Federated Machine Learning in Network Intrusion Detection

Bram Van de Walle

Supervisor(s): Laurens D'hooge, Prof. dr. Bruno Volckaert, Dr. ir. Tim Wauters

Abstract— **This thesis has conducted research to the use of federated learning in network intrusion detection. Network intrusion detection systems monitor the network traffic and try to detect attacks if they occur. Such intrusion detection systems (IDSs) can use machine learning models that classify network traffic flows captured by the IDSs as benign or malicious. Such machine learning models require datasets to be trained with. Organizational networks that consist of two (or more) geographically separated networks, in which each network has its own IDS that uses machine learning models, must have local training datasets for each of the IDSs. If, over time, one of the IDSs learns to recognize a new attack type, it would be useful that the other IDSs could learn from this IDS. A simple solution is to share the new training dataset with all IDSs. However, this may not always be feasible due to privacy/security policies, because sending the training sets requires too much network bandwidth, or because it is computationally too expensive to retrain all models in the IDSs. This master's thesis found that federated learning is a possible solution that allows for multiple IDSs to learn from each other during training. Seven different experiments give insights in why and how using federated learning in network intrusion detection systems can be useful. In these experiments, the CIDDS-001 datasets are used that provide captured network traffic flows of both benign and malicious flows. The framework that is used for performing the federated learning experiments is TensorFlow Federated.**

Keywords— **Federated Learning, Machine Learning, Random Forest Classifier, Multi-Layer Perceptron, Network Intrusion Detection, TensorFlow Federated, CIDDS-001**

I. Introduction

Almost all organizations today have information technology (IT) as a vital part of their business model. Since companies may store sensitive information in their IT infrastructure, IT has become a coveted target for hackers. As been proven multiple times by the damage that can be done by attackers hacking their way into corporate IT infrastructure, companies must try to limit the chances of hackers being able to break into the companies' IT infrastructure. Possible means of doing so are by imposing user authentication before employees can gain access to IT infrastructure, firewall rules to impose network policies or installing anti-virus software on the computers of the employees. However, none of these solutions are bulletproof because of the risk of human error. Hackers can use social engineering approaches to gain information about employees or even obtain user credentials. Even more worrying is that the flaws that attackers exploit, may not even come from a mistake made by any of the employees of the company itself. There could be flaws in any of the software that the company uses. Even software that one might think of as being secure and reliable, can contain flaws. Recently, on two separate occasions, hackers were able to use flaws in widely used software and breach into tens of thousands of governmental and private computer networks. The first started in March 2020 when Russian hackers were able to compromise an update of SolarWinds' network management software Orion and install malware in organizations and governing bodies that were updating their SolarWinds Orion software. The scope of this breach was about 18,000 networks [1], [2]. More recently, another very serious attack occurred. Early March 2021, Chinese hackers exploited vulnerabilities in Microsoft Exchange and were able to hack into hundreds of thousands of organizations worldwide [3].

Hence, it is important that companies invest in trying to avoid letting hackers break into their IT infrastructure. However, since human errors cannot be avoided at all times, prevention alone does not suffice. What if an attacker did manage to break into the infrastructure? Will this ever be discovered? If not, nobody will ever be able to handle the breach. To make sure that an intrusion can be dealt with, it must be detected. To lower the damage done and thereby the costs of the attack, the detection must happen quickly. This is where Intrusion Detection Systems come in handy.

Intrusion Detection Systems (IDS) are hardware or software applications that deal with the task of monitoring systems or networks with the purpose of detecting malicious activities [4], [5]. Once malicious activity has been detected, the IDS must generate alarms/notifications so that the intrusion can be dealt with appropriately. Either by a human operator or by an automated response. There are different types of IDSs: they can be host-based or network-based, signature-based or anomaly-based, stateless or stateful, centralized or distributed [6]. In any case, the goal of an IDS is to detect attacks as soon as possible in order to limit the damage done to the IT infrastructure of the company and to the company itself.

