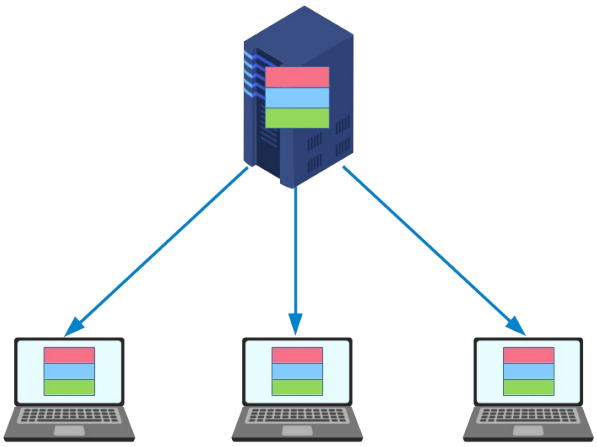

Illustration 3: Each client can recognize an attack using the information on all clients (after training)

1.2.1 Overview of Federated Learning

The main motivation for the development of a Federated Learning application besides privacy and security is also the eradication of issues that traditional machine learning deals with. Primarily, the amount of data that organization produce are growing exponentially. It is only logical to conclude, that the transfer of all the distributed data to a central location can consume vast amounts of precious network resources, as well as computational and storage resources.

Secondarily, one key aspect in the development of Federated learning is the very nature of the training data. In particular, federated learning applications are expected to handle data that have the following characteristics:

- **Non-IID:** No single node's data are representative for the data population. The datasets in each node may be from a different distribution. For example, in the setting of a text message prediction application, the way that a teenager types messages probably differs substantially from the way that her parents type.
- **Massively Distributed:** In a common federated learning application the devices that participate in the training process are of a great number, sometimes tens of millions of devices. Normally, the number of nodes participating can be much bigger than the average number of training examples stores on a given node.
- **Unbalanced:** The devices usually store locally a different amount of training data. Again, in the scenario of the text message application, a teenager sends way more data than their parents.
- **Non-Static:** The data of the users are not stored forever on the device. Data can be added, edited or even deleted.

Additionally, federated learning must take into consideration the state of the devices. Because the devices that participate in the training process are not controlled by a single entity, as is in the case of normal machine learning, where the data is stored on the server that is controlled probably by one

organization, federated learning devices can experience slow internet connections, limited bandwidth, different computational capacities and even violent interruptions of their availability. Such is the case, when a device is connected to a metered network (cellular) or is shutdown in the middle of the process because of low battery or even for network reasons.

1.2.2 Problem Formulation and Participating Entities

As every machine learning problem, federated learning also aims to optimize a cost function based on given datasets.

$$\min_{w \in R^d} f(w) \quad \text{where } f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

Here, w is the vector of the parameters of the model, d is the dimension of the feature vector, that is, how many features our model has and n is the number of training samples that are used for this round of training. Using that definition, the federated learning algorithm can cover linear regression, logistic regression models, support vector machines and more complicated models such as neural networks. For completion the $f_i(w)$ could take any of the following forms in order to compute the cost of the model at each sample:

- Linear Regression: $f_i(w) = \frac{1}{2}(x_i^T w - y_i^2), y_i \in \mathbb{R}$
- Logistic Regression: $f_i(w) = -\log(1 + e^{-y_i * x_i^T w}), y_i \in \{-1, 1\}$
- Support Vector Machines: $f_i(w) = max\{0, 1 - y_i x_i^T w\}$

Suppose that N devices $\{F_1, F_2, ..., F_N\}$ participate in the training of the global model by consolidating their data $\{D_i, D_2, ..., D_N\}$, where D_i is a matrix containing the training data available on the i^{th} device. Each row represents a sample and each column represents the features available for the respective samples. Also, some samples may also contain labels. The label space is denoted by Y, containing all the labels.

The traditional method would be to transfer the distributed data in D, where D contains all the data from all the devices, and use that dataset to train one model. The Federated Learning method instead, is learned by collaboratively training a model, in which each data owner or device never exposes his data to others. So each communication between the devices and the server contains generally less information than the original raw data that are stored in the devices. The accuracy of the model that is finally learned is not substantially lower that if a traditional approach was used. Please note, that in some cases the vector of the model parameters w which is eventually learned and sent to the devices by the server, can contain some sensitive data so it can be susceptible to some kinds of attacks, such as inference attacks.

1.2.3 Participating in the Training Process

As noted before, Federated Learning is based on the iterative process of some devices training a model locally, sending an update to the server and receiving an answer from the server containing a new jointly learned model. The first step of the devices in a federated learning implementation as is outlined by

Bonawitz et al. (2019) [9] is the announcement of their willingness to participate in the following training round. Each device that wants to participate in the training process has to announce first its availability to the aggregating server. This process, which is called the *Selection Phase,* chooses which devices will participate in the next training round. Normally out of probably the tens of thousands of devices that announce that want to participate in the round, only a small subset of them is chosen to participate, close to some hundreds of devices.

The devices that want to participate in the training process announce to the server their willingness to participate by opening a bidirectional stream, which is used to track aliveness and orchestrate multi-step communication. It's important to say, that the devices make such an announcement to the server only if they meet some eligibility criteria, such as connection to an unmetered network (such as WiFi) to not spend a significant amount of the data that the user pays for, as well as a charging state to not affect the user experience regarding normal usage of the device etc. Those criteria are determined based on nature and the requirements of the applications. After a specific amount of time, the server finally selects the subset of the devices that will eventually participate in the following training round. If a device is not selected to participate in the round, it is given by the server a time window that instructs the device when to check in again in order to participate in the next round.

Server side, during the selection phase, the server listens for incoming announcements from devices. The server collects all these announcements and decides whether the round can start or it should be abandoned. During this phase the server considers a timeout, a device participation goal count and a minimum percentage of the goal count that Is required for the round to operate normally. The server listens for announcements as long as the goal count has not been met and the timeout has not been reached. If the goal has been reached before the timeout has passed, then the round starts. In the other case, where the timeout has been reached, the server checks whether the minimum percentage of the goal count has been reached and starts the round. On the other hand, if the percentage has not been reached then the round is abandoned.

1.2.4 Training the Model

1.2.4.1 Training on the Client

If the round can start, before the training can initiate, the server must contact the selected devices from the Selection Phase, in order to give them instructions on which task they should perform (the devices can perform a variety of tasks, such as training and evaluation). Specifically, according to [9], the server sends to the selected devices an *FL Plan* that contains a description of the computation that they must perform and instructions on how to do it. Also, the server sends an *FL Checkpoint* to each participant which contains the current global parameters and any other necessary information. The initial global parameters can be either randomly assigned, or they can be previously trained on some proxy data. That is the case in the first round of the application, since the next rounds as it will be seen later, will send the global parameters from the previous round. At this point, the server awaits for the updated model parameters as calculated by each device.

When a device receives an *FL Plan* and an *FL Checkpoint*, the training (or evaluation) can start. The device now starts training the new model parameters, using advanced optimization algorithms such as normal Gradient Descent (GD), Stochastic Gradient Descent (SGD) and many more, but SGD is used more frequently. After the loss function has converged or if a predetermined amount of iterations has been completed, the device has in its possession the updated model parameters. Of course, as one can suspect, because each device's dataset is different from the datasets on the other devices, those new model parameters are fitted only to that device's data. Thus, this task is performed on every device that was selected for the training process.

After the devices have finished training the model, they send to the server the updated model parameters along with a weight of their total update, so the server can perform the averaging of all the updates. In order to add a simple initial defense mechanism to the system, the devices do not send the raw updated parameters, but instead the change of each respective parameter in comparison to the global parameters sent with the FL Plan.

$$Update: w, \theta_{train} - \theta_{init}$$

The server starts collecting the updates from the devices that participate in the training process. Because the devices that implement the federated learning application can not guarantee their constant and uninterrupted availability, some devices may fail to respond with an update to the server, either by failing altogether or by taking too much time to respond. The server in order to operate normally, has to take these limitations into consideration. If enough devices respond in time, the averaging can proceed normally. But if there are many straggling devices those devices are simply ignored, if a certain time has elapsed.

1.2.4.2 Averaging the Updates

When the server finishes collecting the updates from all the devices that responded with their updated model parameters, it realizes that it is time to aggregate all the updates in order to create the global model. The averaging in the server is performed by the *Federated Averaging* algorithm first introduced by McMahan et al. (2019) [10].

The Federated Averaging algorithm is divided into two parts, the client side and the server side. I will begin by explaining the algorithm from the perspective of the client. Primarily, the client receives from the server the initialized vector of the weights of the model. Generally, as it was shown in [10], it is most efficient for all the clients to initialize their weight vector using the same initialization vector. So the same gradient vector is sent to all the selected clients.

CLIENT:

1. The client divides its local data into mini batches where the mini-batch used for this round is denoted by the *B* parameter, because the Federated Averaging was built with using the Stochastic Gradient Descent as an optimization algorithm for the training process on the clients. The size of the local mini-batch is determined by the *B* parameter.

2. The client calculates the weight n of its update as a whole, which is proportionate to the size of the local mini-batch.
3. The client computes the updated gradients using a learning rate η, for all the batch of the training samples.
4. The client calculates the weighted update. As one can see, the update doesn't contain the updated gradients, but instead it includes the difference of the new gradients from the ones supplied by the server.
5. The client returns the updated gradient vector and the weight of its updates to the server.

From the perspective of the server, the server is in charge of selecting the clients what will participate in the training, of sending the initialized gradient vector to the clients and averaging their updates as soon as they arrive. At this point, the developer of the federated learning application must decide how many rounds will take place, as well as how many clients can be selected for any given round (denoted by K).

SERVER:

1. The server initializes the gradient vector that will be sent to the clients. This vector, as mentioned before can be randomly initialized, or previously trained on proxy data.
2. Until the loss function converges or until a certain threshold of rounds has been reached the server executes the following. It selects the number of clients and those clients that it will expect to receive an update from. The number of the clients is denoted by K. As one can observe, the server selects more clients than it necessarily needs in order to have some resilience to clients that fail to respond to the server with their updates.
3. In parallel, the server waits for updates from the clients that were selected. It doesn't have to wait for all clients that were selected in the previous step. If only K clients respond, instead of the 1.3K clients that were selected, the round proceeds.
4. The server adds all the updates that were received from the clients that responded in time.
5. The server adds all the weights of the updates that were sent by the clients that responded along with the updated gradient vectors.
6. The server averages the updates by dividing the updates with the weights of the updates calculated in step 4 and 5 respectively.
7. The server calculates the new model parameters based on the average that was calculated in step 6. This is now the current global model parameters.

The process repeats until the loss function converges or a specified amount of iterations is reached.

