applied sciences

Article

Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study

Davy Preuveneers [1,*], Vera Rimmer [1], Ilias Tsingenopoulos [1], Jan Spooren [1], Wouter Joosen [1] and Elisabeth Ilie-Zudor [2]

1 imec-DistriNet-KU Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium; vera.rimmer@cs.kuleuven.be (V.R.); ilias.tsingenopoulos@cs.kuleuven.be (I.T.); jan.spooren@cs.kuleuven.be (J.S.); wouter.joosen@cs.kuleuven.be (W.J.)
2 MTA SZTAKI, Kende u. 13-17, H-1111 Budapest, Hungary; ilie@sztaki.mta.hu
* Correspondence: davy.preuveneers@cs.kuleuven.be; Tel.: +32-16-327853

Received: 4 November 2018; Accepted: 14 December 2018; Published: 18 December 2018

Abstract: The adoption of machine learning and deep learning is on the rise in the cybersecurity domain where these AI methods help strengthen traditional system monitoring and threat detection solutions. However, adversaries too are becoming more effective in concealing malicious behavior amongst large amounts of benign behavior data. To address the increasing time-to-detection of these stealthy attacks, interconnected and federated learning systems can improve the detection of malicious behavior by joining forces and pooling together monitoring data. The major challenge that we address in this work is that in a federated learning setup, an adversary has many more opportunities to poison one of the local machine learning models with malicious training samples, thereby influencing the outcome of the federated learning and evading detection. We present a solution where contributing parties in federated learning can be held accountable and have their model updates audited. We describe a permissioned blockchain-based federated learning method where incremental updates to an anomaly detection machine learning model are chained together on the distributed ledger. By integrating federated learning with blockchain technology, our solution supports the auditing of machine learning models without the necessity to centralize the training data. Experiments with a realistic intrusion detection use case and an autoencoder for anomaly detection illustrate that the increased complexity caused by blockchain technology has a limited performance impact on the federated learning, varying between 5 and 15%, while providing full transparency over the distributed training process of the neural network. Furthermore, our blockchain-based federated learning solution can be generalized and applied to more sophisticated neural network architectures and other use cases.

Keywords: blockchain; federated deep learning; anomaly detection; audit; performance

1. Introduction

Organizations and industries are faced with a growing number of cyber-attacks on their systems, services, and infrastructure. Adversaries target anything of value including general platform and service providers, as well as businesses in vertical domains such as healthcare, manufacturing, finance retail, etc. These cyber-attacks aim at various types of damage, such as service disruption, theft of sensitive or competitive information, user credentials, etc. As the pace of the digital transformation increases, so does the impact of cyber-attacks. To name one example: the Industrial Internet of Things (IIoT) and Industry 4.0 paradigms [1,2] envision manufacturing companies collaborating in networked production systems. These cyber-physical production systems (CPPS) have their embedded sensors linked with cloud architectures where the gathered data are used for predictive maintenance and production optimizations. As the BrickerBot malware infecting 10 million IoT devices has shown (https://www.bleepingcomputer.com/news/security/brickerbot-author-retires-claiming-to-have-bricked-over-10-million-iot-devices/), an unknown vulnerability in one of the connected systems offers an ideal opportunity for an attacker to deploy malware that quickly spreads through the network. Maersk, one of the world's largest shipping companies, suffered massively from 2017's NotPetya malware outbreak (https://www.bleepingcomputer.com/news/security/maersk-reinstalled-45-000-pcs-and-4-000-servers-to-recover-from-notpetya-attack/), needing to reinstall 45,000 PCs, 4000 servers, and 2500 applications and estimating the cost to its business at $250 million–$300 million.

To avoid causing the organization great financial and reputational harm, cyber-security professionals have to respond in a timely and effective manner to such threats. As security experts are becoming increasingly concerned about the accelerating pace of change and sophistication of the threat landscape, they have to rely on an expanding arsenal of security services to handle known and unknown threats adequately. The growing complexity of quickly and effectively managing vulnerabilities and remediating IT environments has motivated a growing number of security professionals and service providers to invest in security analytics [3] tools to gather, filter, integrate, and link diverse types of events to gain a holistic view of the business processes and security of an organization's infrastructure (see Figure 1). Advanced security analytics technologies help them cope with a variety of (un)known attacks and (un)known threat actors during different stages of a cyber-attack lifecycle. Such security analytics solutions usually encompass machine learning (ML) methods for anomaly detection [4] to identify unusual patterns of behavior that do not conform with the expected activities of the enterprise business processes. Beyond the necessity to apply well-known and important ML techniques to adapt to emerging security threats (e.g., finding the best configurations through automated parameter optimization) dynamically, there is a need to improve the robustness of ML-based security. As pointed out by Verma [5], any security analytics solution must deal with (1) an active adversary, (2) the base rate fallacy, (3) the asymmetrical costs of misclassification, (4) potential poisoning of datasets, and (5) the attack time-scale. Furthermore, they must handle sensitive or confidential information used as input (e.g., in ML-based anomaly detection) effectively.

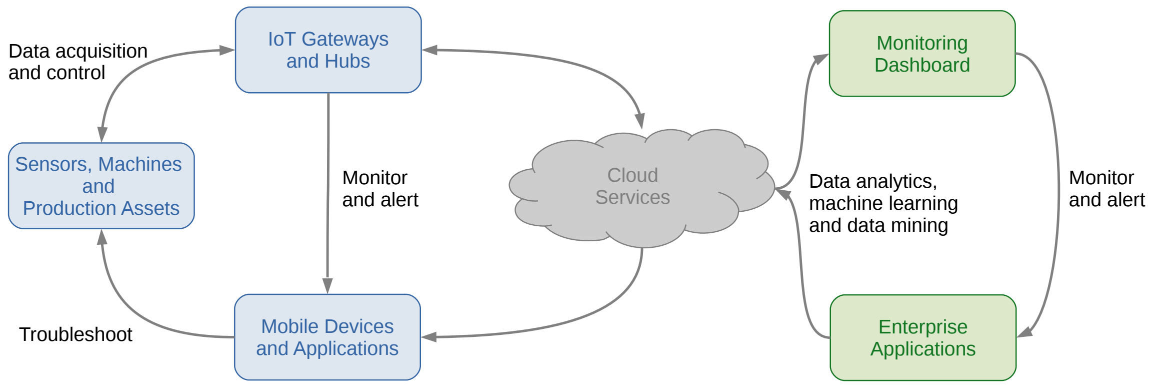
Data acquisition and control
IoT Gateways and Hubs
Sensors, Machines and Production Assets
Monitor and alert
Cloud Services
Troubleshoot
Mobile Devices and Applications
Monitoring Dashboard
Data analytics, machine learning and data mining
Monitor and alert
Enterprise Applications

Figure 1. Big Data analytics and security intelligence for Industry 4.0 applications.

In this work, we address the fourth and fifth challenge with a blockchain-based federated deep learning method. A key property of federated learning is that it does not require the centralization of training data, which has the advantage of increased the efficiency and confidentiality of the training data. We leverage the distributed ledger to record updates to machine learning models in a manner that is secure, transparent, highly resistant to outages, and auditable. By combining federated learning with blockchain technology, we enable accountability for collectively-learned machine learning models. The non-repudiation property of blockchains increases the reliability of the federated learning solution.

The mere adoption of blockchain technology does not provide this capability out of the box, as the only thing a blockchain offers is a distributed immutable ledger. With our solution, the traceability of transactions—in our case, updates to a deep neural network model—is exploited for the detection of tamper attempts or fraudulent transactions of adversaries aiming to evade anomaly detection. Indeed, every model update—or local weight or gradient updates for deep learning—can be cryptographically associated with an individual. Furthermore, it guarantees the integrity of incrementally-learned machine learning models by cryptographically chaining one machine learning model to the next.

The main contribution of this work is evaluating the practical feasibility of auditing a federated deep learning process by using blockchains, demonstrated in a realistic use case on intrusion detection and a real-world dataset (i.e., CICIDS2017). We find that our proof-of-concept implementation—on top of the MultiChain (https://www.multichain.com) open source blockchain platform—has a limited relative performance impact and network overhead, while providing full transparency over the distributed deep learning process.

In Section 2, we review relevant related work and identify the gap in the current state-of-the-art. Section 3 describes our novel approach towards accountable federated learning using blockchain technology to ensure non-repudiation with respect to machine learning model updates. We present the evaluation results of our proof-of-concept implementation using an anomaly detection use case in Section 4. We conclude this work in Section 5, summarizing our main findings and highlighting opportunities for future work.

2. Related Work

This section reviews relevant related work in the area of federated machine learning with a particular focus on deep learning, typical attacks on machine learning methods, and the use of machine learning in combination with blockchains.

2.1. Federated Machine Learning and Deep Learning

Machine learning (ML)-based anomaly detection [4] requires sufficient amounts of data to establish a baseline of normal behavior, expected noise, and abnormal deviations. In complex environments where multiple entities must be monitored for anomalies, centralizing all data for training and testing purposes may not be feasible due to (1) resource constraints, (2) security and privacy concerns, or (3) unacceptable communication latencies to transfer the raw data. Federated ML [6] distributes the workload by enabling multi-party on-device learning. As federated learning is still a relatively new technique, its applicability to optimize anomaly detection for resource- constrained environments and the implications on the achieved performance and accuracy trade-off is still unclear and needs to be studied. Deep learning has been investigated for federated ML [7] and has been applied to find anomalies in system logs [8]. While federated ML minimizes the amount of raw data to be shared or centralized, it can be considered as a tactic to preserve security and privacy. However, recent research shows that data leakage is theoretically still possible in such collaborative learning environments [9].