II. Goals of the thesis

In this thesis, research is conducted in the field of network-based/anomaly-based intrusion detection systems where the IDS uses a machine learning model to classify monitored traffic as benign or malicious. These machine learning models must be trained in order to be able to correctly classify flows of network traffic. Therefore, an IDS must contain a local training dataset on which its machine learning model can be trained. Such a training set contains flows that are similar to what the IDS will monitor when used in real life scenarios. The starting point of this thesis is an organization that has two geographically separated networks. In each of the two networks, an IDS is placed that monitors the network traffic right before it enters the Local Area Network (LAN). An example of such a network setup is depicted in Figure 1.

As mentioned, each IDS has its own local training dataset. The training dataset can, for example, consist of labeled net-

B. Van de Walle, student Master of Science in Information Engineering Technology, Ghent University (UGent), Gent, Belgium. E-mail: Bram.VandeWalle@UGent.be .

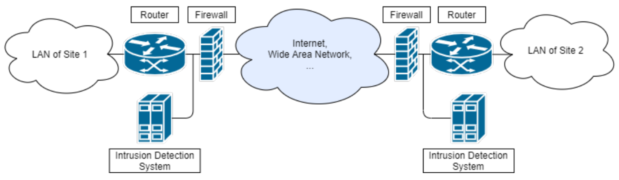
Router
Firewall
Firewall
Router
LAN of Site 1
Internet, Wide Area Network, ...
LAN of Site 2
Intrusion Detection System
Intrusion Detection System

Fig. 1. Example network setup of organization with two, geographically separated, sites

work flows that the IDS has captured over time. How the creation of such a local training set can be done, is out of the scope of this thesis. However, it is possible that the local dataset of one IDS contains flows of an attack type that the other IDS does not have and vice versa. The consequence of this is that one IDS can detect certain attacks that the other cannot because the latter will not learn to recognize the attack type during the training phase. A straightforward solution is to exchange the local datasets between the two IDSs. However, this may not always be possible since the local training datasets contain captured network flows that possibly contain privacy sensitive data and thus certain policies (e.g. GDPR or corporate privacy/security policies) may prevent the IDSs to share their local datasets. Another option is to exchange the internal parameters of the machine learning models that are used in the IDSs that classify the monitored network traffic. If the same model is used in both IDSs and during training, both models exchange and aggregate their internal parameters, maybe the two models can learn from each other. The concept of exchanging the internal parameters and aggregating them is called federated learning. The goal of this thesis is to find an answer to the following question: “Can two network intrusion detection systems that each have their own local training datasets learn from each other when one training dataset lacks captured flows from a certain attack type while the other training dataset does not?”

In order to be able to answer the aforementioned question, seven experiments are executed where each experiment is based on the one before or is introduced because the results of the previous experiment led to new questions. Before the experiments are discussed, section III gives an overview of the related work and section IV gives some required knowledge to be able to follow the discussion of the experiments. Finally, in section V the experiments, the results and the conclusions are discussed.

III. Related Work

A. Coburg Intrusion Detection Data Sets

As mentioned in the introduction, network intrusion detection systems that internally use machine learning models to classify the network traffic flows they monitor, require a training dataset to be trained with. Throughout the experiments in this thesis, the datasets that represent the local training datasets of the two IDSs depicted in Figure 1 will come from the Coburg Intrusion Detection Data Sets (CIDDS). More specifically, the CIDDS-001 datasets will be used. The researchers of the CIDDS-001 project emulated the network of a small business environment with the open source cloud computing infrastructure OpenStack. They used Python scripts in combination with Linux tools like `nmap` to generate benign and malicious traffic [7]. Inside the emulated network, the network traffic flows are monitored at two places: at an internal router and at an external server. Capturing these flows is done with Cisco’s network monitoring protocol NetFlow [8]. The researchers used 11 different NetFlow attributes which are stored in the CIDDS-001 datasets: *duration of the flow*, *start timestamp of the flow*, *number of transmitted bytes*, etc. Next to these 11 features, each flow is labeled with 4 attributes: *Class*, *AttackType*, *AttackID* and *AttackDescription*. The *class* indicates whether a flow is “normal”, “attacker”, “victim”, “suspicious” or “unknown”. The *attack type* indicates which attack was executed: “bruteForce”, “dos”, “pingScan”, “portScan” or “—” for normal (i.e. benign traffic) flows. The label *attack id* assigns a unique ID to each attack and each flow of that attack gets assigned this attack id. Finally, the *attack description* provides additional information about the executed attack corresponding to the flow. Throughout a period of four weeks, the network traffic was generated and captured [9].