Algorithm 1 `FederatedAveraging` targeting updates from K clients per round.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
Select $1.3K$ eligible clients to compute updates
Wait for updates from K clients (indexed $1, \dots, K$)
$(\Delta^k, n^k) = \text{ClientUpdate}(w)$ from client $k \in [K]$.
$\bar{w}_t = \sum_k \Delta^k$ *// Sum of weighted updates*
$\bar{n}_t = \sum_k n^k$ *// Sum of weights*
$\Delta_t = \Delta_t^k / \bar{n}_t$ *// Average update*
$w_{t+1} \leftarrow w_t + \Delta_t$

ClientUpdate(w)**:**
$\mathcal{B} \leftarrow$ (local data divided into minibatches)
$n \leftarrow |\mathcal{B}|$ *// Update weight*
$w_{\text{init}} \leftarrow w$
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
$\Delta \leftarrow n \cdot (w - w_{\text{init}})$ *// Weighted update*
// Note Δ is more amenable to compression than w
return (Δ, n) to server

Illustration 4: The Federated Averaging Pseudo-Algorithm

After all the rounds have been completed or convergence has been reached, whichever happens first, the server pushes the new model to the devices, so every client has in its possession the updated global model in order to make predictions.

1.2.5 Privacy

As one might have observed from the previous description, Federated Learning so far hasn't mentioned any mechanisms that enhance the privacy and the security of the data, besides the fact that the clients do not expose their data and the updates do not contain any raw training data. That is, because Federated Learning, if implemented alone is vulnerable to many kinds of attacks, as will be described in the next section. Federated Learning gains its privacy and security features from additional mechanisms protocols that are developed and implemented along with Federated Learning.

Anonymized Datasets
Primarily, because the source of the updates are not needed in any part of the process, the updates that are sent to the server from the devices could be transmitted over a mix network as Tor or a trusted third party in order to hide the source of the devices. While this measure seems to protect the user, it still can put the user privacy at risk. Specifically when Netflix announced their competition 'The Netflix Prize' where the goal was to predict the ratings of movies based on an anonymized dataset of its current users' reviews, researchers [11] published a paper announcing that they successfully identified people from this dataset by combining with data from IMDB. So while anonymized data may seem harmless, they can be susceptible to so called, linkage attacks.

Model Parameters in the Updates

Additionally, as it has already been mentioned, the raw training data remains strictly stored locally, so no single raw dataset is sent to any place in the network,. This is of course, the main goal of the development of Federated Learning and a direct application of the principle of focused collection and data minimization proposed by the 2012 White House report on privacy of consumer data.

Ephemeral Updates

One important note is that the updates that are aggregated by the aggregating server are discarded as soon as they are used for the calculation of the global model. So, once the server finishes aggregating all the updates from the devices that sent one to it, the updates are not stored in any persistent storage in order to remove the risk if the control of the storage of the server ends up in a malicious party that attempts to read its data.

Homomoprhic Encryption

Homomorphic Encryption [12] generally protects the user data privacy during the parameter exchange when the update aggregation needs to take place. Initially, the computing device encrypts the update using homomorphic encryption and sends it to the server. Then, the interesting part of the homomorphic encryption is that it is not necessary to decrypt the message in order to perform arithmetic calculation. Instead, the receiver can do some arithmetic calculations on the ciphertexts and also between ciphertexts. This property, as will be seen right below is very important for the Secure Aggregation protocol which is discussed right below.

But, one key disadvantage of this type of encryption is that it is usually computationally very expensive and it only integer arithmetic calculations can be performed. Also, other problems arise concerning who has the encryption and decryption keys that are needed in the process. So while it is undoubtedly a hugely helpful protocol, it should be used frugally.

Secure Aggregation/Secure Multi-Party Computation

Secure Aggregation or Secure Multi-Party Computation along with Differential Privacy are probably the two most important protocols in the Federated Learning framework that enhance the user data privacy. Secure Aggregation works as an intermediary between the devices and the server that collects the updates and is implemented in the case where the server is assumed to be honest but curious, by hiding the update content from the server and when it is possible to infer the parts that comprise an update from looking at the parameters. In the most simple case, this is possible when an update has all the model parameters equal to zero except from one. So, in this case we infer that for that sample, that feature that has a non-zero value appears as frequently as it is indicated by the respective parameter value.

Secure Aggregation takes advantage of a trusted third party which aggregates all the updates from the devices. So the devices encrypt their updates before they send them with Homomorphic Encryption and they send it to the third party. That third party, which is in charge of the aggregation of the updates cannot see the content of the updates but instead sees a random ciphertext. As mentioned before, when a message is encrypted with Homomorphic Encryption, the aggregating party can aggregate the updates without deciphering them. After the aggregate has been calculated, the aggregating party sends the output to the server in order to calculate the updated global model.

Another key aspect of the Secure Aggregation algorithm is that is tolerant to drop-outs of devices. Devices can unexpectedly stop sending updates to the server for various reasons, but Secure Aggregation can produce a correct output based on the updates that the devices that remain in the protocol send. Of course, those devices cannot pass a certain threshold of minimum participants. In addition, during the aggregation phase, the server waits for a specific period of time to receive updates from the devices. If enough devices (above the minimum threshold) send their updates to the server before the time has finished, the process continues normally. Otherwise, if the timeout passes and the minimum threshold has not been reached yet, the round is aborted.

While Secure Aggregation enhances the privacy of the data, one shortcoming is that the computation overhead of the encryption and the communication overhead for sending the keys are significant and can be the bottleneck of the whole Federated Learning process. So, the designer of such an application must realize that it's a trade-off between security and time efficiency. Also, while the server cannot see the individual updates it can infer some individual updates but cannot know to which users these updates refer.

Differential Privacy

Differential Privacy [13] or k-anonymity is another key protocol that is complementary to Federated Learning. It has two main goals: to guarantee that no individual update can influence the joint model above a specific bound and to obscure or to "disfigure" as much as possible the updates sent to the aggregating server by adding noise, in order to not be able to violate the privacy of the data and simultaneously without reducing too much the overall accuracy of the trained model. More simply, Differential Privacy describes a promise make by a data holder to a data subject: "You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, datasets, or information sources are available". As this sentence suggests, linkage attacks such as the one of the Netflix Prize could be neutralized if Differential Privacy was implemented.

Differential privacy makes some alterations to the Federated Averaging algorithm presented earlier. Primarily, the server no longer selects the users participating in the round randomly. Instead, it selects every user independently based on an individual probability. Also, the size of the selected users is not fixed, but varies and is generally larger than the Federated Averaging size. Secondarily, a bound is set on the influence of an individual update to the joint model by using the technique of clipping, in order to mitigate the possibility of one specific update being in control of the whole joint model.

According to this protocol, in order to tighten the privacy of the updates there are two courses that can be followed, but both of them obey to the same principle, that Gaussian noise will be added to some data in order to obscure the reality of the updates slightly but slightly enough to ensure their privacy. The first way, which tends to be deprecated is to perform the addition of the noise on the trusted aggregator (the server when Secure Aggregation is not used or the trusted third party when Secure Aggregation is used), that is on the joint update. The second way, which is where applications tend to move towards, is to perform the noise addition on the individual updates before they are sent to the server.

One minor disadvantage that must be taken into consideration when designing Federated Learning systems is that the model which is trained with Differential Privacy has slightly worse accuracy than if

Differential Privacy was not used. Also, that accuracy is decreased as the amount of noise added is increased. So, here too, there is a trade-off, but this time between security and accuracy.

BlockFL

A really robust implementation of Federated Learning made its appearance in 2019, where a group of researchers (Kim et al. 2020) [14] started leveraging the blockchain technology, in order to implement a safer Federated Learning platform. They proposed a Blockchained Federated Learning, referred to as BlockFL, where local learning model updates are exchanged and verified, enabling on-device machine learning in a decentralized manner by utilizing a consensus mechanism in blockchain. They have also considered an optimal block generation rate, network scalability and robustness issues.

The operation of BlockFL is summarized as follows: Each device computes and uploads its local model update to its associated miner in the blockchain network which is selected uniformly and randomly from the complete set of miners. The miners, exchange and verify all the local model updates and then run Proof-of-Work (PoW). Once a miner completes the PoW, it generates a block where the verified local model updates are recorded, and finally the generated block storing the local model updates is added to the blockchain, also known as the distributed ledger, which is downloaded by devices. At this point, each device computes locally the global model update from the new block.

1.2.6 Vulnerabilities

While Federated Learning seems to be secure and to tighten the privacy of the user's data, like all the systems it has its own set of vulnerabilities that can compromise the whole application. Attacks can happen on the participating devices and the aggregating server, but also on the connection between them. Depending on whether the attack happened on the communication between the server and the participants, the attack is classified as outsider attack or an insider attack if it was the server or the devices that attacked [15].

The most common form of attack that can be launched against a Federated Learning system is an *inference attack*. When updates flow from the devices to the aggregators, a malicious user who intercepts the data can infer the content of the training data by examining the values of the locally trained model parameters that are the content of the updates. As was mentioned in the previous section that is possible in the most simple form when all the parameters are zero except one. Of course, an inference attack can occur if the transferred data are not sent through an secure and private channel. Also, if the channel is secure but the server is not completely trustworthy, that is, it is honest but curious then a secure channel is not enough to prevent such attacks. One possible mitigation technique would be to use differential privacy to inject noise into the parameters or use Secure Multi-Party Computation.

Furthermore, another recent type of attack can take advantage of the nature of the training of the global joint model. Researchers (Bagdasaryan et al. 2020) [16] showed that Federated Learning is vulnerable to model-poisoning attacks which are significantly more powerful than data-poisoning attacks. Generally, malicious users can take advantage of the fact that users, malicious or not, influence the joint model via their updates. An attacker can compromise a user's device and initiate a model replacement attack, in

order to replace the joint model with another that suits the attacker's purposes. Also, that new model is equally accurate on the main task (for example classification) and the attacker can control how the model performs on a chosen backdoor subtask that he chooses even after many rounds after the attack. So, in the setting of an animal image classification, the compromised model can intentionally misclassify dog pictures as cats (backdoor subtask), while correctly classifying other images (main task).

One might think, that in order to mitigate that attack, anomaly detection techniques can be deployed on the aggregating server. Unfortunately, one key characteristic of that kind of attack, is that common defense mechanisms like anomaly detection can not be deployed against it, because it requires access to the participant's local training data or their model updates, both of which are either impossible or discouraged in the setting of a privacy concerning applications, especially when Secure Aggregation is used (I remind that when Secure Aggregation is used the individual updates cannot be inspected when they are being aggregating). Even if anomaly detection is somehow incorporated into the aggregating process, the model replacement will remain effective by using constrain-and-scale or train-and-scale techniques, if the anomaly detector examines the model's weights or its accuracy on the main task. Thankfully, Ozdayi et al. 2020 [17] have proposed a defense system that mitigates backdoor attacks by adjusting carefully the server's learning rate, per dimension, at each round based on the sign information of agent's updates.