2.2. Intrusion Detection and Response Systems

In our work, we use intrusion detection [10] as a motivating use case for our blockchain-based federated learning solution rather than as the core topic and contribution of our research. As such, we are not aiming to provide an extensive survey about the related work on the matter, but discuss the following works as ongoing research in the area of intrusion detection. Inayat et al. [11] provided an extensive overview of intrusion detection and intrusion response systems. They provided a taxonomy of different design parameters and classify existing schemes. Additionally, they identified which of the design parameters are crucial to mitigate attacks in real time. Their contribution mainly focused on the response design parameters to improve the decision-making process of the response according to the statistical features of attacks. They argued that many schemes disregard semantic coherence and dynamic response parameters as key parameters, and therefore produce false alarms. They also pointed out that the research on a fully-automated intrusion response system is still in its infancy.

Karim et al. [12] provided a comparative experimental design and performance analysis of snort-based intrusion detection systems in realistic computer networks. They developed an effective testbed with different types of operating systems in an enterprise network consisting of multiple virtual local area networks, generating various network traffic scenarios involving a.o. high data speed, heavy traffic, and large packet sizes. They presented a centralized parallel snort-based network intrusion detection system (CPS-NIDS) and illustrated how their solution enhanced the performance in terms of a reduced number of dropped packets while facilitating network security management to access and keep records of all attack behaviors.

As our goal is not to improve the detection mechanisms of intrusion detection systems themselves, we rather refer to various relevant surveys on this particular topic [13–16] covering more than two decades of research that inspired the motivating use case of our work.

2.3. Machine Learning in Adversarial Settings

While progress has been made in the past few years in the domain of advanced machine learning techniques for security—including deep learning and data/pattern mining—several key challenges remain for which the state-of-the-art is yet to produce an adequate answer. The machine learning (ML) algorithms used in security analytics applications are also subject to attacks. As the interest in and use of machine learning for security applications increases, so will the awareness of cyber-criminals. When frequently updating an ML model to account for new threats, malicious adversaries can launch causative/data poisoning attacks to inject misleading training data intentionally so that an ML model becomes ineffective. For example, various poisoning attacks on specific ML algorithms [17,18] were able to bypass intrusion detection systems. More recently, Chen et al. [19] demonstrated how to automate poisoning attacks against malware detection systems. However, keeping ML models fixed can give rise to exploratory/evasion attacks to find the blind spots. Understanding the trade-offs between either keeping a machine-learning model fixed or updating the model, taking into account its complexity, is non-trivial.

Wang [20] investigated how deep learning can be applied for intrusion detection in the presence of adversaries. Given the fact that deep neural networks have been demonstrated to be vulnerable to adversarial examples in the image classification domain, he argued that deep neural networks also leave opportunities for an attacker to fool the networks into misclassification. He investigated how well state-of-the-art attack algorithms performed against deep learning-based intrusion detection, on the NSL-KDD dataset. He discovered different levels of effectiveness, as well as varying degrees of significance across features in terms of their rates of being selected to be perturbed by an adversary.

More recently, similar research was done by Huang et al. [21] in the area of intrusion detection for software-defined networks (SDN). They conducted SDN-based experiments with adversarial attacks against a deep learning detecting system, evaluating the fast gradient sign method (FGSM), Jacobian-based saliency map attack (JSMA), and JSMA reverse (JSMA-RE) schemes as perturbation functions for the adversarial attacks. They determined that JSMA is more suitable for use in perturbing testing data, whereas FGSM is more suitable for perturbing the training model.

2.4. Machine Learning and Blockchains

Given all the excitement around deep learning and blockchains over the past few years, it is not surprising that researchers and business developers are exploring both disruptive technologies to understand how AI can help blockchains and vice versa and how both paradigms may converge in the future. One such example is the emergence of AI marketplaces, such as SingularityNET (https://singularitynet.io), Effect.ai (https://effect.ai), and Cerebrum (https://cerebrum.world), where blockchains are used to run AI algorithms in a decentralized manner.

With the hype around cryptocurrencies and the development of decentralized applications on public blockchains, a straightforward example is the use of machine learning for automated trading of cryptocurrencies [22,23]. Chen et al. [24] used machine learning to detect Ponzi schemes on the blockchain using various features of smart contracts and user accounts. With their XGBoostclassification model, they were able to estimate more than 400 Ponzi schemes running on Ethereum.

More related to our work is the anomaly detection research done by Bogner et al. [25]. They presented an IoT solution for monitoring interdependent components. It uses online machine learning for anomaly detection optimized for interpretability, combined with the public Ethereum blockchain to guarantee data integrity. Meng et al. [26] focused on intrusion detection systems and collaborative intrusion detection networks. Their motivation for adopting blockchain technology was that the latter can protect the integrity of data storage and ensure process transparency. They reviewed the intersection of blockchains and intrusion-detection systems and identified open challenges. DeepChain by Weng et al. [27] specifically focuses on federated learning and privacy-preserving deep learning. They addressed a threat scenario including malicious participants that may damage the correctness of training, a concern also relevant in our work. DeepChain gives mistrustful parties incentives to participate in privacy-preserving learning and share gradients and update parameters correctly. DeepChain addresses privacy requirements which are beyond the scope of our work. As a consequence of these additional requirements, their solution relies on the availability of trusted execution environments—like the Intel Software Guard Extensions (SGX)enclaves—and expensive cryptographic functions—such as homomorphic encryption—to compute model updates. As a result, their federated learning experiments on a single workstation node using the MNISTdataset demonstrated a rather significant computational complexity, which makes the proposed solution not practical for our use case.

2.5. Bridging the Gap

While various related works have already addressed some of the challenges identified earlier in the Introduction, our research complements the state-of-the-art by not only investigating the auditability of local gradient updates in a federated deep learning setting for an anomaly detection use case, i.e., intrusion detection, but also demonstrating the practical feasibility in a more realistic distributed deployment environment. That is why our experimental setup will also evaluate our proof-of-concept implementation from a performance impact perspective by measuring the relative computational overhead of blockchain interactions in a multi-node deployment setup, including resource- constrained devices.

3. Accountable Federated Learning for Anomaly Detection

In this section, we will first elaborate on the machine learning model we chose for anomaly detection in our intrusion detection use case. We will then proceed on how to extend the concept towards federated deep learning with an iterative model averaging technique and accountability on the blockchain.

3.1. Intrusion Detection with a (Deep) Neural Network-Based Anomaly Detection

Our motivating use case features a network intrusion detection scenario with multiple interconnected devices, possibly spanning multiple collaborating enterprises. For our evaluation, we make use of the Intrusion Detection Evaluation Dataset (CICIDS2017) (https://www.unb.ca/cic/datasets/ids-2017.html). This dataset contains network traffic information from 12 machines containing benign traffic and common attacks collected throughout a period of five days. The malicious attacks include brute force FTP, brute force SSH, DoS, heartbleed, web attacks, infiltration, botnet, and DDoS. More details of the dataset and the underlying principles of its collection can be found in the corresponding paper [28].

There is a large body of work on using artificial (deep) neural networks for anomaly detection [29,30] and intrusion detection [31–33] that covers at least two decades of research. This previous research compared different approaches in terms of the best classification accuracy or recognition rates. The goal of our work, however, is to measure the relative computational impact of accountability through blockchains in a federated scenario. We therefore assume a secure environment equipped with an intrusion detection system with a machine learning model at the core of it, performing in a federated manner.

For the task of intrusion detection itself, we adopt a one-class classification neural network for unsupervised anomaly detection, as proposed in the work by Raghavendra et al. [29]. In such a scenario, the assumption is that we have training data only for one class (i.e., normal baseline behavior of benign traffic), but during testing, we may encounter anomalies or outliers, representing deviations in normal traffic, which may be caused by cyber-attacks against the enterprise network. As most neural network models are discriminative—i.e., they need data belonging to all possible classes to train a classifier—we use autoencoders [34] that typically learn a representation or an encoding of a set of data in an unsupervised manner. As such, a trained autoencoder is able to recognize the data sample belonging to the baseline class (benign traffic in our use case) and mark any test sample that does not fit into the nature of the data seen during training as an anomaly (a potential cyber-attack). A high-level approach is then as follows: the autoencoder uses a certain metric—a loss function—to express how well a test sample fits into the baseline class. In order to identify whether a test sample belongs to the baseline class or represents an anomaly, a threshold has to be defined on top of this metric. If a loss function value of the test sample exceeds a pre-defined threshold, this sample is therefore recognized by the autoencoder as an anomaly, and the intrusion detection system raises an alert.