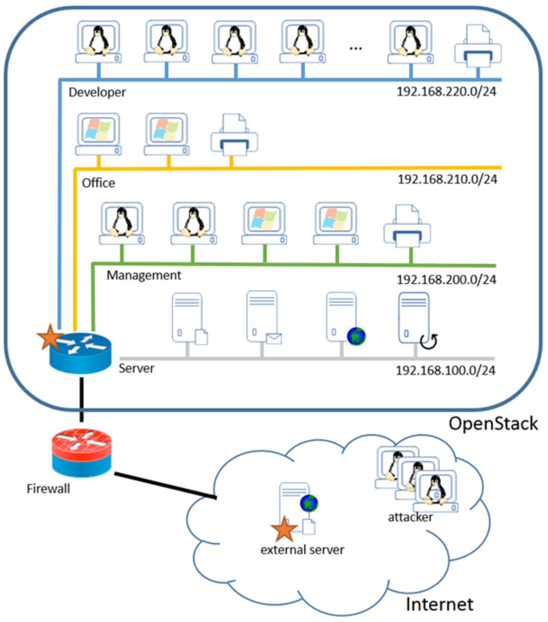

Fig. 2. Emulated small business network environment (*source:* [7])

B. Evaluation of existing machine learning based network intrusion detection algorithms

In this thesis, anomaly-based network intrusion detection systems are considered that use machine learning models to classify the network traffic flows it monitors. In [10], Abhishek Verma and Virender Ranga have evaluated multiple existing machine learning models using the CIDDS-001 datasets. Both supervised (i.e. the k-Nearest Neighbors (KNN), Support Vector Machine (SVM), decision tree (DT), random forrest classifier (RFC), artificial neural network (ANN), Naive Bayes (NB) and deep learning (DL) models) and unsupervised (i.e k-means clustering (KMC), expectation maximization clustering (EMC) and self-organizing maps (SOM) models) machine learning algorithms are evaluated. Most of the algorithms that are learned in a supervised manner score very good in the evaluation. KNN, SVM, DT, RFC and DL all have an accuracy of more than 94% on the CIDDS-001 data captured at the external server, and more than 99% accuracy on the CIDDS-001 data captured internally in the OpenStack network. The unsupervised machine learning models KMC, EMC and SOM have a much lower score. All scores are below 50%, with the exception of KMC for the flows captured at the internal router (where the accuracy is 99.6%). Therefore, only supervised machine learning algorithms will be used in this thesis.

C. Federated Learning

In 2017, Google proposed a concept called *federated learning* in [11] as an answer to problems with machine learning models that have to learn on huge datasets and/or datasets containing privacy sensitive data. A lot of people have smartphones these days which contain useful information that can be used to enhance the user experience. For example, Google could collect text written by people to enhance their text entry models. Traditionally, such models were trained on centralized data, but text written by smartphone users should not be centralized for privacy reasons. An alternative is to let the smartphones learn a model on their own local data, centralize the locally computed updates to a central server which in turn aggregates these locally computed updates into a global model. This global model is then redistributed among the participating smartphones. The process of locally computing updates, centralizing and then updating the updates can be done repeatedly.