1.2.7 Categories of Federated Learning

Before a federated learning project is developed, the designer must consider first, the architecture of the federated learning implementation and the privacy possibilities and constraints that it introduces, because the way the data are split matters in terms of how Federated Learning is implemented. As introduced in section 1.2.2 the dataset matrix of every device i is denoted by D_i where each row represents a sample and each column a feature. Some datasets inside D_i can also contain labels. I shall denote X as the feature space, Y the label space and I as the Sample ID space. For instance, in an tumor classification example, Y may be the type of tumor, benign or malignant, and X could be the size of the tumor, shape, chemical composition and many others. Out of these 3 components and specifically out of the way that the data are distributed among the devices in the feature and sample ID space, emerges a categorization of 3 federated learning architectures, namely Horizontally Federated Learning, Vertically Federated Learning and Transfer Federated Learning.

Horizontal Federated Learning

Horizontal Federated Learning or Homogenous Federated Learning is the case in which the datasets of the participants share the same feature spaces but differ in the sample ID space. For example, two adversarial (that is, which offer the same products) local supermarkets, which want to predict the consumer habits of their customers will probably have the same feature space because their businesses are very similar, so the keep the same characteristics for them. But because of the supermarkets' geographic location, the sample IDs will be different in its majority, because most of the customers go to the supermarket in their neighborhood. Horizontal Federated Learning is the most common form of Federated Learning implementation inside one organization, as the interesting data inside inside the same organization have more or less the same form.

$$X_i = X_j, Y_i = Y_j, I_i \neq I_j, \forall D_i, D_j, i \neq j$$

In this type of Federated Learning usually two security assumptions are made. First, that the participants are honest and second, that the server is honest but curious. This forces the designer to implement Secure Aggregation and Homomoprhic encryption in order to provide security against the server. Also, as was clearly stated before, it's possible to attack a Federated Learning application by taking control of a user's device, so the first assumption may be inaccurate in the Horizontal Federated Learning setting.

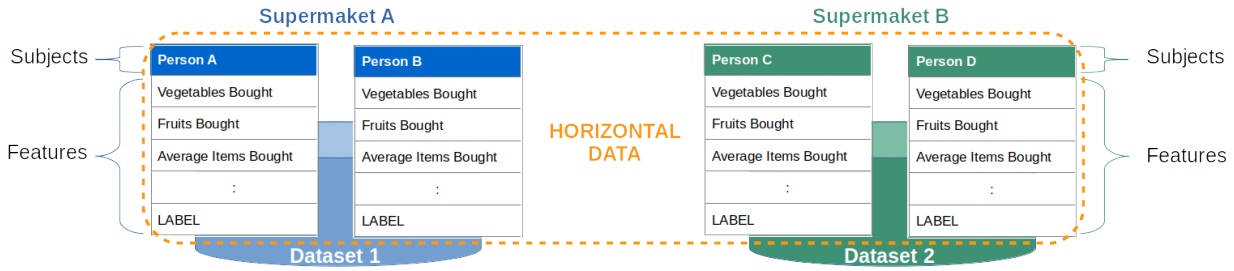

Illustration 5: Horizontal division of the data

Vertical Federated Learning

Vertical Federated Learning or Heterogenous Federated Learning concerns itself with the cases in which two datasets have the sample ID space but differ in the feature space. For example imagine the case of two companies, an online blog publisher and a book store. Those two businesses may have the same customers, because many customers can read blog posts from the online blog and also buy books from the book store. In contrast, the feature space will be different because these two businesses keep track of different habits.

$$X_i \neq X_j, Y_i \neq Y_j, I_i = I_j, \forall D_i, D_j, i \neq j$$

One key step when training a joint model under a vertical federated setting is to perform encrypted entity alignment, during which the two companies confirm the common users of both parties without exposing their respective data and the users that do not overlap with each other. In order to ensure the confidentiality of the data during the entity alignment a trusted third party collaborator is involved. By the end of learning, each party holds only those model parameters that are associated with the local set of features. Therefore, at inference time the two parties must cooperate in order to generate the output.

In this category of Federated Learning, the participants are typically assumed to be honest but curious, therefore, no information leakage to the other participants is allowed. This problem can be solved by using a trusted third party during the entity alignment and implementing Secure Multi-Party Computation which provides privacy under such circumstances.

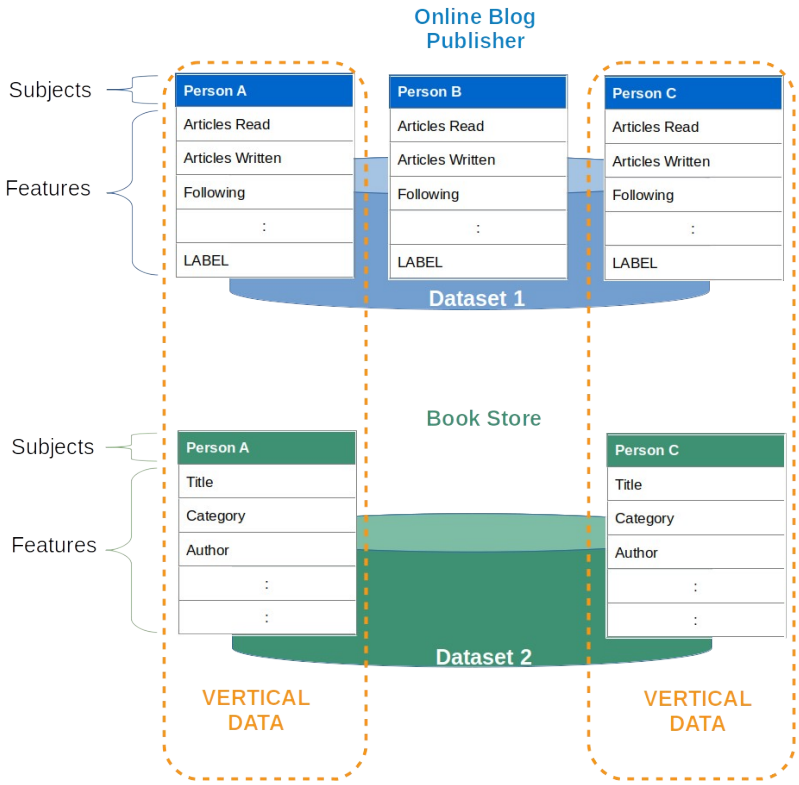

Illustration 6: Vertical division of the data

Federated Transfer Learning

Although Federated Transfer Learning is used very sparingly, it is worth a mention. Federated Transfer Learning is used when the collaborating parties differ in feature space and sample space. Consider the case of a hospital located in Japan and a bank located in Germany. Because they are different businesses they will have different features. Also, because they are located in different geographical regions and because their service is mostly not digital, their customers come from the local regions, so they have mostly different user groups also.

$$X_i \neq X_j, Y_i \neq Y_j, I_i \neq I_j, \forall D_i, D_j, i \neq j$$

In order to collaboratively train a joint model, the two distinct parties follow a similar process to the vertical federated learning, but with a few changes. Specifically, a common representation must be learned using limited common sample sets and later applied to obtain predictions for samples with only one-side features. Suppose that the parties A and B have a very small set of overlapping samples and we are want to learn the labels for the entire dataset in party A. The vertical architecture focuses on learning the labels only on the overlapping samples. Therefore, in order for Transfer Learning to work, the intermediate results that are exchanged between parties are changed. Specifically, the two parties learn a common representation of both feature spaces in A and B and minimize the errors in predicting the labels for party A by leveraging the labels that party B possesses. Again, at inference time, both parties need to collaborate in order to produce the output.

As in the vertical federated learning setting the participants are assumed to be honest but curious so no sensitive information is allowed to be leaked between them. Once more, a trusted third party collaborator must be used.

1.2.8 Client Architecture

After the designer has realized which category of Federated Learning applies to his situation, it's time to start designing the client and server devices. Below, I start by explaining the architecture of the client and right afterwards the architecture of the server, while simultaneously referring to the jobs that each component is in charge of.

1.2.8.1 Client

In this section I describe the software architecture running on a device that is participating the Federated Learning, that is the party that holds data.

Example Store

Initially, in its core, every device must keep a local repository of locally collected data (example store) that are used for model training and evaluation, which must be available to the FL Runtime. The FL Runtime which is the interface between the server and the example store, when provided a task by the server such as training, accesses the example store to compute the updated model parameters and form the updates that will be sent to the server. Generally, it is recommended that the applications limit the total storage footprint of their example stores and automatically remove any old data. Also, it is crucial that in order to reduce the attack surface to only the server and the communication between them, it is also recommended that the device owners follow best practices for on-device security such as encrypting any permanently stored data and physically disassembling the device as soon as it is ended its life cycle.

Programmatic Configuration

Before anything else happens, the configuration of the device must take place. At first, an application configures the FL Runtime by providing an FL population name and instructing it to register its example stores. This configuration also instructs the device to schedule a periodic FL Runtime job that is invoked only when the device meets the eligibility criteria to participate in the selection phase that was described in section 1.2.3 (the device is idle, connected to an unmetered network, charging etc). If the job is started but at some point some of these criteria is not longer met, then the FL Runtime will abort freeing the allocated resources. In addition, the FL Runtime can run either on the same application with the configuration application, or on another separate centralized application. One must consider first the capabilities of the platform on which the application resides that facilitate the inter-app communication before it decides which type of scheme he will follow.

Job Invocation

If the eligibility criteria are met, the FL Runtime contacts the FL server to request to participate in the next task for the FL population that it belongs to as was configured earlier. The device now awaits a response from the server, which either supplies the device with an FL Plan stating what task it should be perform and the way to do it, or a suggested time to check in later.

Task Execution

When the device is selected, the device receives the FL Plan from the server, retrieves all necessary data from its example stores and starts executing the task, either trains a model and computes the updated model parameters, or extracts metric for the evaluation of the model, which is the equivalent to the validation step on held-out data in data center training.

Reporting
When the computation of the model parameters or the extraction of the metrics have been completed, the device sends to the server the results of its computations and cleans and frees any allocated resources.

Multi-Tenancy
While one might think that one device can belong to at most one FL Population, this is not true. The implementation that McMahan et al. 2019 [9] suggest allows for multiple instances of FL Populations to exist on one device, performing independent calculations for their respective FL population. In order to to avoid the overloading of the device with many simultaneous training sessions at once, some coordination between the activities must take place.