Figure 2 represents a simplified general structure of a (deep) autoencoder with an input layer, an output layer, multiple hidden layers in the encoding and decoding parts, and a compressed layer in the middle. An autoencoder that learns such a condensed vector in the middle is used for the task of representation learning. Its architecture imposes a bottleneck that forces computing a compressed representation of the input data. The decoder part will subsequently learn how to reconstruct the original input from the compressed feature vector. An autoencoder that was trained on data of the same structure (such as benign traffic traces in our use case) will successfully learn the correct compressed representations, which can be used to reconstruct outputs that closely resemble the inputs. The way that these representations can be learned is by minimizing the loss function: a reconstruction error, computed using a distance metric, such as the L_2 norm, between the original input layer x and the reconstructed output layer x':

$$\mathcal{L}_2 = \sqrt{\|x - x'\|^2}$$

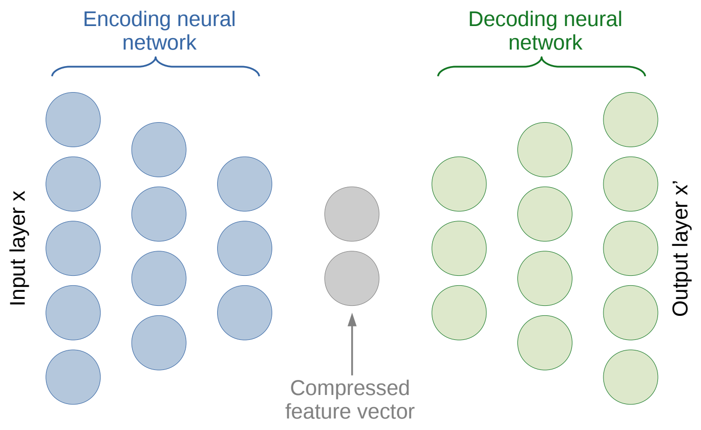

Figure 2. Generic architecture of an autoencoder learning a compressed representation of benign network traffic.

During training on the baseline class, a reconstruction error also serves as a measure of how well the decoder network performs. In our experiments, for the training phase of the autoencoder, we split the benign traffic of the first day in the CICIDS2017 5-day dataset into two parts, one for training and the other one for validation, to avoid overfitting to the training data. As a result, an autoencoder learns the structure and correlations intrinsic to the benign traffic. The data of the four remaining days in the CICIDS2017 dataset, including both benign and malicious traffic, are used for testing. During testing, the compressed feature vector generated by the autoencoder is used for detecting anomalies. Under the assumption that malicious traffic deviates significantly from benign traffic, benign network flow test samples should have a low reconstruction error, while malicious traffic test samples result in a high reconstruction error. In order to distinguish between the benign and malicious test samples, one has to define a threshold on the reconstruction error. A test sample reconstructed with an error below a certain threshold would be considered benign, whereas a test sample with a reconstruction error above the threshold would be marked as an anomaly, possibly indicating malicious network traffic. A typical value for this threshold is the maximum reconstruction error observed on benign traffic during the training phase of the autoencoder.

In order to achieve optimal performance in any particular machine learning application, the hyperparameters of the (deep) neural network have to be empirically tuned. These hyperparameters include the number of layers, the number of neurons per layer, the learning rate of the model, regularizers, and other affecting parameters. Since the goal of this research is to illustrate the general approach of how to integrate federated learning with blockchains to create a distributed and immutable audit trail, we chose a rather common parameterization of our autoencoder and only tuned the main architectural parameters. Namely, we varied the number of hidden layers, from a shallow autoencoder having just one hidden layer up to deeper autoencoders with three hidden layers, and the number of neurons in the hidden layers, and tested each architecture on our dataset. The CICIDS2017 dataset we use in our experiments provides preprocessed network flow samples composed of 78 traffic features (an overview of the available features is provided in Table 1). We selected a subset 50 features and normalized the values between 0.0 and 1.0. A network flow input sample is fed into the autoencoder, converted into a compressed feature vector, and decoded back into a network flow output sample. As the provided feature vectors are initially rather high-level, it is not necessary to utilize very deep neural networks for further feature extraction. Indeed, as will be shown in the evaluation, an autoencoder with just three hidden layers is able to perform accurate anomaly detection on our data. Results reported in this paper were achieved under the following neural network configuration and parameters:

- two symmetrical encoding and decoding parts each with three dense layers;
- the total amount of five dense layers including three hidden layers;
- the number of neurons per layer $50 \rightarrow 25 \rightarrow 10 \rightarrow 25 \rightarrow 50$;
- the total amount of learnable weights $(25 \times 50) + (10 \times 25) + (25 \times 10) + (50 \times 25) = 3000$;
- a rectified linear unit (ReLU) as the activation function;
- the Adam stochastic gradient descent (SGD) optimization algorithm;
- the learning rate of 0.05;
- L_2 norm (or the mean squared error) as the loss function;

A different autoencoder structure or optimization algorithm may lead to better recognition results or a faster learning process, but these are not expected to influence the relative overhead of the blockchain interactions. The threshold to distinguish between benign and anomalous traffic optimally may change as well. However, only if the number of weights changes, there will be an impact on the network traffic towards both the parameter server and the blockchain.

Note that the approach we describe in our paper is straightforwardly generalized to any neural network that uses backpropagation, including more sophisticated deep learning algorithms such as the feedforward convolutional neural network (CNN). The main impact on the federated learning algorithm in the case of substituting the neural network or adding more layers would be a change in the number of weights to share during model aggregation, e.g., in the case of intrusion detection directly on raw network traffic data rather than on 78 high-level features, more complex deep neural networks would be required to achieve anomaly detection rates comparable to the state-of-the-art. We refer to other recent related work using the same CICIDS2017 dataset for more details [35,36].

Table 1. Network flow features of the CICIDS2017 dataset.

Destination Port	Flow Duration	Total FwdPackets	Total Backward Packets
Total Length of Fwd Packets	Total Length of BwdPackets	Fwd Packet Length Max	Fwd Packet Length Min
Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	Bwd Packet Length Min
Bwd Packet Length Mean	Bwd Packet Length Std	Flow Bytes/s	Flow Packets/s
Flow IATMean	Flow IAT Std	Flow IAT Max	Flow IAT Min
Fwd IAT Total	Fwd IAT Mean	Fwd IAT Std	Fwd IAT Max
Fwd IAT Min	Bwd IAT Total	Bwd IAT Mean	Bwd IAT Std
Bwd IAT Max	Bwd IAT Min	Fwd PSHFlags	Bwd PSH Flags
Fwd URGFlags	Bwd URG Flags	Fwd Header Length	Bwd Header Length
Fwd Packets/s	Bwd Packets/s	Min Packet Length	Max Packet Length
Packet Length Mean	Packet Length Std	Packet Length Variance	FINFlag Count
SYNFlag Count	RSTFlag Count	PSH Flag Count	ACK Flag Count
URG Flag Count	CWEFlag Count	ECEFlag Count	Down/Up Ratio
Average Packet Size	Avg Fwd Segment Size	Avg Bwd Segment Size	Fwd Header Length
Fwd Avg Bytes/Bulk	Fwd Avg Packets/Bulk	Fwd Avg Bulk Rate	Bwd Avg Bytes/Bulk
Bwd Avg Packets/Bulk	Bwd Avg Bulk Rate	Subflow Fwd Packets	Subflow Fwd Bytes
Subflow Bwd Packets	Subflow Bwd Bytes	Init_Win_bytes_forward	Init_Win_bytes_backward
act_data_pkt_fwd	min_seg_size_forward	Active Mean	Active Std
Active Max	Active Min	Idle Mean	Idle Std
Idle Max	Idle Min		

For initial testing, we used the high-level Python-based Keras (https://keras.io) neural network library (on top of Tensorflow). For the implementation of the federated learning variant of the autoencoder, we used the Deeplearning4j (DL4J) (http://deeplearning4j.org) library, which offers a Java API to implement deep artificial neural networks and uses a C++ backend for the computation. DL4J can import neural network models originally configured and trained using Keras. Furthermore, DL4J easily integrates with Java-based Big Data frameworks like Hadoop and Spark and mobile operating systems like Android.

The federated learning component was implemented as a Spring Boot 2.0 application (https://spring.io/projects/spring-boot) that runs within a Java 8.0 virtual machine. Each Spring Boot application instance provides a RESTful interface through which it processes one or more network flow traces of the CICIDS2017 dataset in order to learn a local learning model for the autoencoder. Each application is deployed on one and up to 12 desktop nodes and has its own identifier to select a proper subset of the dataset to use in the training. Additional functionality of this Spring Boot application will be described in the next subsections.

3.2. Anomaly Detection through Federated Learning

Rather than learning an anomaly detection model and evaluating it on a single high-end machine (as done in DeepChain [27]), we partition and distribute the CICIDS2017 dataset on a real network of desktop nodes and resource-constrained devices. They all will participate in the federated learning setup. This setup is depicted in Figure 3. In a centralized learning environment, the low-end and high-end systems, whose network traffic is being monitored, send their data to a centralized deep learning node that trains the neural network model. In a federated learning environment, K clients (acting as gateways for the monitored systems) learn a local model. Each of the $k = 1..K$ models has the same structure (i.e., number of layers and neurons per layer), but they are trained with different datasets originating from their own connected clients. Figure 4 depicts a high-level overview of the algorithm, with the pseudocode listed in Algorithm 1.