In [12], S. Rahman et al. further explore federated learning by comparing it to an on-device learning approach called *SecProb* and the centralized approach. The centralized approach is considered to be the best case scenario because no compromises are made with respect to training a model: all local training data are available for training the model. S. Rahman et al. found that, after enough training rounds, their federated learning model outperforms the SecProbe model and approaches the accuracy reached by the centralized model.

IV. Experimental Prerequisites

This section discusses how the CIDDS-001 datasets are used in the experiments and how they are preprocessed. Also, the algorithmic choices are explained, while the next section discusses the actual experiments and the conclusions thereof.

A. Content and use of the CIDDS-001 datasets

It was already mentioned that two IDSs are considered for the experiments. Each of the IDSs will be a federated learning client and thus requires a local training dataset. The traffic that is captured inside the CIDDS-001 datasets was captured over a time span of four weeks. For each week, a dataset is provided for both the internal captured flows and the external captured flows, totalling eight datasets. However, only the datasets of the first two weeks with flows captured internally contain flows from all attack types “port scan”, “ping scan”, “DoS” and “brute force”. Therefore, these two datasets, *internal week 1* and *internal week 2*, will be used in the experiments where each dataset represents the local dataset available for each IDS shown in Figure 1.

The goal of classification models is that given a set of features (also known as the feature vector), the model can predict the correct class/label corresponding to the feature vector. With the CIDDS-001 datasets, the feature vector consists of the captured NetFlow attributes and the class/label that a model has to predict is one of the label attributes “Class”, “AttackType”, “AttackID” or “AttackDescription”. The last two label attributes are merely metadata about the attacks that will never be available in real life scenarios. In some papers (e.g. [10]) the classification label used is the attribute “Class”. Although it can be useful to know whether a network flow is either “normal”, “attacker”, “victim”, “suspicious” or “unknown”, if a model can predict the specific attack type in case of malicious flows, the IDS can be more specific in the alerts it generates or can take specific actions to prevent further damage. Therefore, in the experiments in this thesis, the attribute that is chosen to be the classification label, is the attribute “AttackType”.

B. Preprocessing the CIDDS-001 datasets

Before the CIDDS-001 datasets *internal week 1* and *internal week 2* can be used in the experiments, some preprocessing steps need to executed. The first four preprocessing steps are executed in all experiments while the fifth preprocessing step is executed depending on which model is used in the experiments.

B.1 Remove abnormalities

There are two abnormalities in the CIDDS-001 datasets. The first abnormality is the appearance of an extra column *Flows* inside the datasets for which no documentation is provided in [7] or [9]. The only value in this column is a “1”, making this attribute a useless feature for the classification models. Therefore, this column is dropped. The second abnormality is found in the column *Bytes* (which specifies the number of bytes transmitted in a flow). In flows for which the number of transmitted bytes exceeds one million, the value is encoded with an “M” to denote the million. For example, if the number of transmitted bytes is 6,000,000 the value for the attribute *Bytes* will be “6 M”. This encoding is reversed so that all values in the column *Bytes* are integers.

B.2 One-hot encode categorical attributes

Next, the categorical attributes in the CIDDS-001 datasets are one-hot encoded. That means that each of the possible values in a categorical attribute, becomes a binary column. The value of the resulting column indicates whether the value corresponding to the column applies or not. The only categorical attribute in the CIDDS-001 dataset is the attribute *Proto* which indicates the protocol used in the captured flow: “ICMP”, “IGMP”, “TCP”, or “UDP”.

B.3 Unfold OR-concatenated attributes

The following step in the preprocessing phase is unfolding the OR-concatenated values in the column *Flags*. If the protocol of a flow is “TCP”, the TCP flags URG, ACK, PSH, RST, SYN and FIN of the flow are captured in the column *Flags* using a six-character string. Each dot in the string `"......"` is replaced with “U”, “A”, “P”, “R”, “S” and/or “F” respectively, depending on whether or not the corresponding TCP flag was set. If the protocol of the flow is not “TCP”, the string `"......"` is used as is. In this preprocessing step, this column is unfolded into six separate binary columns indicating the use of the TCP flag.