Attestation
One key privacy feature of the whole Federated Learning framework is that the devices can participate in the Federated Learning anonymously, in order to overcome certain types of attacks that exploit the source of the updates and conform to international regulations. In contrast, we want to verify the devices that participate in the tasks, to ensure that only genuine devices influence the results. Thankfully, there is no need for user identity verification to participate in the tasks. In the setting of an Android application this is possible with the Android's remote attestation mechanism, which helps to ensure that only genuine devices and applications participate in Federated Learning and gives some protection against data poisoning.

1.2.8.2 Server

While the design of the client is straight forward and easy to implement, the server's design is much more complex, because of the scalability that is necessary for a proper Federated Learning application. The server must be able to operate with hundreds of millions of devices in each FL Population with thousands of participants in each round. Also, the size of each message sent to and from the the devices range from some kilobytes to tens of megabytes. In order to enable and facilitate the implementation of a scalable server, an Actor Programming Model is considered (Hewitt 1973) [18]. Actors are universal primitives of concurrent computation which use message passing as the sole communication mechanism, which are treated strictly sequentially. In order to enable dynamic scalability, multiple ephemeral instances of some actors can be run just for the duration of a given FL task, each of whom can make logical decisions, pass messages to other actors or create more actors dynamically. Also, actor instances can be located on the same process/machine or distributed across data centers in multiple geographic regions.

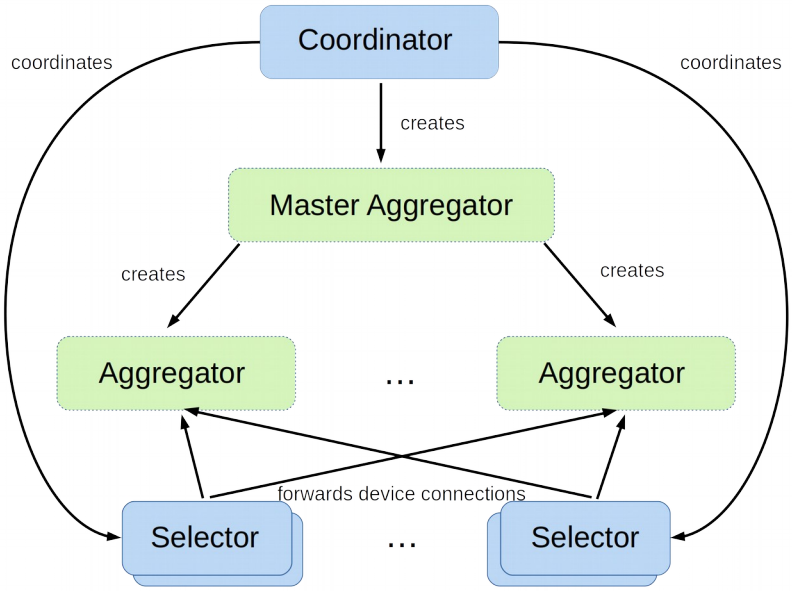

Illustration 7: The server's internal architecture

Coordinators

As the name reveals, coordinators are the top-level actors which enable the synchronization of the whole process and there are that many coordinators as there are FL Populations. Each coordinator registers its address and the FL Population that is manages in a shared locking service, so that other actors, the Selectors, can always reach a single owner of every FL Population. The Coordinator based on the FL Task scheduled knows how many devices are connected to each Selector and instructs them how many device connections to accept for the next round. The Coordinator spawns Master Aggregators to manage the rounds of each FL Task.

Master Aggregators

Master Aggregators manage the rounds of each FL Task and for scalability reasons many Aggregators may be spawned dynamically by those Master Aggregators. It is important to note, that all actors keep their state in memory and are ephemeral, so no information is written on persistent storage until it is fully aggregated by the Master Aggregators. That property removes the possibility of many kinds of attacks that target the persistent logs of per-device updates, because no such logs are kept.

Selectors

Selectors are the ones that accept and forward connections with devices. As noted earlier, the Coordinator instructs the Selectors how many devices are needed for a FL Population, based on which the Selectors accept or reject connections from devices. When the Master Aggregator and their Aggregators are spawned, the Coordinator also instructs the Selectors to forward a subset of its connected devices to the Aggregators, in order to facilitate the efficient allocation of devices to FL Tasks from the Coordinator. Also, the Selectors can be globally distributed, near the devices from which they accept connections and limit communication with the remote Coordinator.

Pipelining

In order to achieve latency optimization, some phases can be pipelined. Specifically, because the Selection phase of the next round, in contrast to the Configuration and Reporting phases, does not depend on any input from a previous round, it can be run in parallel with the Configuration and Reporting phases of the previous round. Simply, the Selector actors run the Selection process continuously.

Failure Modes

It is worthy to note that Federated Learning is not tolerant only to the failures of the devices but also to the components that comprise the server. For instance, if an Aggregator or Selector crashes, only the devices that were connected to those actors will be lost. If the Master Aggregator fails, the current round of the FL task it manages will fail, but will then be restarted by the Coordinator. If the Coordinator itself crashes, a Selector will detect its failure and will respawn it. Because the Coordinators are registered in a shared locking service, this will happen exactly once, so at any time only on instance of each failed Coordinators will respawn. So in every case of failure, the system will continue to make progress, either by completing the current round, or restarting from the results of a previously committed round.

2. APPLICATION DEVELOPMENT

In order to prove that Federated Learning is a viable option to choose when developing Intrusion Detection Systems, it is necessary to build a testing application that receives a dataset containing *realistic* network traffic, both benign (that is, normal user behavior) and malignant behaviors(that is, behavior linked to an attack), and trains a model using Federated Learning. The following sections discuss the environment that was chosen o develop the application, the dataset used for the training of the model and its preprocessing and the network's architecture.

2.1 DEVELOPMENT ENVIRONMENT

2.1.1 Programming Language and Frameworks Used

First and foremost, it is essential to elaborate first on the programmatic choices that I have made in order to develop the application[1].The network (which I will explain later) and the training of the model is going to be developed using "Python" and mostly the packages "TensorFlow Federated", "Keras", "scikit-learn" and "Pandas".

"TensorFlow" is a free and open-source software library, which aims to facilitate the development of machine learning applications, both using traditional machine learning techniques and more modern methods such as neural networks. "TensorFlow Federated" is a free and open-source framework as well and was developed to ease open experimentation and research with Federated Learning. It is comprised by two main interfaces, a high level interface called "Federated Learning (FL) API" and a lower-level interface called "Federated Core (FC) API". For the application I used the first interface, the Federated

1 The source code of the application can be found on my personal Github page and the following link: https://github.com/AndronikosGiachanatzis/FLinCTI

Learning (FL) API, because it allows developers to apply the included implementations of federated training and evaluation to existing TensorFlow models.

At this point, it must be given attention to the fact that the FL API supports at the moment only local simulation of the federated computations. This means, that all the necessary computations of Federated Learning and all the participating entities of the network which will be described in later sections, will be inside one device, and specifically inside the same process. This though, will be done taking also into consideration the *virtual* separation of those entities and their respective data.

"Keras" is a deep learning framework, and is widely used along with TensorFlow. In this application, Keras is used to define the model's architecture, such as the number of layers and their dimensions, the number of neurons per layer, their activation functions and the weight initialization techniques of each layer's neurons.
"Scikit-Learn" is another useful and robust machine learning package, that offers an enormous variety of tools for machine learning and statistical modeling. Here, it was used in order to perform some fundamental preprocessing steps to the dataset, such as scaling and splitting the dataset into a train and test set.

Finally, "Pandas" is a fast and powerful open source data analysis and manipulation tool and was used mostly to save the datasets, on which the aforementioned transformations were performed by scikit-learn.

2.1.2 Model

The model which will be trained is going to be a multi-layered Feed-Forward Neural Network. This type of network architecture was chosen as Feed-Forward Neural Networks are used primarily in cases where supervised learning must be applied and the data to be learned is neither sequential not time-dependent. This, as will be described in the next section, is true also for the dataset that I used. In addition, the dataset (after a few modiciations that I applied to it) includes a feature that characterizes each observation as a Benign behavior or an Attack, so the problem is clearly a supervised binary classification problem. Thus, a multi-layered Feed-Forward Neural Network is the most suitable candidate for this problem.

Moreover, in order to enhance the utility of Federated Learning in a Cybersecurity related application, there needs to be trained another model, with which the federated model will be compared. To accomplish that, another *completely local* neural network is going to be trained using the same architecture of the trained federated model. When these two models are trained, their results are going to be compared in order to have a sample of the probable superiority of one model over the other. I have to remind, that the model that is produced using Federated Learning is expected to be less accurate than a traditional local neural network model. So, what I must focus upon, is the magnitude of that difference in accuracy.

Having outlined the development environment it is time to move on to the examination of the dataset used.

2.2 DATASET

The dataset "Intrusion Detection Evaluation Dataset" (CIC-IDS2017) [19] that was used for the training of the current model was developed by the University of New Brunswick and specifically by the Canadian Institute for Cyber Security (Sharafaldin et al.). It is completely labeled with common updated attacks such as DoS, Heartbleed, DDoS, Brute Force, Port Scan, Botnet, XSS and SQL Injection attacks and contains 78 network traffic features extracted and calculated for all benign and malignant flows. The data was captured in a five day period, starting from 9 am, Monday, July 3, 2017 until 5 pm, on Friday, July 7, 2017. The attacks mentioned were captured both morning and afternoon on Tuesday, Wednesday, Thursday and Friday, while Monday was reserved for purely benign traffic.

In this paper I have chosen to take into consideration only Wednesday of all the days, as it contains data on DoS attacks, one of the most common type of attacks used, and Heartbleed, another critical attack. A short description of both attacks is given below.

2.2.1 Dataset Description

The initial dataset that was used contained 78 features on Heartbleed and common DoS attacks. Shortly these attacks are:

- **DoS (or Denial of Service) Attack:** In this scenario the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of Service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. A DoS attack is analogous to a group of people crowding the entry door of a shop making hard for legitimate customers to enter, thus disrupting trade.

  In this current implementation the DoS attacks that I focused on were DoS Slowloris, Hulk, GoldenEye and Slowhttptest. Slowloris and Slowhttptest consume the server's resources very fast by keeping connections open with minimal bandwidth. GoldenEye is similar to an HTTP flood attack and is designed to overwhelm a web server's resources by continuously requesting a single or multiple URLs. Hulk (or HTTP Unbearable Load King) is a DoS tool that was built to stress testing of the web servers and it can generate a big volume of obfuscated and unique traffic that can access directly the server's resource pool as it bypasses caching.

- **Heartbleed:** Heartbleed was a security bug in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. In order to exploit this bug, an attacker typically sends to the vulnerable host (usually a server) a malformed heartbeat request with a small payload and large length field, in order to elicit the victim's response and divulge secret information such as private keys.