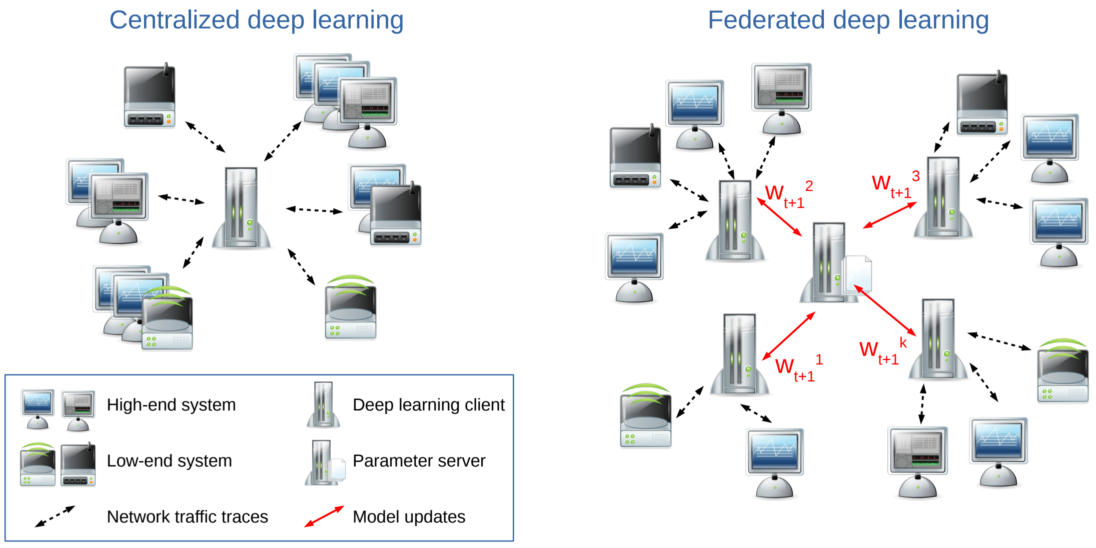

Figure 3. Centralized versus federated learning (local optimization and sharing of new weights with parameter server).

Algorithm 1 *FederatedAveraging*. C is the global batch size; B is the local batch size; the K clients are indexed by k; E is the number of local epochs; and η is the learning rate (adapted from [37]).

```
1:  procedure SERVERUPDATE:
2:      initialize $w_0$
3:      for each round $t = 1, 2, ...$ do
4:          $m \leftarrow \max(C \cdot K, 1)$
5:          $S_t \leftarrow$ (subset $m$ of clients)
6:          for each client $k \in S_t$ in parallel do
7:              $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
8:          end for
9:          $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
10:     end for
11: end procedure

12: procedure CLIENTUPDATE($k, w$)
13:     // Run on client $k$
14:     $B \leftarrow$ (split data $k$ into batches of size $B$)
15:     for each local epoch $i$ from 1 to $E$ do
16:         for batch $b \in B$ do
17:             $w \leftarrow w - \eta \nabla F_k(w)$
18:         end for
19:     end for
20:     return $w$ to server
21: end procedure
```

To initiate the federated learning scheme, at $t = 0$, the parameter server initializes a master model with a first set of weights w. Second, each of the $k = 1..K$ deep learning clients downloads this master model from the parameter server. Third, each of the $k = 1..K$ deep learning clients computes in parallel a new local set of weights w_{t+1}^k using the training data of its connected monitored systems. Last, the parameter server updates the weights of the master model w_{t+1} by aggregating the weights w_{t+1}^k of each client k (by weighted averaging) to create an improved master model. The cycle repeats, and a new epoch is started until a certain stop criterion is reached.

Our methodology follows the design principles for federated learning using model averaging as proposed by McMahan et al. [6,37]. They formalize the notion of federated learning as a problem of federated optimization where at time $t \geq 0$, the parameter server distributes the model w_t to (a subset of) K clients of the federation. In order for the model averaging to have a similar expected learning capacity as the centralized case, three assumptions have to hold: (a) all the clients share the same initial weights; (b) the weight updates are disseminated to the clients synchronously and irrespective of their participation in the last global epoch; (c) the learning rate is common amongst all the clients. With these assumptions holding, the only variable between the federated and the centralized mode is how apart the distributions of the local datasets are. Approaching it from the global scope of the server, we would reframe the question as how independent and identically distributed (henceforth IID) are the mini-batches.

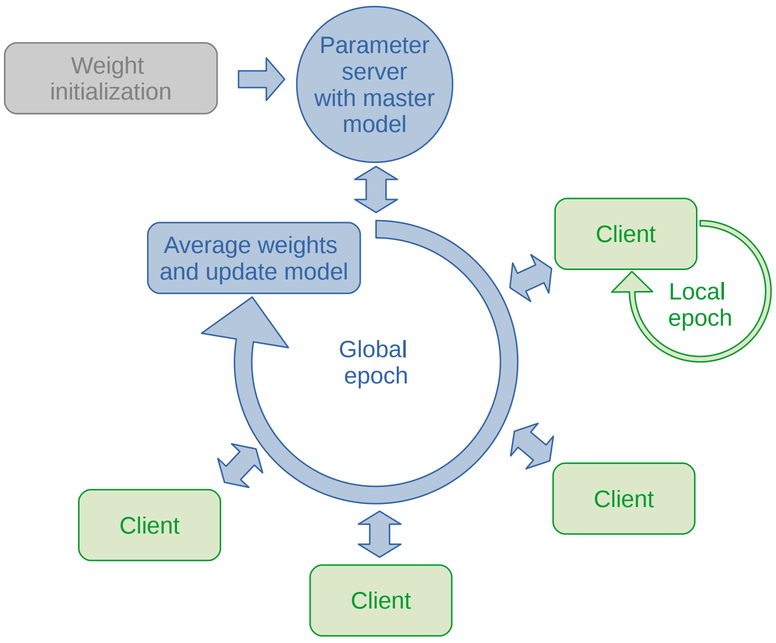

Figure 4. Flowchart of the federated aggregation of the local weight updates into the master model.

After we fix the learning rate η, each client $k = 1..K$ downloads the weights from the parameter server and then independently calculates $g_t^k = \nabla F_k(w_t)$, the average gradient on its local data at the current model w_t. The clients can perform this in batch or mini-batch mode and for more than one epoch. Then, the server aggregates these gradients and applies the following update:

$$w_{t+1} \leftarrow w_t - \eta_t \nabla f(w_t) \quad \text{with} \quad f(w_t) = \sum_{k=1}^{K} \frac{n_k}{n} F^k(w_t) \quad \text{or} \quad \nabla f(w_t) = \sum_{k=1}^{K} \frac{n_k}{n} g_t^k$$

where $f(w_t)$ is the loss function, w_t the model parameters at time t, and n_k the size of the data partition of client k so that $n = \sum_k n_k$. In equal manner for each client k, the final weight update becomes:

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta g_t^k \qquad w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$$

At the end of each local update, the parameter server computes the weighted average of the local weights w_{t+1}^k. At this point, we need to address the underlying distribution of each local dataset. The work by McMahan et al. [37] shows that model averaging, with the aforementioned assumptions holding, is robust towards unbalanced and non-IID data distributions, with a certain overhead in required training epochs to achieve similar model performance. For this intrusion detection use case that learns an autoencoder model based on features of benign network traffic, we can establish a claim for approximately-IID data given that the infrastructure, protocols, and benign user behavior are largely the same for all clients. Moreover, when we consider this unsupervised learning case, the federated gradient descent finds a lower dimensional manifold where the benign traffic data collectively lie, which implies that there is a distribution that describes and generates them, irrespective of what client from which they originate. Ultimately, the efficacy of the federated averaging algorithm compared to the centralized version is validated in non-IID use-cases, which provides a lower bound on expected performance.

Considering the potential upscaling of the federation, for a large value of K and especially when the deep learning algorithms would be directly run on a heterogeneous set of powerful and resource-constrained clients, the federated learning algorithm can be dynamically adapted to collect weight updates w_{t+1}^k from a smaller random subset of the K nodes. Otherwise, computing the federated average could significantly delay the computation of an updated deep learning model if the algorithm has to always wait for the slowest contributor.

In our experimental setup, we can vary the number of deep learning clients from 1–12 as the CICIDS2017 dataset contains network flow traces of 12 machines being monitored. With just one deep learning client, as shown in Figure 3 on the left, we end up with a centralized learning configuration where each of the 12 monitored systems is connected to the same deep learning client. The other extreme configuration has 12 deep learning clients and a parameter server. In that setting, each monitored system is connected to a single deep learning client. Each of the deep learning clients is connected to the parameter server. In such a deployment and monitoring scenario, the monitored system and deep learning client are unified, such that each client node in the network learns its own autoencoder based solely on its own network traffic data. In an intermediate deployment configuration, as depicted in Figure 3 on the right, the 12 monitored systems are connected to a few deep learning clients, which in turn are connected to the parameter server. The CICIDS2017 dataset is partitioned in such a way that each deep learning client processes the network flows of 2–4 monitored systems. For all scenarios, the goal is that each deep learning client can recognize anomalous traffic or intrusions, even if those attacks were not present in the data it used as input for its own local autoencoder.

Our Spring Boot 2.0 application provides a RESTful interface, not only to retrieve network flow traces as discussed in the previous subsection, but also to collect and compute the aggregation of the weights to produce the updated deep learning model, i.e., an improved autoencoder model. However, this part of the functionality is only used on the node that acts as the parameter server.

3.3. Federated Learning in an Adversarial Setting

In an adversarial setting, one or multiple nodes in the network may be compromised. The adversary can launch data poisoning attacks on the federated machine learning by altering a small fraction of the training data or the weight updates in order to make the global trained classifier satisfy the objective of the attacker whose goal is to remain undetected. The following attack scenarios are possible, sorted in terms of their security threat impact:

- A monitored node is compromised and provides malicious network flow features such that the connected deep learning client optimizes its local model based on erroneous traffic data.
- A deep learning client is compromised and provides malicious weight updates to the parameter server and/or does not adopt new model updates broadcast by the parameter server.
- The parameter server is compromised and provides malicious neural network model updates back to the deep learning clients.