B.4 Drop some columns for classification

Some of the attributes that are available in the CIDDS-001 datasets have to be removed prior to training because they are highly correlated with the target, yet contribute no true improvement to model generalization. Most often these features fall under sample metadata. The columns that are dropped in this preprocessing step are: *Date first seen*, *Src IP Addr*, *Dst IP Addr*, *Class*, *AttackID* and *AttackDescrption*.

B.5 Normalize non-binary columns

The final step in the preprocessing phase is normalizing the non-binary columns. However, normalization is not always required. Classification algorithms that use decision trees (such as the random forest classifier) do not require the data to be normalized because each node in a decision tree partitions the data in two groups based on one feature. Whether the data is normalized or not, has no influence on how the data is partitioned. The parameter that splits the data would be scaled just as the data is scaled during normalization. Classification algorithms that execute calculations using multiple features at once, such as the multi-layer perceptron (MLP) neural network, do require the data to be normalized since using multiple features at once may result in certain features to have a bigger weight on the classification model than other features. Thus, in experiments where the MLP model is used, the non-binary features are normalized with either min-max normalization or z-score normalization.

C. Classification models

Throughout the experiments, two different machine learning models are used as classifiers. The first two experiments start using the random forest classifier (RFC) since this model showed great results on the CIDDS-001 data in [10] as discussed in III. However, in later experiments, TensorFlow Federated's implementation of federated learning is used to execute the federated learning experiments and TensorFlow Federated does not support the use of RFC models. Therefore, the multi-layer perceptron neural network as proposed by [13] is evaluated in experiments 3 and 4. As the results of the MLP model approaches the results of RFC, it is further used in the federated learning experiments 5, 6 and 7.

V. Experiments & Conclusions

The goal of this thesis was to find an answer to the question: "Can two network intrusion detection systems learn from each other using federated learning if the network flow classification models behind each of the IDSs have a local training dataset in which flows of one attack type are not available while the other local training dataset does have flows of that attack type available?" The answer to this question will be discussed throughout this section.

Experiment 1 is a simple classification problem using RFC models on the CIDDS-001 datasets *internal week 1* and *internal week 2*. The datasets are preprocessed as discussed in IV-B. Afterwards they are balanced out. In *internal week 1*, 1,626 flows are randomly selected from each attack type. In *internal week 2*, 2,731 flows are randomly selected from each attack type. These two datasets are split into two training and testing datasets. The first RFC model *rfc_week1* is trained with the training dataset coming from *internal week 1*. Analogously for *rfc_week2* and *internal week 2*. Then each RFC model is tested twice. First with the test sets and a second time with the full dataset (i.e. the datasets before *internal week 1* and *internal week 2* are balanced out) coming from "the other week". The results shown in Tables I and II show that the RFC models are very suitable to classify the CIDDS-001 network flows with respect to the classification label "attack type".

TABLE I
PRECISION SCORES OF TRAINED MODEL "RFC_WEEK1" IN EXPERIMENT 1 (ALL VALUES IN PERCENTAGES)

Attack type	Results of **rfc_week1**	
	Testing set week 1	Full dataset week 2
Normal	99.40	99.72
BruteForce	94.21	86.04
DoS	100.0	99.96
PingScan	92.83	90.85
PortScan	96.94	92.60

TABLE II
PRECISION SCORES OF TRAINED MODEL "RFC_WEEK2" IN EXPERIMENT 1 (ALL VALUES IN PERCENTAGES)

Attack type	Results of **rfc_week2**	
	Testing set week 2	Full dataset week 1
Normal	98.75	98.93
BruteForce	100.0	96.37
DoS	100.0	99.98
PingScan	91.07	96.58
PortScan	91.29	95.60