Because the main concern of this work, is not to classify the exact type of an attack, but to be able to distinguish an intrusive network flow from a normal one, the dataset's labels were transformed in this fashion, the 'BENIGN' labels were not altered and all the DoS attacks Heartbleed labels were replaced by the label 'ATTACK'. Thus, the problem has been reduced to *binary classification* problem, where the Intrusion Detection System that will be developed, is going to be able to distinguish malignant ('ATTACK') from benign ('BENIGN') network behavior.

2.2.2 Feature Exploration and Data Cleaning

Initially, before the dataset is fed into the model for training it is crucial that some steps of exploration and preprocessing must take place. In this section I begin with the exploration of the dataset's characteristics and in a later section I process the data according to the findings of the data exploration stage.

First and most importantly, I must examine the data types per feature. According to the table below, in which only a subset of the features is presented as the rest of the features are similar to the ones presented, only the label of the dataset is not numerical (object type). That is because the Label has two distinct string: "BENIGN" and "ATTACK". All the other features are already numerical (int64 and float64)and hence, it is not necessary to do any conversions for now.

FEATURE NAME	NON-NULL COUNT	DATA TYPE
Destination Port	692703 non-null	int64
Flow Duration	692703 non-null	int64
...	...	...
Bwd Packet Length Std	692703 non-null	float64
Flow Bytes/s	691695 non-null	float64
Flow Packets/s	692703 non-null	float63
...	...	...
Idle Min	692703 non-null	int64
Label	692703 non-null	object

Table 1: The number of non-null values and the data types of the features of the dataset

Next, I examine whether there are any missing values. It is easy to find that the feature 'Flow Bytes/s' does have some missing values. Specifically while all the other features (and the label included) have 692.703 non-null entries, the 'Flow Bytes/s' feature has only 691.695 non-null entries. Thankfully, the difference is about 1000 entries, which is small, so it is probably safe to remove the entries containing missing values without causing any unwanted side effects. Also on a side note, the dataset includes also some infinite values which will have to be dealt with later.

Also, it is important to check the balance of the classes of the label.

BENIGN	440.031
ATTACK	252.672

Table 2: Number of "BENIGN" and "ATTACK" observations in the original dataset

As we can observe there are 440.031 "BENIGN" samples and 252.672 "ATTACK" labels. While the labels are imbalanced, there is a slight imbalance in favor of the BENIGN observations, as the proportion is 1,7 BENIGN observations for one ATTACK observation.

As mentioned earlier, the dataset has at the moment some missing and infinite values. In order to continue the analysis I have to handle these entries. Because the number of these entries is relatively small (close to 1000 entries out of approximately 690.000 entries), I chose to discard these samples without causing any unwanted side effects. Therefore, the dataset contains now

BENIGN	439.683
ATTACK	251.723

Table 3: Number of "BENIGN" and "ATTACK" observations in the dataset after the removal of missing and infinite values

Now that the dataset does not contain any problematic values, I can continue its analysis, with the discovery of the most important statistical characteristics of the features. The table 11 in the Appendix lists these characteristics, namely the minimum and maximum values, the mean, the standard deviation and the variance of each feature, all of which will prove to be useful in the following sections.

It is crucial to point out at this moment that there are several features that have variance equal to zero. As these features are constant, they do not offer anything of significance to the model and therefore can be discarded from the dataset. This was achieved using the VarianceThreshold() function of the scikit-learn package. Also, none of the features follows the standard normal distribution as none has a mean equal to 0 or a standard deviation equal to 1.

As a last step of the analysis of the features I calculate the correlation between the independent variables, those are the features of the dataset and the dependent variable, that is the label. Because the label is a categorical variable and the features are all numerical I have to calculate the Point-Biserial Correlation Coefficients of each feature and the label. Shortly, the Point-Biserial Correlation Coefficient is a correlation measure of the strength of association between a continuous-level variable and a binary variable, that is a variable of nominal scale with only two values. Like all correlation coefficients (e.g. Pearson's r, Spearman's rho), the Point-Biserial Correlation Coefficient measures the strength of association of two variables in a single measure ranging from -1 to +1, where -1 indicates a perfect *negative* association, +1 indicate a perfect *positive* association and 0 indicates *no association* at all.

From the table 10 at the Appendix A, one can observe that there are plenty of features that have a relatively strong relationship with the label. Those features are potential candidates to be included in the training of the model.

At this moment there is a clear view of the characteristics of the data. In the next section I will perform some necessary changes to the dataset in order to be better prepared to be fed into the training algorithm.

2.2.3 Data Preprocessing

So far, I have explored the most fundamental properties of the dataset and have also applied some data cleaning methods in order to make the dataset more efficient. At this section, I will process the data even more, so that they can be in a format that will optimize the behavior of the training algorithm.

It is widely known, that neural networks and generally most machine learning techniques expect the data to be in numerical format, even though the might represent categorical attributes. Therefore, it is considered best-practice to convert any categorical data into numerical. In the current dataset, as was discovered in the beginning of the previous section, only the label is in categorical (or string) format, the values of which are "BENIGN" and "ATTACK". These two values have to be encoded in numerical values before they can be fed in the training process. In consequence, the labels "BENIGN" and "ATTACK" were converted according to the table below:

BENIGN	1
ATTACK	0

Table 4: Encoded values of the label's classes

In addition, It was discovered that not all features contribute to the model, so I discarded those that do not relate to the objective and kept those that have higher relation with the Label. This distinction, of course, was not made arbitrarily. I first trained an Extra Trees Classifier on the whole dataset, in order to discover the effect of every variable on the Label. The Extra Trees Classifier (or Extremely Randomized Tree Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a "forest" to output its classification result and is used frequently in both the training of models but also during feature selection. It is very similar to a Random Forest Classifier, differs only from it in the manner of construction of the decision trees in the forest and one of its many advantages is that it can recognize non-convex patterns in a dataset.

On a side note, it can be assumed that a previous research has been made independently on a similar dataset to explore the importance of some features, so this does not violate the assumption of Federated Learning for the absence of knowledge of the datasets from other entities participating in the network.

The 25 most important features that were discovered through the training of this classifier are presented in the table below:

Destination Port	ACK Flag Count
Bwd Packet Length Mean	Avg Bwd Segment Size
Idle Min	Flow IAT Max
Fwd IAT Max	Bwd Packet Length Std
Average Packet Size	Idle Mean
min_seg_size_forward	Packet Length Mean
Idle Max	Max Packet Length
Fwd Packets/s	Min Packet Length
Fwd IAT Total	FIN Flag Count
Fwd IAT Std	Bwd Packet Length Max
URG Flag Count	Packet Length Std
Flow IAT Std	Fwd Packet Length Min
Flow Packets/s	

Table 5: The 25 most important features discovered with the Extra Trees Classifier

Also, some additional features must be discarded from the model, as they have either very low variance or have very low correlation coefficients with the label, such as the "URG Flag Count", which has variance equal to 0,06 and does not offer anything of significance to the model, or "Destination Port" which has a coefficient equal to 0,26. Also, from the pairs of features that have high Pearson's Correlation Coefficients, one is discarded, as their values are already represented by their pair.

So, finally, the features that participated in the final model are:

Bwd Packet Length Mean	Avg Bwd Segment Size
Flow IAT Max	Fwd IAT Max
Bwd Packet Length Std	Idle Mean
min_seg_size_forward	Packet Length Mean
Max Packet Length	Fwd Packets/s
Min Packet Length	Fwd IAT Total
Fwd IAT Std	Bwd Packet Length Max
Packet Length Std	Flow IAT Std
Fwd Packet Length Min	Flow Packets/s

Table 6: Variables participating in the model

After selecting the most important features the dataset is divided is distributed to the clients as will be described in a later section. Now, each client's dataset is split into train and test sets keeping also the proportion of the BENIGN and ATTACK labels from the original client's dataset, from which the former is used for the training of the model and the latter is used for the evaluation of the trained model.

Furthermore, the datasets' features are standardized in order to have a mean equal to (or a very small number close to) 0 and standard deviation 1, which has been proven that neural networks perform better when their input variables are standardized. Also, the test set was standardized using the standard deviation and mean of the train set only, as in a real world scenario a test set is not available and the input is fed directly to the model for inference. And finally, each client's dataset is batched into batches of 32 samples each, shuffled and repeated 100 times.

I set aside for now the data themselves and I begin describing the architecture of the network, on which the application is going to be simulated and deployed.

2.3 NETWORK ARCHITECTURE

In this section I will begin to describe the architecture of the network based on which the final model will be trained and evaluated and the Intrusion Detection System will be deployed. Therefore, it is appropriate to begin by enumerating the entities that will be present in the network and how the dataset is going to be separated and assigned to the clients.

2.3.1 Participating Entities

The training and deployment network will be consisted by 6 main entities, 5 clients and 1 server. Notice that in the picture below an attacker is also presented. This attacker is not going to be developed and is present at the illustration, just to give a notion to the reader about the nature of a *deployed* Intrusion Detection System and the capabilities of the system that I developed.

The 5 clients have the following responsibilities. In the beginning, they have permanently and privately stored in their local memory their respective datasets, as it will be explained further later. On these datasets, certain preprocessing and preparation techniques will have been imposed independently from other devices, such as feature standardization. Since this processing has been completed the clients will await for their participation in the Federated Learning process, which can be either training or evaluation of a model. When the order from the server comes to begin a certain process, which will be described in the server's message, the clients compute the requested calculation and send back to the server their updates, as the Federated Learning protocols demand. It is important to mention again that the dataset of each client is *virtually* unknown to any other entity of the network. Finally, when a satisfactory model has been trained (or the evaluation of a model has been completed) the clients will be able to recognize an attack in the network If an intrusive flow comes through them, since they have in their possession the globally trained model. So in a hypothetical scenario that includes an attacker, as can be seen in the photo below, if a malevolent user tries to make a DoS Slowloris or Heartbleed attack on client 4, then this client will be able to recognize this flow as an attack.

At this point, I must highlight that the development of a malicious user that performs an attack is redundant and is presented in the picture and the description only to grasp the objective of a *deployed* Intrusion Detection System. The attacker's role would be to test the trained model if it can correctly classify a flow as intrusive or non-intrusive (after the flow has been processed so that it can have the same format of the features of the dataset). But this function is already performed, and in fact in a much larger scale, by the test set used during the evaluation of the model, as it possesses a plethora of flows that are either Benign or an Attack. So conclusively, because the objective of the test set is to test the trained model how well it generalizes, or how well it can react to newly presented behaviors, the attacker's role is overshadowed by the test set, and therefore there is no need to develop an attacker.

The server is in charge of coordinating the Federated Learning processes. Specifically, the server does not have any knowledge of the datasets of the clients and its sole duties are sending instructions to the clients to initiate a round of a given process (training or evaluation), the aggregation of the updates of the clients, pushing the updated global model back to the clients and presenting the results of the training or evaluation of the model.