Obviously, more complex attack scenarios involve a combination of these nodes being compromised. However, the assumption of this threat model is indeed that at least one node is under the control of the attacker. A second assumption is that the benign traffic data is not too noisy so that the maximum of the reconstruction error on the training set can be taken as a threshold to identify intrusions as anomalies.

There are various techniques to execute poisoning attacks against machine learning algorithms. This is beyond the scope of this work, but we refer interested readers to the related work on poisoning attacks and defenses [38–40] for more details.

3.4. Auditing the Anomaly Models and Weight Updates on the Blockchain

Given that the parameter server is the most security-sensitive component in the federated learning setup, we will replace the direct interaction with this server with a blockchain. This way, all deep learning clients and monitored systems can interact with the blockchains such that all interactions and autoencoder model updates are traceable. The peer-to-peer architecture of a blockchain has the additional advantage that the parameter server does not become a single point of failure, as each node has a full copy of the ledger and can compute the aggregated weight updates.

Given that the federated learning process involves a large amount of weight updates, a public blockchain that relies on a proof-of-work consensus protocol, such as Bitcoin or Ethereum, is not feasible as the generation of new blocks on the ledger would be too slow. We therefore rely on a private and permissioned blockchain that supports multiple assets and mining without proof-of-work. The MultiChain [41] open source blockchain platform and its round-robin-based mechanism for permitted miners to create new blocks meets the aforementioned prerequisites.

For our proof-of-concept implementation, we deploy MultiChain 2.0 (alpha 4) on 12 nodes and deploy an *ids*chain with the following commands on node *host01*with IP address 192.168.100.24:

```
host01:~$ multichain-util create ids
host01:~$ multichaind ids -daemon
```

In order for *host02* to connect to the *ids* chain on the blockchain, we grant its wallet address *1anp8AquvvPi87LP5BQ3HPAKNDRnwKbb99fLTu*the permission as follows:

```
host01:~$ multichain-cli ids grant 1anp8AquvvPi87LP5BQ3HPAKNDRnwKbb99fLTu connect
host02:~$ multichaind ids@192.168.100.24:6469 -daemon
```

We repeat the above process for nodes *host03*–*host12* where each of these nodes has its own wallet address. We then create a stream to store all updates to a machine learning model *anomaly*, store an initial model of the autoencoder in JSON format under the *init*key, and grant all other nodes—from *host02*–*host12*—the permission to publish to the *anomaly* stream:

```
host01:~$ multichain-cli ids create stream anomaly false
host01:~$ multichain-cli ids publish anomaly init '{"json":{"nb_layers": 5, ...}}'
host01:~$ multichain-cli ids grant 1anp8AquvvPi87LP5BQ3HPAKNDRnwKbb99fLTu send
host01:~$ multichain-cli ids grant 1anp8AquvvPi87LP5BQ3HPAKNDRnwKbb99fLTu anomaly.write
```

Each node can then subscribe to the stream and query all items or only those with a specific key:

```
host01:~$ multichain-cli ids subscribe anomaly
host01:~$ multichain-cli ids liststreamitems anomaly # all items
host01:~$ multichain-cli ids liststreamkeyitems anomaly init # items with the key 'init'
```

On the second node *host02*, we publish a local model update as follows:

```
host02:~$ multichain-cli ids publish anomaly host02_1 '{"json":{"deltaweights": [ 0.1, -0.23, ... ]}}'
```

MultiChain implements a consensus algorithm based on block signatures and a customizable round-robin consensus scheme. In order for the 12 nodes to start validating blocks, they must be granted the *mine*permission:

```
host01:~$ multichain-cli ids grant 1anp8AquvvPi87LP5BQ3HPAKNDRnwKbb99fLTu mine
```

The blockchain will now start mining blocks at fixed intervals. The updated weights of *host02* can then be collected on *host01* as follows:

```
host01:~$ multichain-cli ids liststreamkeyitems anomaly host02_1
{"method":"liststreamkeyitems","params":["anomaly","host02_1"],"id":"85490301-1541081528","chain_name":"ids"}
[
{
"publishers" : [
"1anp8AquvvPi87LP5BQ3HPAKNDRnwKbb99fLTu"
],
"keys" : [
"host02_1"
],
"offchain" : false,
"available" : true,
"data" : {
"json" : {
"deltaweights" : [ 0.1, -0.23, ... ]
}
},
"confirmations" : 52,
"blocktime" : 1541079108,
"txid" : "6631960d2e75dcae7839653979c1d9872d1a01958fb3b93bf76d22a50ffd9298"
}
]
```

The model update has been stored, timestamped, notarized, and confirmed on the blockchain. Each connected node can now not only validate blocks, but also audit the updated weights of each party involved in the federated learning. If a node subscribes to a stream, it will index that stream's contents to enable efficient retrieval.

Each Spring Boot application instance that learns a local model of the autoencoder interacts with an instance of the MultiChain blockchain deployed on the same node. The main reason for this is to publish and retrieve the weight updates for the autoencoder model with minimal network latencies. For the NVIDIA and ODROID boards, the Spring Boot application runs within a Java 8.0 virtual machine for ARM. However, MultiChain is a native application and not publicly available for that target. We therefore re-compiled the MultiChain 2.0 source code on both ARM embedded platforms.

4. Evaluation

In this section, we will first describe our experimental setup with more detail before evaluating our proof-of-concept implementation in different configurations.

4.1. Experimental Setup

Our Linux-based deployment configuration consists of a mix of high-end computing nodes and low-end embedded boards, as depicted in Figure 5:

- 18 desktop nodes with an Intel i5-2400 CPU at 3.10 GHz and 4 GB of memory
- 1 laptop with an Intel i5-3320M CPU at 2.60 GHz and 16 GB of memory
- An NVIDIA Jetson TX2 with a Quad ARM A57 processor, 8 GB of memory and 256 CUDA cores
- An ODROID U3 mini-board with a Quad Cortex-A9 processor and 2 GB of memory

All the above nodes run a 64-bit ARM or x86-64 version of Ubuntu 16.04, except for the laptop, which runs Ubuntu 18.10 and is only used to administer the deployment remotely. The Jetson TX2 with its CUDA cores is specifically suited for deep learning on a resource-constrained device. All the nodes are connected to a wired 1-Gbps network. Depending on the experiment, up to 12 desktop nodes are used to run the Spring Boot application to compute local models through federated learning. Another desktop node acts as a parameter server for blockchain-less experiments, and yet another desktop node is dedicated to centralized learning experiments. All desktop nodes run the MultiChain blockchain to validate new blocks.

For our deep learning experiments, we only use the data captured on Monday in the CICIDS2017 five-day dataset, as that day is confirmed (https://www.unb.ca/cic/datasets/ids-2017.html) to only include benign traffic, whereas the next four days include the various attacks.

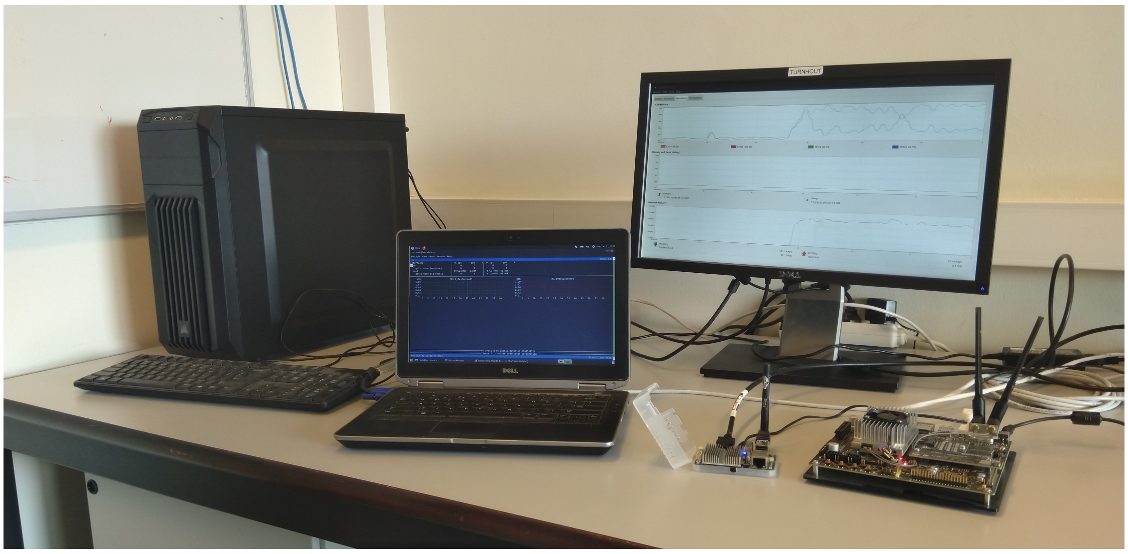

Figure 5. Hybrid deployment of high-end devices and low-end device. From left to right: desktop, laptop, ODROID U3 mini-board, and NVIDIA Jetson TX2 embedded board.

4.2. Centralized Learning

In a first experiment, we learned an autoencoder model with a configuration described in Section 3.1 on a single node, both with a Keras implementation on top of Tensorflow and with our Spring Boot application that uses Deeplearning4j [42]. Figure 6 gives an overview of the training process for 50 epochs, using 20% of the benign samples for validation to avoid overfitting. The duration of training on a single node was 57 min, or a little more than 1 min per epoch on average. The results show an accuracy of around 97% for both the training and validation phase, with a loss value that stabilizes around 5.08 $\times$ 10^{-3}. Table 2 provides an overview of the reconstruction error distribution.