The first experiment is followed by the so-called "base experiment". This experiment is very reminiscent of experiment 1. However, all "brute force" network flows are removed from the training dataset of *rfc_week1* and 3,359 flows of the other attack types are randomly chosen. Analogously, all "ping scan" network flows are removed from the training dataset of *rfc_week2* and 3,366 flows of the other attack types are randomly chosen. The trained models are tested in a similar way as seen in experiment 1. The results can be found in Tables III and IV. Without surprise, once the models are trained and tested with a test set that contains flows of all attack types, the models cannot correctly predict the flows of which the attack type was not available in their training dataset.

The base experiment serves as a motivation for why federated learning may be useful in the context of network intrusion detection. The base experiment shows that if an organization has two geographically separated networks where each network has its own IDS in which the classification model is trained with a local training dataset, it is possible that one IDS can detect attacks that the other cannot and vice versa. This is unfortunate since the organization would benefit from both IDSs being able to detect all attacks. Sharing the local training dataset could be the solution, but may be disallowed due to certain policies (pri-

TABLE III

Precision scores of trained model "rfc_week1" in Experiment 2 (all values in percentages)

Attack type	Results of **rfc_week1**	
	Testing set week 1	Full dataset week 2
Normal	100.0	99.90
BruteForce	n/a	0.000
DoS	99.96	99.92
PingScan	97.02	90.92
PortScan	95.73	92.63

TABLE IV

Precision scores of trained model "rfc_week2" in Experiment 2 (all values in percentages)

Attack type	Results of **rfc_week2**	
	Testing set week 2	Full dataset week 1
Normal	98.53	98.89
BruteForce	99.89	96.25
DoS	100.0	99.97
PingScan	n/a	0.000
PortScan	95.83	98.22

vacy/security policies) since the training datasets may contain sensitive data, as the training datasets consist of captured network traffic. On top of that, sending the training datasets over the network and having to retrain the models can be too costly. However, the parameters of the classification models of each client can be exchanged and may be aggregated using federated averaging.

For the federated learning experiments 5, 6 and 7 (cfr. infra), the framework TensorFlow Federated is used. Unfortunately, TensorFlow Federated does not support RFC models to be used in federated learning. Therefore, a new classification model is introduced in experiment 3. The model that is used is a multi-layer perceptron neural network of which the configuration is based on the MLP presented in N. Oliveira et al.'s paper [13]. The same experiment that N. Oliveira et al. executed in their paper (which, to be clear, does not make use of federated learning) is executed in experiment 3. Their approach leads to creating a preprocessed dataset that is subject to great imbalance. Hence, the prediction scores of the attack types "brute force" and "ping scan" are very poor while the prediction scores of "normal", "DoS" and "port scan" are very good as shown in Table V.

Since the results of the third experiment that introduces the MLP model is subject to the negative effects of training a model with imbalanced data, the MLP model is reused in the forth experiment in which it is trained with a balanced dataset. Now, the results are good for all attack types and they are similar to the results of the RFC models used in the first experiments as shown in Table V.

Knowing all of this, it is time to start experimenting with federated learning. Experiment 5 tries to identify whether or not an MLP model that is trained using federated learning can reach competitive prediction scores when compared to the RFC and MLP models that are trained without federated learning. Two federated learning clients are considered by creating two local training datasets out of *internal week 1* and *internal week 2*. These two local training datasets are balanced out (in the same way as done in experiment 1) so that no negative effect due to imbalanced training datasets is experienced. Then, the global MLP model is trained with federated learning and when tested, the conclusion is that the MLP model trained using federated learning is indeed competitive with the RFC and MLP models that were trained without federated learning. The results can be seen in Table V. Thus, the loss of averaging multiple weights of the local MLP models is minimal.