Furthermore, owing to the limitations that the chosen library "Tensorflow Federated - FL API" imposes, the network that was described and its subsequent functions will be located inside the same simulating host, and specifically inside the same process. This limitation though, does not change any assumptions that I made so far, as both Tensorflow Federated and the system that I developed take into consideration the necessary *virtual* separation of the participating entities and their respective data.

Also, the analysis of the dataset that I made so far hasn't violated any of the assumption of Federated Learning as it was conducted for the purpose to get an insight of the data and its properties and it can be assumed that a similar research has been conducted independently in the past.

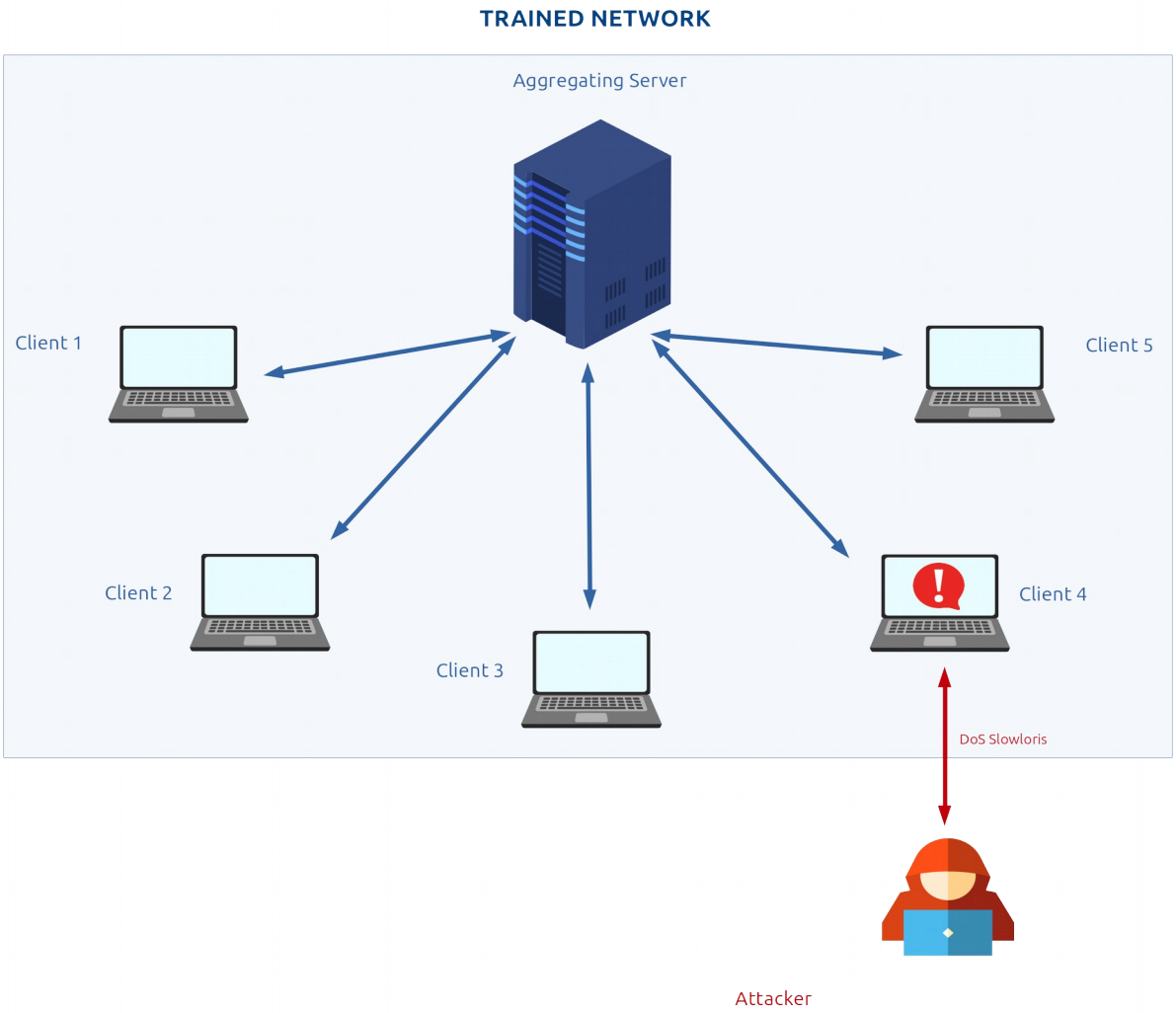

Illustration 8: Network architecture and participating entities

2.3.2 Dataset Division and Distribution

The dataset that I examined in the previous sections remains to be divided and assigned to each of the five clients of the network. This will be done by shuffling first the samples of the dataset and afterwards splitting it into five sub-sets, each containing approximately the one fifth of the original dataset.

In detail:

DATASET NUMBER	OBSERVATIONS	BENIGN OBSERVATIONS	ATTACK OBSERVATIONS
0	138282	88223	50059
1	138282	87712	50570
2	138282	88186	50096
3	138282	87834	50448

4	138278	87728	50550

Table 7: The number of observations per sub-dataset

From the table above one can observe that only the fifth dataset contains slightly fewer observations that the first four and that all the datasets contain different amounts of BENIGN and ATTACK labeled observations, which is in accordance with the assumptions of federated learning, which state that the clients may have different distributions of the data.

In the expectation of simulating the distribution of the datasets that Federated Learning demands, these split datasets are now assigned to each of the five clients in respective order. That is, that the dataset 1 is assigned to the client 1, the dataset 2 is assigned to the client 2 etc. These datasets will be *virtually* separated from the others as Federated Learning demands. Additionally, the division of the data is *horizontal*, as all the clients share the same feature space and differ in the sample space. Also, in this manner, all the participating clients have a common data format, something that is aligned with the scenario of an independent organization, that does not combine data from different organizations, which probably support another knowledge representation format.

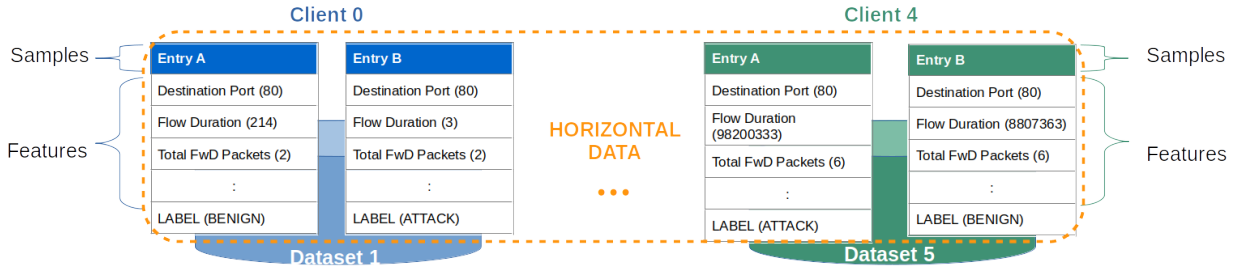

Illustration 9: Horizontal division of the data and assignment of the sub-datasets to the clients

Finally, having described the development environment, the architecture of the model and the dataset's intricacies, it is time to move on to the training of the model and present the experimental results.

3. EXPERIMENTAL RESULTS

The main object of this work is to train and present a model that has a good accuracy ratio, so that if it is fed with new data it can recognize a new intrusive flow from a normal one in the network or device, depending on where the final model is deployed. Therefore, in the following sections I describe the training and testing process of the final model, present the final's model architecture and parameters and make a comparative analysis with a traditionally trained model.

3.1 TRAINING WITH FEDERATED LEARNING

At this section, I elaborate on the training of the neural network with Federated Learning.

3.1.2 Training

At this point, the dataset has been modified to be entered to the model and the model is ready to be trained. After exploring various architectures with different layers, different number of neurons per layer, different activation and initialization techniques and different optimizers, it was discovered that the best performing architecture is the one that is described below.

The final model is a *sequential* contains two *dense* hidden layers and one output layer. The first dense layer of the model contains 300 neurons, uses the "Scaled ELU" (or SELU) activation function and all the neurons' weights are initialized using the "LeCun normal" initialization technique. The second dense layer contains 250 neurons, uses the "selu" activation function and all the neurons' weights are initialized using the "LeCun normal" initialization technique as well. The output layer of the model is also a dense layer and contains only one neuron, as it is a *binary* classification problem, uses the "sigmoid" activation function and its kernel is also initialized using the "he_normal" initialization technique.

Scaled ELU (or SELU), introduced in a 2017 paper by Günter Klambauer et al., was chosen as the designated activation function as it has been observed that it outperforms other widely used activation functions, owing to the fact that if a neural network is built exclusively of a stack of dense layers and if all hidden layers use the SELU activation function, then the network will *self-normalize*, that the output of each layer will preserve a mean of 0 and standard deviation of 1 during training which solves the vanishing/exploding gradients problem. There are, of course, a few conditions for self-normalization to happen. First, the input features must be standardized (mean 0 and standard deviation 1), every hidden layer's weights must be initialized using the LeCun normal initialization, the network's architecture must be sequential and all layers must be dense. As described earlier, all these conditions are met.

The optimizer that was chosen is the *"Adam"* optimizer, configured with a *learning rate* equal to 0,001.

3.1.3 Training Metrics and Evaluation

The training of the model lasted 20 rounds, leading to a model with 78,50% accuracy and 0,054 loss on the training data. This is a very satisfactory result as the model can classify correctly more than the three quarters of the flows observed.

	TRAINING	EVALUATION
ACCURACY	78,50%	78,77%
LOSS	0.054	0,056

Table 8: Metrics of the trained federated model

The model was later evaluated using only the test set, which resulted in an accuracy equal to 78,77% and loss equal to 0,056. Of course the small difference in the two metrics is expected and especially the small difference between the two losses, as the model is fit to the training data only and does not take into consideration the test set. As it is known, the test set represents the measure to which a model generalizes, or the way that it behaves on new data.

In order to get also a visual representation of the training process, the following diagram presents the training curves, specifically the model's accuracy (in blue) and loss (in yellow) with each round. It can be observed that both the accuracy and the loss of the model initially change really fast and after a few rounds their rate declines, so much that in the final rounds the metrics are almost steady.