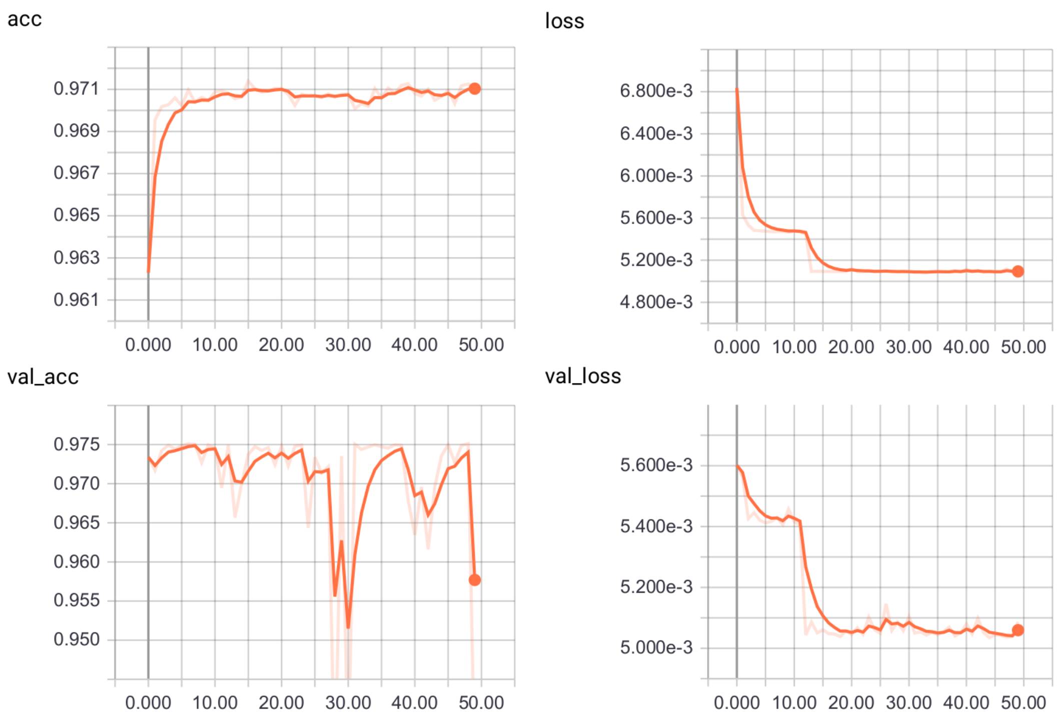

Figure 6. Centralized training of the autoencoder for 50 epochs on benign data only using 20% of the data for validation to avoid overfitting (X-axis depicts epochs, and Y-axis depicts accuracy and loss value on training and validation set).

Table 2. Statistical breakdown of the reconstruction error.

	Reconstruction Error
count	105,984.0
mean	4.988194×10^{-3}
std	9.683384×10^{-3}
min	4.865089×10^{-7}
25%	4.711496×10^{-5}
50%	1.177916×10^{-4}
75%	4.335404×10^{-4}
max	1.791145×10^{-1}

In principle, we use the maximum value in the above table to define a threshold (e.g., 1.8×10^{-1} for the reconstruction error to distinguish benign from anomalous behavior, with the assumption that benign traffic has a lower reconstruction error and the malicious traffic a higher reconstruction error. However, setting this value too high may lead to some malicious traffic going unnoticed, while setting it too low may flag benign traffic as anomalous. Setting a proper value for the threshold depends on whether one wants to minimize the false positives or the false negatives. This optimization concern is application specific and ideally the result of a broader hyperparameter search experiment to find the optimal neural network model and/or thresholds. However, this assessment falls beyond the scope of this work.

4.3. Federated Learning With Varying the Number of Deep Learning Clients

In a federated learning setup, we vary the number of nodes involved in the distributed deep learning process from 2–12 and repeat the training for 50 epochs. We expect the overall training to take less time given that the workload is shared and that the size of the neural network (i.e., the number of weights) is fairly small to cause significant network delays. However, we do anticipate that more epochs are required to reach the same loss value on the training and test sets due to the federated averaging of the weight updates of the individual local models. An overview of the experiment runtimes in minutes and the number of epochs to converge is depicted in Figure 7.

The results appear to confirm our assumptions. Whereas the centralized learning converges after around 20 epochs, this value increases with more nodes. The overall training time for 50 epochs goes down as more nodes contribute to the distributed learning process. However, it is clear that the scalability of our solution is non-linear. We posit out that this outcome is partly due to a naive synchronous implementation of the aggregation at the parameter server, and we anticipate improved runtimes with an asynchronous implementation of the federated model averaging algorithm. A similar approach or method akin to model compression will be essential when the number of local deep learning clients increases significantly.

Another contributing factor to the average epoch time—and hence the model updates in our synchronous implementation—is the heterogeneity in computational resources available on the deep learning clients. If we replace one of the 12 desktop nodes with either the ODROID or the NVIDIA Jetson TX2 embedded boards and their slower ARM processors, our naive synchronous aggregation algorithm must wait for the slowest node. For the Jetson TX2 board, this issue can be mitigated by leveraging the CUDA cores of the board. This accelerates the deep learning by at least a factor of four compared to the ARM processor, as long as the deep learning model fits into the memory of the graphics chip. An asynchronous version of the aggregation model may alleviate the resource heterogeneity problem as well, as long as weight updates against older deep learning models are discarded.

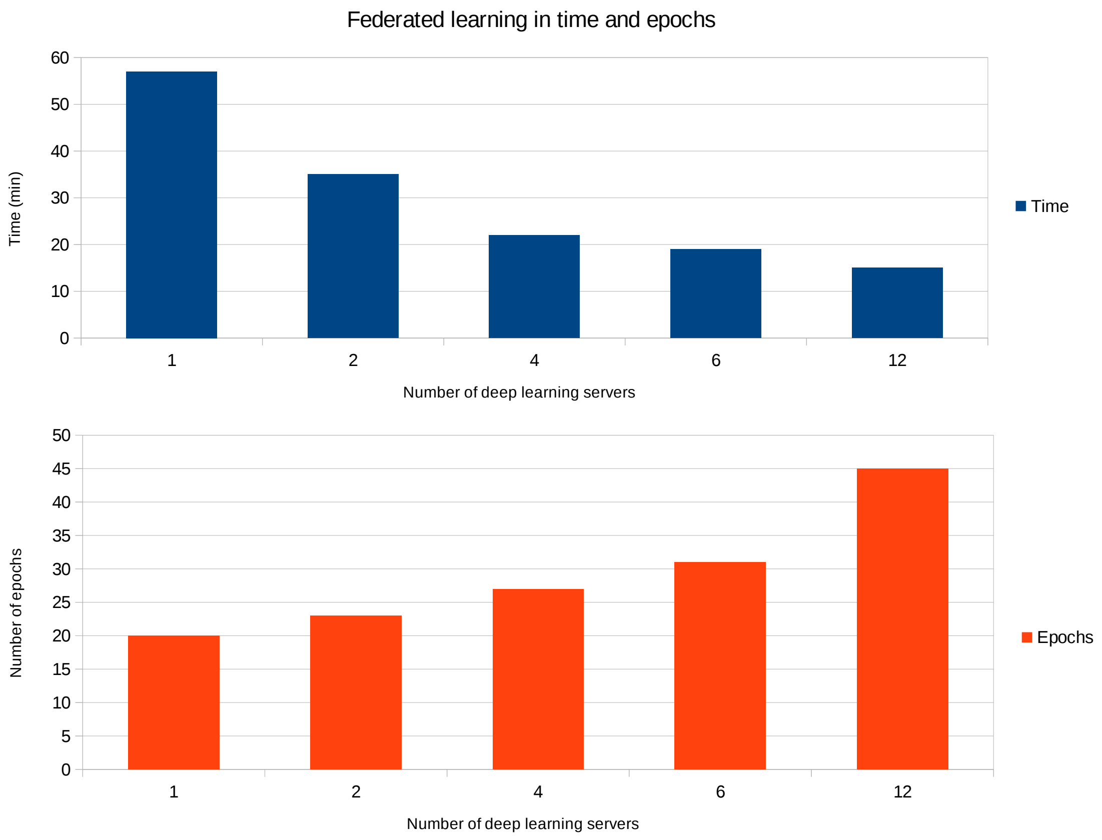

Figure 7. Federated training the autoencoder on benign data only using 20% of the data for testing.

4.4. Federated Learning with Blockchain Support

With the addition of the MultiChain blockchain, both the weight updates and revised models are stored with each epoch on the blockchain. There are now two strategies to leverage the blockchain:

1. A parameter server receives the weight updates directly from the deep learning servers and immediately processes them.
2. A parameter server only processes those weight updates that have been stored and validated on the blockchain.

In both scenarios, the deep learning clients store their weight updates on the blockchain. However, depending on the block generation time (i.e., 15 s by default for MultiChain), the federated model averaging would be delayed a fair amount of time in the second scenario, especially if the parameter server waits for multiple block confirmations. However, the impact of this block generation and confirmation delay is relative to the actual duration of a single epoch, which for a single deep learning client is more than 1 min. Furthermore, we can reconfigure the block confirmation time in MultiChain to as low as 2 s and pause the block generation on a chain if there are no new weight updates as a method to reduce the number of blocks in a chain.