Now that it is known that federated learning is capable of classifying the attack flows, the sixth experiment repeats the base experiment, but in a federated learning setup. This means that there are two clients and each of the clients gets assigned a local dataset. The same flow distributions as in the base experiment are reused, i.e. all "brute force" flows are removed from *internal week 1* and all "ping scan" flows are removed from *internal week 2* while the other attack types receive 3,359 and 3,366 flows respectively. Then, TensorFlow Federated's federated learning implementation, with federated averaging as aggregation algorithm, is used to see whether the global MLP model does learn to recognize flows from the attack types "brute force" and "ping scan", although only one of the clients contains flows from these attack types. The results in Table V show that the global MLP model indeed starts to learn how to classify flows of the attack types "brute force" and "ping scan". However, there is a big difference in the score of "brute force" and the score of "ping scan". The scores of the other attack types are comparable with the results in experiment 4. Compared to the base experiment, the first client is now able to predict 20% of the "brute force" flows as opposed to 0% without federated learning, but also loses in capability of correctly predicting "ping scan" flows. The score drops from about 95% to 67%. The second client improves its score on "ping scan" from 0% to 67% but also loses in capability of predicting the attack type "brute force": the score drops from 98% to 20%. The conclusion for the sixth experiment is that federated learning allows for their clients to start detecting attacks they previously could not detect. But, federated learning is no silver bullet. With the advantage of federated learning, also comes a disadvantage. Namely, that clients can also lose the ability to correctly predict some attacks due to the fact that the weights of the local MLP models of both clients are averaged.

TABLE V

Precision scores of trained RFC models in experiments 3 to 6 (all values in percentages)

Attack type	exp. 3	exp. 4	exp. 5	exp. 6
Normal	99.56	90.33	96.42	98.08
BruteForce	6.179	97.15	81.04	19.76
DoS	99.92	100.0	99.89	100.0
PingScan	0.1831	94.97	92.26	67.00
PortScan	90.21	85.65	85.85	94.52

Finally, a seventh experiment that also uses federated learning and provides additional tests is presented. A first shortcoming of the sixth experiment is that only flows of the attack types "brute force" and "ping scan" are removed from the local datasets of respectively client 1 and client 2. What is the result if other combinations of attack types are removed? A second remark on the sixth experiment is that removing all flows of certain attack types is quite abrupt. Although such scenarios are not impossible, the usability of federated learning is not only limited to scenarios where certain datasets have no flows of a certain attack type while others do. Therefore, some additional tests are provided in the seventh experiment.

In the seventh experiment, instead of removing all flows of an attack type, a small portion of flows of the attack type is left in the local training dataset of a client. The experiment starts with creating more or less balanced datasets (3,359 flows for all attack types except 1,626 for "brute force" in *internal week 1* and 3,366 flows for all attack types except 2,731 for "ping scan" in *internal week 2*) for each client that can be used as a baseline dataset. Then, six tests are executed in which every time, for each of the baseline datasets, 75% of the available flows of one of the attack types is removed. In each test, a different combination of attack types is chosen. This way, each of the combinations of attack types is tested. It turns out that the results of these tests are impressive, as can be seen in Table VI. In this table, the attack types from which 75% of the flows were removed during each test, are indicated by accentuating the corresponding prediction scores in bold and italic. Almost all scores are well above 90%, also for the attack types from which 75% of the flows in the balanced dataset is removed. But, like in experiment 6, "brute force" deviates from this in two occasions. The first test where brute force has a lower score is the test where the combination is "brute force" and "ping scan", this is to be expected since it is the same combination as used in experiment 6. The second test that has the same fate for the score of "brute force" is the test where the combination "brute force" and "port scan" is used. In both tests where the scores for "brute force" deviate from the rest, the biggest group of the samples is predicted to be "port scan". This is also to be expected because this was also observed in experiment 6. A possible explanation for this effect is that the feature distribution of the "brute force" flows is similar to the distribution of the "port scan" flows and until the model has not seen enough "brute force" flows, it cannot separate them correctly from "port scan" flows. But it has to be noted that the deviated score in this seventh experiment is much better than seen in experiment 6. Here, the scores are respectively 80% and 68% which is a big difference with the 20% as seen in experiment 6.