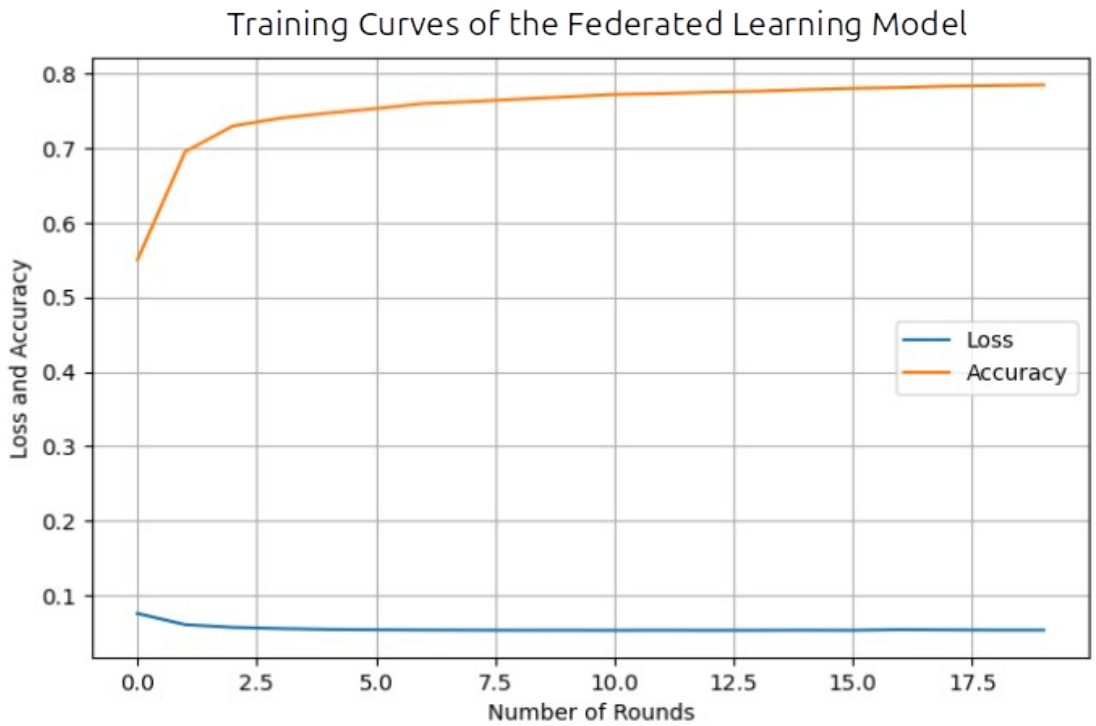

Illustration 10: Training curves of the Federated Learning Model

It must be noted that the training curves do not appear to be oscillating, which indicated that the learning rate is not high.

3.2 TRAINING A TRADITIONAL MODEL

As was mentioned earlier, in order to prove the importance of federated learning in Cyber Security and the success of the trained model, a traditional model must be trained in order to be compared those two.

The traditional model's architecture is identical to the federated model's, as a comparison is valid only when two same models are compared. Hence, the two models have the same number of layers and

number of neurons per layer, same activation functions and the neurons are initialized using the same techniques. Additionally, the dataset that is fed into the neural network contains the same set of features and is also scaled, shuffled, batched and repeated.

The trained neural network has a final training accuracy equal to 95% and training loss equal to 0,088. Also, the model's evaluation on a test set resulted in an accuracy equal to 94% and test loss equal to 0,095.

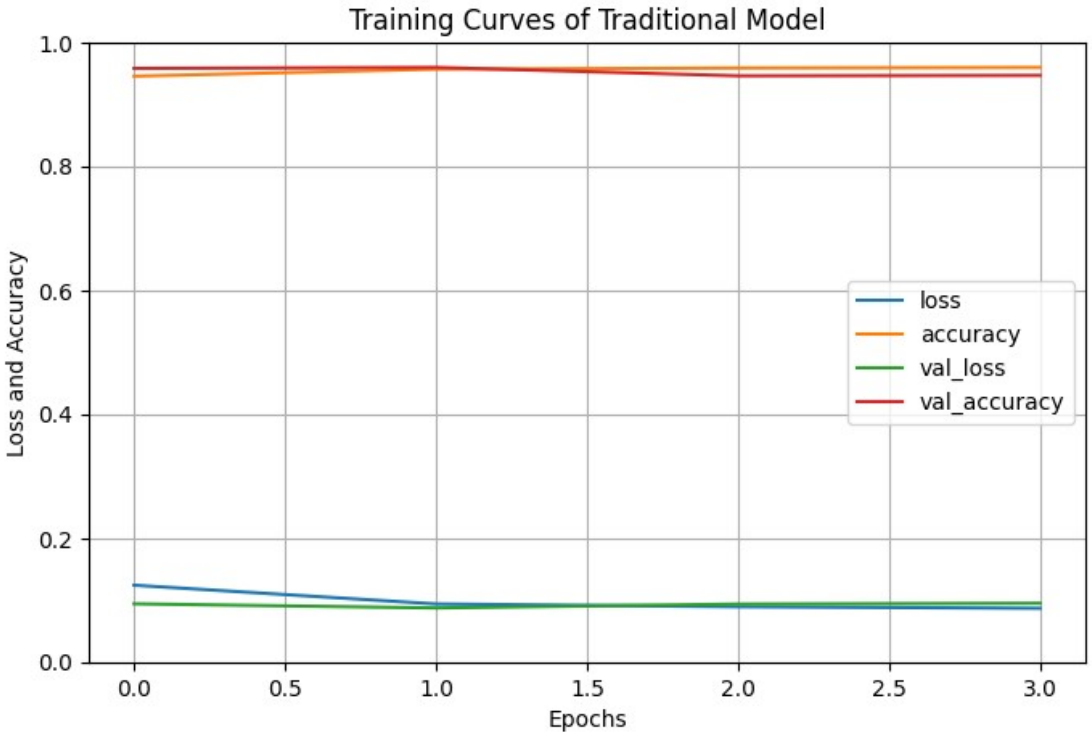

Illustration 11: Training curves of the traditional model

3.3 COMPARISON OF THE TWO MODELS

At length, after having trained the two models, their results must be compared. At the table below, I have collected the results

	FEDERATED LEARNING		TRADITIONAL	
	TRAINING	**EVALUATION**	**TRAINING**	**TEST**
ACCURACY	78,50%	78,77%	95%	94%
LOSS	0.054	0,056	0,088	0,095

Table 9: Model Metrics Comparison

Conclusively, comparing the metrics from the two trained models, it is apparent that the Federated Learning model has worse performance than the traditionally trained model. Of course, as it was

described in earlier sections, this is expected, and the magnitude of that difference should the actual point of comparison. Therefore, since the difference of those metrics is not very large about 18%, it is obvious that the Federated Learning model has value and can be used for correctly recognizing intrusive flows inside a network or a host device.

4 CONCLUSION

As the distribution of the data became wider and the privacy of data was emphasized over the last years, Federated Learning was introduced in order to produce accurate models on heavily distributed data, but also emphasizing on the privacy of the data. So far, Federated Learning has not been used widely in Cyber Security products, but in this work I have taken the initiative to exploit the Federated Learning's advantages and use them in order to develop an accurate Intrusion Detection System that can successfully differentiate a malicious network flow from a benign one.

This work explains thoroughly the Intrusion Detection Systems and the Federated Learning technology, their architecture, their advantages and their potential vulnerabilities, and emphasizes mostly on developing an Intrusion Detection System by training a Feed-Forward Neural Network model using Federated Learning on realistic data on Cyber Security. This trained model was then tested on separated data and verified the Federated Learning is indeed a viable option to use when developing Intrusion Detection Systems, as the difference of accuracy from a traditionally trained model is not big.

Bibliography

[1] Peterson, A. 2016, 'Yahoo discovered hack leading to major data breach two years before it was disclosed', *The Washington Post, 10 November 2016,* Accessed 29 June 2021, <https://www.washingtonpost.com/news/the-switch/wp/2016/11/10/yahoo-discovered-hack-leading-to-major-data-breach-two-years-before-it-was-disclosed/>.

[2] McMillan, R. and Knutson, R. 2017, 'Yahoo Triples Estimate of Breached Accounts to 3', *'The Wall Street Journal'*, 3 October 2017, Accessed 29 June 2021, <https://www.wsj.com/articles/yahoo-triples-estimate-of-breached-accounts-to-3-billion-1507062804>.

[3] European Union 2016. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/EC (general data protection regulation). Accessed 29 June 2021, <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679>.

[4] Silver, D., Huang, A., Maddison, C.J., Guez, A., Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis 2016, 'Mastering the game of Go with deep neural networks and tree search', *Nature 529*, 27 January 2016, pp. 484–503, DOI: <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>.

[5] Bace R., Mell, P., 2001, *Intrusion Detection Systems*, National Institute of Standards and Technology (NIST), Special Publication 800-31, U.S. Government Printing Office, Washington DC, USA.

[6] Stavroulakis P and Stamp M. 2010, *Handbook of ilformation and Communication Security,* Springer-Verlag Berlin Heidelberg, New York, USA.

[7] Liao, Hung-Jen and Lin, Chun-Hung and Lin, Ying-Chih & Tung, Kuang-Yuan 2013, 'Intrusion detection system: A comprehensive review', *Journal of Network and Computer Applications, vol.* 36, pp. 16–24, DOI: <https://doi.org/10.1016/j.jnca.2012.09.004>.

[8] Konecný, J., McMahan, B., Ramage, D. and Richtárik, P. 2016, 'Federated Optimization: Distributed Machine Learning for On-Device Intelligence', *CoRR*, vol. Abs/1610.02527. Accessed 28th August 2020, DOI: <http://arxiv.org/abs/1610.02527>.

[9] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A. Ivanov, V., Kiddon, C., KoneČný, J., Mazzocchi, S., McMahan, H.b., Overveldt, T.V., Petrou, D., Ramage, D., Roselander J. 2019, 'Towards Federated Learning at Scale: System Design', *CoRR,* vol. Abs/1902.01046, Accessed 28th August 2020, DOI: <http://arxiv.org/abs/1902.01046>.

[10] McMahan, B., Moore, E., Ramage, D., Hampson, S. and Arcas, B.A.y. 2016, 'Communication-Efficient Learning of Deep Networks from Decentralized Data', *CoRR,* vol. Abs/1602.05629, Accessed 28th August 2020, DOI: <http://arxiv.org/abs/1602.05629>.

[11] Narayanan and Shmatikov V. 2008, 'Robust De-anonymization of Large Sparse Datasets', *IEEE Symposium on Security and Privacy (sp 2008)*, Oakland, CA, 2008, DOI: <https://doi.org/10.1109/SP.2008.33>.

[12] Rivest, R. L., Adleman, L. and Dertouzos, M. L. (1978), 'On Data Banks and Privacy Homomorphisms', Foundations of Secure Computation, Academia Press , 169-179.

[13] Dwork C., Roth, A. 2014, 'The Algorithmic Foundations of Differential Privacy', *Foundations and Trends® in Theoretical Computer Science*, vol. 9, Accessed 3rd September 2020, DOI: <http://dx.doi.org/10.1561/0400000042>.