The results of the first scenario are depicted in Figure 8. There is a slight delay caused by the blockchain—with a computational overhead varying between 5% and 15%—due to the fact that each deep learning client now also stores the local weight updates on the blockchain. The parameter server also stores the updated model on the blockchain. With a growing number of nodes, the blockchain needs to be synchronized more frequently. This causes interference in the network, albeit minimal. In the second scenario, the delays are more outspoken due to the block generation and confirmation times, even when waiting for just one block confirmation. For a target block time of 15 s, the time to generate a new autoencoder model update increased up to 41 s. We were able to reduce this to 13 s by lowering the target block time to 5 s. Additionally, it is not possible to change the target block dynamically time at runtime depending on the number of connected deep learning nodes, as this would be a security risk. An adversary could attempt to reduce this time and add blocks in a malicious manner.

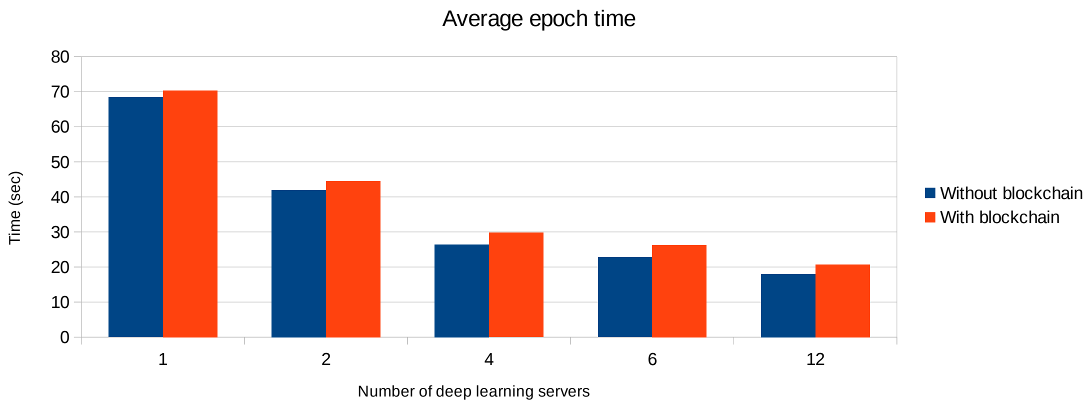

Figure 8. Average epoch time in seconds with and without the blockchain when the deep learning clients directly interact with the parameter server.

To minimize latencies, our proof-of-concept implementation relies on the first strategy, where the parameter server keeps track of unconfirmed weight updates for up to four block generations. In that case, it rolls back to a previous autoencoder model stored on the blockchain and discards any unconfirmed weight updates. In an online learning scenario with infrequent data updates, the second strategy is more appropriate.

4.5. Auditing the Federated Learning Process

Note that the blockchain can only guarantee the immutability and verify the integrity of the data stored on the blockchain. It cannot assure the veracity of the data. Any party that is granted the permission to write to a stream can store any data and launch a poisoning attack on the federated deep learning process. Additional functionality is required to monitor and audit the content of the blockchain, both for (1) the local weight updates by the deep learning servers and (2) the averaged model updates by the parameter server.

If the federated averaging algorithm is known, each client connected to the MultiChain blockchain with a receive permission can verify the computation of the model updates by the parameter server. Ideally, this functionality is offered by means of a smart contract, but the capability of running code directly on the blockchain—and not by relying on external services—is not supported by MultiChain (contrary to blockchain solutions like Ethereum and Hyperledger Fabric).

A similar observation can be made for the weight updates by the local deep learning servers. However, the computational complexity of the stochastic gradient descent algorithm makes execution as a smart contract impractical. As a result, external monitoring for poisoning attacks [38–40] needs to be put in place to audit the veracity of the data stored on the blockchain, possibly in an offline manner. We implemented an additional countermeasure for making poisoning attacks more challenging by:

- Creating two streams, one for local weight updates and another one for averaged model updates.
- Granting deep learning clients to write only to the first stream and read only from the second.

If a deep learning client were compromised, an adversary could not learn from the weight updates of the other deep learning clients to launch a carefully- crafted poisoning attack that will also go undetected by the local models of the other deep learning clients.

4.6. Discussion, Limitations, and Validity Threats

In this work, we proposed a solution that integrates blockchain technology with federated learning to help audit the model updates of the federated deep learning clients. Our results demonstrated a limited performance overhead imposed by the MultiChain blockchain in a specific security use case of federated intrusion detection. Generalizing these results to other applications is not trivial. However, certain hypotheses can be made about the general tendencies of our approach. We believe that our integration approach is especially beneficial for security use cases where inference is performed directly on raw data (rather than on the preprocessed data, such as the CICIDS2017 dataset). In this case, the relative performance overhead of the blockchain will reduce even further, as federated learning will be exempt of the costly transmission of vast amounts of raw data. At the same time, analogous use cases will presumably require using deeper neural networks with more sophisticated architectures, necessary for automatic feature extraction. Deeper models in turn will have their own not easily estimated impact on performance overhead. To begin with, they will require relatively increased local processing in order to compute the model updates at each client in the federation, which may affect the latency of global updates. Secondly, a higher amount of learnable parameters in a deeper architecture will require storing more data on the blockchain. In order to confirm these assumptions, numerous federated learning deployments have to be systematically validated in different application scenarios that vary in the nature of data, processing, and network capacity.

Another consequence of the proposed integration approach is a trade-off between the local rate of convergence on the clients and the audit frequency of the global model updates. On the one hand, iterating more frequently at each client locally before computing the federated average allows converging faster at the nodes. On the other hand, having fewer local iterations causes more federated averages to be computed and stored on the distributed ledger. More frequently-chained model updates on the blockchain improve the auditability of the machine learning model. However, as most blockchains create new blocks at fixed time intervals, we propose to align the federated averaging algorithm with the block creation period to minimize any latencies.

In this work, we used an intrusion detection security use case in order to demonstrate the feasibility of the introduced approach to the integration of federated learning and blockchain technology. The type of neural network we used in our experiments, i.e., the autoencoder, has a fairly simple architecture, while more advanced neural networks have been proposed in the literature for anomaly and network intrusion detection. Our approach remains equally applicable and scalable to deeper or more complex neural networks, such as convolutional neural networks, as well as to other use cases that could benefit from federated learning. However, the federated averaging algorithm has to be tuned accordingly for each particular application in order to get the best results, which may indirectly affect the interaction with the MultiChain blockchain.

5. Conclusions

In this work, we explored federated deep learning as a use case for permissioned blockchains using the latter to audit the distributed learning process. Our federated learning experiments using (a) an autoencoder for anomaly detection, (b) a real-world intrusion detection dataset (i.e., CICIDS2017), and (c) MultiChain as the underlying blockchain technology illustrated the practical feasibility to create an immutable audit trail. We obtained a limited relative performance overhead caused by the blockchain of about 5–15%, while providing full transparency over the distributed learning process.

However, there are some validity threats that need to be taken into consideration when aiming to generalize the obtained results. First of all, the performance overhead caused by the blockchain is closely tied to the complexity of a (deep) neural network at the core of the federated learning algorithm. Since the adopted model averaging algorithm relies on iterative sharing and averaging of weights that need to be stored on a blockchain for audit, the total amount of weights of the neural network will define the performance impact of the proposed approach. In our work, an autoencoder with three hidden layers is leveraged as an anomaly detection method, with a total amount of 3000 weights.

A deeper autoencoder or another deep learning model, such as CNN networks with more hidden layers, would require storing many more learnable parameters for auditable model averaging. However, these deep models normally operate on raw input data, and the relative computational overhead of federated learning on a blockchain in comparison to a centralized approach will still be limited. Secondly, in our experiments with a growing number of desktop nodes, we ensured that each deep learning client had the same amount of traffic traces to process in order to compute each local deep learning model. In practice—and as shown in Figure 3 on the right—the amount of data to be processed may be unbalanced. An unequal availability of computational resources and network bandwidth capacity may skew the results, especially for significantly larger deployments. Finally, we assume that the blockchain itself is not compromised. MultiChain is tolerant to a misbehaving minority of nodes, but not to a global network-level attack. Furthermore, our experimental setup relied on a single administrator that managed the governance of the blockchain. The assumption is that this central administrator can be trusted and, e.g., does not grant extra permissions to colluding adversaries.

As future work, we aim to improve the model aggregation algorithm to cope not only with heterogeneous hardware, but also with unreliable network connectivity and intermittently-connected nodes. A transition to the Hyperledger Fabric blockchain may allow for some of the auditing functionality to be executed as smart contracts, so that poisoning attacks can be mitigated before the weight updates are stored on the blockchain.

Author Contributions: Conceptualization, D.P., V.R., I.T., J.S., W.J., and E.I.-Z.; data curation, V.R.; funding acquisition, W.J. and E.I.-Z.; project administration, D.P. and W.J.; software, D.P.; supervision, D.P. and W.J.; writing, original draft, D.P.; writing, review and editing, V.R., I.T., and J.S.

Funding: This research is partially funded by the Research Fund KU Leuven. Work for this paper was supported by the European Commission through the H2020 project EXCELL (http://excell-project.eu/) under Grant No. 691829. In addition, it was also funded by imecthrough ICON RADIANCE(HBC.2017.0629). RADIANCE is a project realized in collaboration with imec. Project partners are BARCO, Skyhaus, and Skyline communications, with project support from VLAIO(Flanders Innovation and Entrepreneurship).