So, after experiments 6 and 7, the overall conclusion about federated learning is that it is certainly valuable to be used in the context of network intrusion detection. Federated learning enables multiple network intrusion detection classification models to learn from each other by sharing and aggregating the weights of the models, rather than sharing the local datasets.

TABLE VI

PRECISION SCORES OF ALL TESTS IN EXPERIMENT 7 (ALL VALUES IN PERCENTAGES)

Attack type	test 1	test 2	test 3
Normal	97.05	96.18	97.21
BruteForce	***93.43***	***80.10***	***68.36***
DoS	***100.0***	99.93	99.93
PingScan	95.66	***95.76***	94.54
PortScan	90.46	90.77	***92.38***

Attack type	test 4	test 5	test 6
Normal	93.58	94.31	94.46
BruteForce	93.44	92.59	94.29
DoS	***99.88***	***99.88***	99.93
PingScan	***95.30***	94.81	***91.91***
PortScan	87.22	***86.75***	***89.62***

ACKNOWLEDGMENTS

First and foremost, I would like to thank Laurens D'hooge for assisting me throughout all steps in the creation of this master's thesis. His feedback and assistance was vital in order for me to come to the thesis as is. Secondly, I would also like to thank my father, Rik Van de Walle, who has thoroughly read my thesis and has given important feedback. Also, thanks to Prof. dr. Bruno Volckaert and Dr. ir. Tim Wauters for giving me the opportunity to do research about federated learning in network intrusion detection. This is, of course, not an exhaustive list of people that deserve to be thanked. Therefore, also a big thank you to all who supported me in the last few months.

REFERENCES

[1] Bill Whitaker, "SolarWinds: How Russian spies hacked the Justice, State, Treasury, Energy and Commerce Departments," 2021.

[2] Lucian Constantin, "SolarWinds attack explained: And why it was so hard to detect," 2020.

[3] Margaret Brennan, ""Hack everybody you can": What to know about the massive Microsoft Exchange breach," 2021.

[4] Setareh Roshan, Yoan Miche, Anton Akusok, and Amaury Lendasse, "Adaptive and online network intrusion detection system using clustering and Extreme Learning Machines," *Journal of the Franklin Institute*, vol. 355, no. 4, pp. 1752–1779, 2018.

[5] AT&T, "Intrusion Detection Systems (IDS) explained," 2021.

[6] Michele Pagano, "Binary and multi-class classification in network anomaly detection using deep neural networks," 2020.

[7] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho, "Flow-based benchmark data sets for intrusion detection," in *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*, pp. 361–369. ACPI, 2017.

[8] Paessler, "All-in-one NetFlow Analyzer PRTG," 2021.

[9] Markus Ring, Sarah Wunderlich, Dominik Gruedl, Dieter Landes, and Andreas Hotho, "Technical Report CIDDS-001 data set," in *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS), to appear*, vol. 001, pp. 1–13. ACPI, 2017.

[10] Abhishek Verma and Virender Ranga, "On evaluation of network intrusion detection systems: Statistical analysis of CIDDS-001 dataset using machine learning techniques," *Pertanika Journal of Science and Technology*, vol. 26, no. 3, pp. 1307–1332, 2018.

[11] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, vol. 54, 2017.

[12] Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad, "Internet of Things intrusion Detection: Centralized, On-Device, or Federated Learning?," *IEEE Network*, vol. 34, no. 6, pp. 310–317, 2020.

[13] Nuno Oliveira, Isabel Praça, Eva Maia, and Orlando Sousa, "Intelligent cyber attack detection and classification for network-based intrusion detection systems," *Applied Sciences (Switzerland)*, vol. 11, no. 4, pp. 1–21, 2021.