[14] Kim, H., Park, J,, Bennis M. and Kim, S. 2020, 'Blockchained On-Device Federated Learning', *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279-1283, Accessed 3rd September 2020, DOI: <https://doi.org/10.1109/LCOMM.2019.2921755>.

[15] Lyu L., Yu L. and Yang Q. 2020, 'Threats to Federated Learning: A Survey', *CoRR,* vol. Abs/2003.02133, Accessed 8th October 2020, DOI: <https://arxiv.org/abs/2003.02133>

[16] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D. & Shmatikov, V. 2020, 'How to Backdoor Federated Learning', *International Conference on Artificial Intelligence and Statistics*, PMLR, vol. 108, pp. 2938-2948, Accessed 8th October 2020, DOI: <https://proceedings.mlr.press/v108/bagdasaryan20a.html>

[17] Ozdayi, M. S., Kantarcioglu, M., and Gel, Y. R. 2020, 'Defending Against Backdoors in Federated Learning with Robust Learning Rate', *CoRR*, vol. Abs/2007.03767, Accessed 9th October 2020, DOI: <https://arxiv.org/abs/2007.03767>

[18] Hewitt, C., Bishop, P. and Steiger, R. 1973, *A Universal Modular ACTOR Formalism for Artificial Intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, USA

[19] University of New Brunswick, *Intrusion Detection Evaluation Dataset (CIC-IDS2017)*, Canadian Institute for Cybersecurity, Accessed 23rd December 2020, <https://www.unb.ca/cic/datasets/ids-2017.html>

[20] Sharafaldin, I., Lashkari, A. H. and Ghorbani, A. A. 2018, 'Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization', *4th International Conference on Information Systems Security and Privacy (ICISSP)*, Purtogal 2018, Accessed 23rd December 2020, <https://www.scitepress.org/Papers/2018/66398/66398.pdf>

[21] Klambauer G., Unterthiner T., Mayr A. and Hochreiter S. 2017, ' Self-Normalizing Neural Networks', *Proceedings of the 31st International Conference on Neural Information Processing Systems*, December 2017, Accessed 15th January 2021, DOI: <https://dl.acm.org/doi/pdf/10.5555/3294771.3294864>

APPENDIX A. TABLES

Feature	Coefficient
Destination Port	0,270
Flow Duration	-0,493
Total Fwd Packets	0,004
Total Backward Packets	0,005
Total Length of Fwd Packets	0,031
Total Length of Bwd Packets	0,003
Fwd Packet Length Max	0,000
Fwd Packet Length Min	0,180
Fwd Packet Length Mean	0,060
Fwd Packet Length Std	-0,042
Bwd Packet Length Max	-0,633
Bwd Packet Length Min	0,396
Bwd Packet Length Mean	-0,641
Bwd Packet Length Std	-0,638
Flow Bytes/s	0,031
Flow Packets/s	-0,159
Flow IAT Mean	-0,390
Flow IAT Std	-0,582
Flow IAT Max	-0,624

Feature	Coefficient
Bwd Packets/s	0,083
Min Packet Length	0,366
Max Packet Length	-0,612
Packet Length Mean	-0,592
Packet Length Std	-0,627
Packet Length Variance	-0,589
FIN Flag Count	-0,349
SYN Flag Count	0,118
RST Flag Count	0,014
PSH Flag Count	0,143
ACK Flag Count	-0,354
URG Flag Count	0,189
ECE Flag Count	0,014
Down/Up Ratio	0,254
Average Packet Size	-0,587
Avg Fwd Segment Size	0,060
Avg Bwd Segment Size	-0,641
Fwd Header Length.1	0,003
Subflow Fwd	0,004

Flow IAT Min	-0,055
Fwd IAT Total	-0,493
Fwd IAT Mean	-0,389
Fwd IAT Std	-0,626
Fwd IAT Max	-0,625
Fwd IAT Min	-0,010
Bwd IAT Total	-0,113
Bwd IAT Mean	-0,113
Bwd IAT Std	-0,287
Bwd IAT Max	-0,265
Bwd IAT Min	0,020
Fwd PSH Flags	0,118
Fwd Header Length	0,003
Bwd Header Length	0,004
Fwd Packets/s	-0,168

Packets	
Subflow Fwd Bytes	0,031
Subflow Bwd Packets	0,005
Subflow Bwd Bytes	0,003
Init_Win_bytes_forward	0,105
Init_Win_bytes_backward	0,128
act_data_pkt_fwd	0,005
min_seg_size_forward	-0,150
Active Mean	-0,035
Active Std	0,008
Active Max	0,006
Active Min	-0,038
Idle Mean	-0,623
Idle Std	-0,094
Idle Max	-0,627
Idle Min	-0,616

Table 10: Point-Biserial Correlation coefficients between every feature and the label

FEATURE NAME	MIN VALUE	MAX VALUE	STD	MEAN	VARIANCE
Destination Port	0	65487	15722,34	5683,12	247191979,65
Flow Duration	0	119999998	42789680,28	28054208,75	1,83E+15
Total Fwd Packets	1	203943	747,9	9,57	559351,78
Total Backward Packets	0	272353	985,13	10,23	970475,67
Total Length of Fwd Packets	0	1224076	6169,4	556,13	38061435,01
Total Length of Bwd Packets	0	627000000	2243276,23	17028,33	5032288246990,39
Fwd Packet Length Max	0	24820	604,23	234,03	365099,29

Fwd Packet Length Min	0	2065	51,11	15,05	2612,55
Fwd Packet Length Mean	0	4640,76	157,77	60,67	24891,64
Fwd Packet Length Std	0	6429,19	226,31	83,05	51216,17
Bwd Packet Length Max	0	19530	2615,38	1664,66	6840234,36
Bwd Packet Length Min	0	1983	64,63	33,89	4177,18
Bwd Packet Length Mean	0	4370,69	797,84	552,97	636547,78
Bwd Packet Length Std	0	6715,74	1098,7	659,87	1207145,12
Flow Bytes/s	-12000000	2070000000	29615636,04	1729533,08	877085898186005
Flow Packets/s	-2000000	3000000	323148,85	99631,51	104425178561,83
Flow IAT Mean	-1	120000000	5600140,27	2507504,34	31361571002778,5
Flow IAT Std	0	84800000	11761290,45	6857157,48	138327953137614
Flow IAT Max	-1	120000000	38417124,25	22936015,55	1,48E+15
Flow IAT Min	-14	120000000	3676678,59	222876,54	13517965424632,7
Fwd IAT Total	0	120000000	42794134,62	27796810,12	1,83E+15
Fwd IAT Mean	0	120000000	11044287,96	5078796,83	121976296616948
Fwd IAT Std	0	83700000	16015980,84	9033598,99	256511642246491
Fwd IAT Max	0	120000000	38459630,8	22841140,85	1,48E+15
Fwd IAT Min	-8	120000000	8874960,24	1032711,25	78764919187902,9
Bwd IAT Total	0	120000000	33378788,6	13890073,05	1,11E+15
Bwd IAT Mean	0	120000000	9602068,7	2652033,19	92199723329310,8
Bwd IAT Std	0	82900000	10747260,2	3529490,75	115503601871038
Bwd IAT Max	0	120000000	26187755,64	9322740,16	685798545542259
Bwd IAT Min	0	120000000	8115415,49	928593,62	65859968577565,2
Fwd PSH Flags	0	1	0,2	0,04	0,04
Bwd PSH Flags	0	0	0	0	0
Fwd URG Flags	0	0	0	0	0
Bwd URG Flags	0	0	0	0	0
Fwd Header Length	-11	4290372	15657,43	242,41	245155244,31
Bwd Header Length	0	5447060	19708,03	249,71	388406384,2

Fwd Packets/s	0	3000000	320133,84	95632,11	102485674484,01
Bwd Packets/s	0	2000000	30947,76	4060,15	957764042,64
Min Packet Length	0	1448	27,54	13,75	758,56
Max Packet Length	0	24820	2635,78	1728,36	6947355,45
Packet Length Mean	0	2279,75	369,28	278,77	136369,25
Packet Length Std	0	4364,02	783,85	524,25	614419,8
Packet Length Variance	0	19000000	1750111,48	889266,88	3062890209342,38
FIN Flag Count	0	1	0,3	0,1	0,09
SYN Flag Count	0	1	0,2	0,04	0,04
RST Flag Count	0	1	0,02	0	0
PSH Flag Count	0	1	0,4	0,19	0,16
ACK Flag Count	0	1	0,49	0,42	0,24
URG Flag Count	0	1	0,25	0,07	0,06
CWE Flag Count	0	0	0	0	0
ECE Flag Count	0	1	0,02	0	0
Down/Up Ratio	0	43	0,57	0,56	0,33
Average Packet Size	0	2612	398,2	306,23	158565,19
Avg Fwd Segment Size	0	4640,76	157,77	60,67	24891,64
Avg Bwd Segment Size	0	4370,69	797,84	552,97	636547,78
Fwd Header Length.1	-11	4290372	15657,43	242,41	245155244,31
Fwd Avg Bytes/Bulk	0	0	0	0	0
Fwd Avg Packets/Bulk	0	0	0	0	0
Fwd Avg Bulk Rate	0	0	0	0	0
Bwd Avg Bytes/Bulk	0	0	0	0	0
Bwd Avg Packets/Bulk	0	0	0	0	0
Bwd Avg Bulk Rate	0	0	0	0	0
Subflow Fwd Packets	1	203943	747,9	9,57	559351,78
Subflow Fwd Bytes	0	1224076	6169,4	556,13	38061435,01
Subflow Bwd Packets	0	272353	985,13	10,23	970475,67
Subflow Bwd Bytes	0	627046409	2243053,59	17026,77	5031289392290,99

Init_Win_bytes_forward	-1	65535	11872,53	5304,72	140956915,73
Init_Win_bytes_backward	-1	65535	7313,62	1476,22	53489061,3
act_data_pkt_fwd	0	197124	715,83	6,13	512406,12
min_seg_size_forward	-1	60	6,32	26,76	39,99
Active Mean	0	100000000	701350,4	92417,82	491892386352,1
Active Std	0	74200000	474648,2	47697,83	225290916715,23
Active Max	0	105000000	1095619,32	163041,58	1200381699170,62
Active Min	0	100000000	605663,36	63270,32	366828106767,56
Idle Mean	0	120000000	38147853,69	22152696,93	1,46E+15
Idle Std	0	76900000	4492672,51	475264,27	20184106318964,6
Idle Max	0	120000000	38506621,73	22563989	1,48E+15
Idle Min	0	120000000	38101294,77	21774503,29	1,45E+15

Table 11: Statistical properties of the features of the dataset