Acknowledgments: We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sadeghi, A.R.; Wachsmann, C.; Waidner, M. Security and Privacy Challenges in Industrial Internet of Things. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; ACM: New York, NY, USA, 2015; pp. 54:1–54:6. [CrossRef]
2. Preuveneers, D.; Zudor, E.I. The intelligent industry of the future: A survey on emerging trends, research challenges and opportunities in Industry 4.0. *JAISE* **2017**, *9*, 287–298. [CrossRef]
3. Verma, R.M.; Kantarcioglu, M.; Marchette, D.J.; Leiss, E.L.; Solorio, T. Security Analytics: Essential Data Analytics Knowledge for Cybersecurity Professionals and Students. *IEEE Secur. Privacy* **2015**, *13*, 60–65. [CrossRef]
4. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly Detection: A Survey. *Acm Comput. Surv.* **2009**, *41*, 15:1–15:58. [CrossRef]
5. Verma, R. Security Analytics: Adapting Data Science for Security Challenges. In Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics, Tempe, AZ, USA, 19–21 March 2018; ACM: New York, NY, USA, 2018; pp. 40–41. [CrossRef]
6. Konecný, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, D. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv* **2016**, arXiv:1610.05492
7. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; Singh, A., Zhu, X.J., Eds.; Volume 54, pp. 1273–1282.

8. Du, M.; Li, F.; Zheng, G.; Srikumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; ACM: New York, NY, USA, 2017; pp. 1285–1298. [CrossRef]
9. Hitaj, B.; Ateniese, G.; Perez-Cruz, F. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; ACM: New York, NY, USA, 2017; pp. 603–618. [CrossRef]
10. Sundaram, A. An Introduction to Intrusion Detection. *XRDS* **1996**, *2*, 3–7. [CrossRef]
11. Inayat, Z.; Gani, A.; Anuar, N.B.; Khan, M.K.; Anwar, S. Intrusion response systems: Foundations, design, and challenges. *J. Netw. Comput. Appl.* **2016**, 53–74. [CrossRef]
12. Karim, I.; Vien, Q.T.; Le, T.A.; Mapp, G. A Comparative Experimental Design and Performance Analysis of Snort-Based Intrusion Detection System in Practical Computer Networks. *Computers* **2017**, *1*, 6. [CrossRef]
13. Lunt, T.F. A Survey of Intrusion Detection Techniques. *Comput. Secur.* **1993**, *12*, 405–418. [CrossRef]
14. Mitchell, R.; Chen, I.R. A Survey of Intrusion Detection Techniques for Cyber-physical Systems. *ACM Comput. Surv.* **2014**, *46*, 55:1–55:29. [CrossRef]
15. Vasilomanolakis, E.; Karuppayah, S.; Mühlhäuser, M.; Fischer, M. Taxonomy and Survey of Collaborative Intrusion Detection. *ACM Comput. Surv.* **2015**, *47*, 55:1–55:33. [CrossRef]
16. Zarpelo, B.B.; Miani, R.S.; Kawakani, C.T.; de Alvarenga, S.C. A Survey of Intrusion Detection in Internet of Things. *J. Netw. Comput. Appl.* **2017**, *84*, 25–37. [CrossRef]
17. Rubinstein, B.I.; Nelson, B.; Huang, L.; Joseph, A.D.; Lau, S.h.; Rao, S.; Taft, N.; Tygar, J.D. ANTIDOTE: Understanding and Defending Against Poisoning of Anomaly Detectors. In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, Chicago, IL, USA, 4–6 November 2009; ACM: New York, NY, USA, 2009; pp. 1–14. [CrossRef]
18. Biggio, B.; Nelson, B.; Laskov, P. Poisoning Attacks Against Support Vector Machines. In Proceedings of the 29th International Coference on International Conference on Machine Learning, Edinburgh, UK, 26 June–1 July 2012; pp. 1467–1474.
19. Chen, S.; Xue, M.; Fan, L.; Hao, S.; Xu, L.; Zhu, H.; Li, B. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Comput. Secur.* **2018**, *73*, 326–344. [CrossRef]
20. Wang, Z. Deep Learning-Based Intrusion Detection With Adversaries. *IEEE Access* **2018**, *6*, 38367–38384. [CrossRef]
21. Huang, C.H.; Lee, T.H.; Chang, L.h.; Lin, J.R.; Horng, G. Adversarial Attacks on SDN-Based Deep Learning IDS System. In *Mobile and Wireless Technology 2018*; Kim, K.J., Kim, H., Eds.; Springer: Singapore, 2019; pp. 181–191.
22. Madan, I.; Saluja, S.; Zhao, A. Automated Bitcoin Trading via Machine Learning Algorithms. 2014. Available online: http://cs229.stanford.edu/proj2014/Isaac%20Madan,%20Shaurya%20Saluja,%20Aojia%20Zhao,Automated%20Bitcoin%20Trading%20via%20Machine%20Learning%20Algorithms.pdf (accessed on 22 November 2018).
23. Bikowski, K. Application of Machine Learning Algorithms for Bitcoin Automated Trading. In *Machine Intelligence and Big Data in Industry*; Springer: Cham, Switzerland, 2016; pp. 161–168.
24. Chen, W.; Zheng, Z.; Cui, J.; Ngai, E.; Zheng, P.; Zhou, Y. Detecting Ponzi Schemes on Ethereum: Towards Healthier Blockchain Technology. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 1409–1418. [CrossRef]
25. Bogner, A. Seeing is Understanding: Anomaly Detection in Blockchains with Visualized Features. In Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers, Maui, HI, USA, 11–15 September 2017; ACM: New York, NY, USA, 2017; pp. 5–8. [CrossRef]
26. Meng, W.; Tischhauser, E.W.; Wang, Q.; Wang, Y.; Han, J. When Intrusion Detection Meets Blockchain Technology: A Review. *IEEE Access* **2018**, *6*, 10179–10188. [CrossRef]
27. Weng, J.; Weng, J.; Zhang, J.; Li, M.; Zhang, Y.; Luo, W. DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-Based Incentive. Cryptology ePrint Archive, Report 2018/679. 2018. Available online: https://www.semanticscholar.org/paper/DeepChain%3A-Auditable-and-Privacy-Preserving-Deep-Weng-Weng/07f229cc6e5b80fc8ffbbb3e4db85142466f55a9 (accessed on 16 December 2018).

28. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy, Funchal, Portugal, 22–24 January, 2018; pp. 108–116. [CrossRef]
29. Chalapathy, R.; Menon, A.K.; Chawla, S. Anomaly Detection using One-Class Neural Networks. *arXiv* **2018**, arXiv:1802.06360
30. Ruff, L.; Vandermeulen, R.; Goernitz, N.; Deecke, L.; Siddiqui, S.A.; Binder, A.; Müller, E.; Kloft, M. Deep One-Class Classification. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Dy, J., Krause, A., Eds.; PMLR: Stockholmsmässan, Stockholm, Sweden, 2018; Volume 80, pp. 4393–4402.
31. Ayesh, A.S.D.D.A. Intelligent intrusion detection systems using artificial neural networks. *ICT Express* **2018**, *4*, 95–99. [CrossRef]
32. Ryan, J.; Lin, M.J.; Miikkulainen, R. Intrusion Detection with Neural Networks. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*; MIT Press: Cambridge, MA, USA, 1998; pp. 943–949.
33. Hodo, E.; Bellekens, X.; Hamilton, A.; Dubouilh, P.; Iorkyase, E.; Tachtatzis, C.; Atkinson, R. Threat analysis of IoT networks using artificial neural network intrusion detection system. In Proceedings of the 2016 International Symposium on Networks, Computers and Communications (ISNCC), Hammamet, Tunisia, 11–13 May 2016; pp. 1–6. [CrossRef]
34. Baldi, P. Autoencoders, Unsupervised Learning and Deep Architectures. In Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop, Washington, DC, USA, 2 July 2011; Volume 27, pp. 37–50.
35. Min, E.; Long, J.; Liu, Q.; Cui, J.; Cai, Z.; Ma, J. SU-IDS: A Semi-supervised and Unsupervised Framework for Network Intrusion Detection. In *Cloud Computing and Security*; Sun, X., Pan, Z., Bertino, E., Eds.; Springer: Cham, Switzerland, 2018; pp. 322–334.
36. Radford, B.J.; Richardson, B.D.; Davis, S.E. Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic. *arXiv* **2018**, arXiv:1805.03735
37. McMahan, H.B.; Moore, E.; Ramage, D.; y Arcas, B.A. Federated Learning of Deep Networks using Model Averaging. *arXiv* **2018**, arXiv:1602.05629.
38. Jagielski, M.; Oprea, A.; Biggio, B.; Liu, C.; Nita-Rotaru, C.; Li, B. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–23 May 2018; pp. 19–35. [CrossRef]
39. Wang, Y.; Chaudhuri, K. Data Poisoning Attacks against Online Learning. *arXiv* **2018**, arXiv:1808.08994
40. Steinhardt, J.; Koh, P.W.; Liang, P. Certified Defenses for Data Poisoning Attacks. *arXiv* **2017**, arXiv:1706.0369.
41. Greenspan, G. Multichain Private Blockchain—White Paper. 2017. Available online: https://arxiv.org/pdf/1706.03691.pdf (accessed on 16 December 2018).
42. Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, 2014. Available online: https://deeplearning4j.org (accessed on 22 November 2018).