2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) | 979-8-3503-3165-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/WoWMoM57956.2023.00082

An Intelligent Mechanism for Monitoring and Detecting Intrusions in IoT Devices

Vitalina Holubenko*†, Paulo Silva*†
* LIS, Instituto Pedro Nunes, Coimbra, Portugal
† CISUC, DEI, University of Coimbra, Coimbra, Portugal

***Abstract*—As of recent years, the growth of data processed by devices has been exponential, resulting of the increasing number of Internet of Things devices connected to the Internet, which has come to play a very critical role in many domains, such as smart infrastructures, healthcare, supply chain or transportation. Despite its advantages, the amount of IoT devices has come to serve as a motivation for malicious entities to take advantage of such devices.**

To deal with potential cyberattacks in IoT devices, Machine Learning techniques can be applied to Intrusion Detection Systems along with Federated Learning to help manage privacy related concerns. Several intrusion detection methods have been proposed in the past, however, there's a lack of research aimed at HIDS. Furthermore, the focus is mostly on applied ML methods and evaluation and not on real-world deployment of such systems. To tackle this, this work proposes a framework for a lightweight host based intrusion detection system based on system call trace analysis for benign and malicious activity detection.

In summary, this work aims to present research about Host Intrusion Detection that could be applied for IoT devices, while leveraging Federated Learning for model updates.

***Index Terms*—Intrusion Detection System, Federated AI, Machine Learning, Internet of Things, Security, Privacy**

I. Introduction

The dispersion and propagation of devices belonging to the Internet of Things (IoT) has propelled the adoption of such wide ranges of Internet connected devices, facilitating far more automation and information exchange than was previously possible. However, as with any device accessible to the Internet, there is no such thing as a completely secure device or network. Malicious parties always look for a vulnerability. IoT devices and applications have also an increased risk of becoming victims of cyberattacks due to a lack of security measures in the IoT ecosystem, which make IoT devices more vulnerable to malicious attacks. Many security flaws and vulnerabilities make devices on the Internet of Things prime targets for exploitation by malware, such is the case in services made accessible such as Telnet and Secure Shell (SSH) are manufactured with default passwords that are hardcoded in the firmware, and the ability to change these passwords is either a difficult process for consumers or is entirely impossible for them. Furthermore, even more recent malware, such as Mirai [5], are capable of launching high-scale floods of Distributed Denial of Service (DDoS) attacks.

IoT is also continuously subject to evolution and is progressively becoming more sophisticated, which is a direct result of the increased ubiquity and processing power of such devices, networking capabilities, introduction of data analytics tools and development of new standards, which in return facilitates data processing in the analysis, management and making intelligent decisions autonomously. This means that the information processed by IoT devices is ever expanding, which may concern relevant information about the owners of the device themselves or other sensitive data.

Given the relevance of cybersecurity in IoT, it is important to research and implement protective measures with minimal impact on the operation of the devices, such an approach is to propose an intrusion detection system that collects and learns from IoT clients' data, while also taking measures that ensure that the data and the device itself is private and secure from malicious entities. To improve the security of IoT devices, some work has been done to detect intrusions on smart devices. ZarpelÃco et al. [6] explained the importance of implementation of host based intrusion detection systems (HIDS) since these systems are able to monitor more information about the device, and therefore can help to detect attacks that could not be identified with only network information. Furthermore, the author explains that traditional HIDS are not effective for IoT, however, research is yet to be improved in this specific area of IDS.

The main objective of this work is based on the development of an Intrusion Detection System (IDS) for IoT devices with the support of Artificial Intelligence (AI) based models. To achieve this task, we perform a state-of-the-art analysis regarding the various aspects that this work entails, such as IDS, how do they work, the overall architecture of such systems and the types of data that could be used for intrusion detection. More specifically, we seek to study host intrusion detection methods that are compatible for IoT oriented systems.

979-8-3503-3165-3/23/$31.00 ©2023 IEEE
DOI 10.1109/WoWMoM57956.2023.00082

We also perform a study of Federated Learning, such as its definition, the uses, why it is a suitable solution for IoT devices and applications. Ultimately, we take a selection of Federated Learning (FL) proposed algorithms and perform an evaluation of the algorithms.

II. Related Works

In this section, we focus on reviewing various intrusion detection systems proposed in the literature for Linux based systems, which is the core Operating System for many IoT devices and Android smartphones.

A. Intrusion Detection

The main objective of an IDS is to examine activities within a system or a network, in order to keep track of possible intrusions from malicious sources. Generally, IDS can be divided into two main groups, Host Intrusion Detection System (HIDS) or Network Intrusion Detection System (NIDS). Furthermore, the detection can be made with two techniques: signature detection (SIDS) or anomaly detection (AIDS). SIDS relies on a system which can be based on known threats', with rules about the collected data, while the second monitors the system behaviour to detect abnormalities that deviate from the normal baseline. Both methods have their benefits, with signature detection being accurate and working well on known threats, but being unable to detect zero-day attacks, contrary to AIDS. On the other hand, anomaly detection requires characterizing and identify the complete normal behaviour of the system, which can be difficult to achieve.

IDS have been studied for a long time, however, traditional IDS are not suitable for IoT for many reasons, such as the limited resources on such devices, which are often insufficient to run a traditional system. In addition, there is considerable heterogeneity in the technologies and network protocols used in IoT. However, little work has been done with HIDS for IoT. In this particular work, we set our focus on host-based intrusion detection systems for IoT.

In HIDS, the data from the host system's audit and logging mechanisms are analysed to look for signs that the system has been broken into or a possible attack.

	DARPA98	KDD99	ADFA-LD	UHN	TON-IoT	LID-IDS
Process IDs	✓	✓	-	✓	✓	✓
Outdates	✓	✓	-	-	-	-
Realistic Traffic	-	✓	✓	✓	✓	✓
Labelled	✓	✓	✓	-	✓	✓
Zero Day attacks	-	-	✓	-	-	-
IoT traces	-	-	-	-	✓	-
Metadata	-	✓	-	-	-	✓
Year	1998	1999	**2014**	2018	2018	2019

Fig. 1. Open source HIDS datasets.

The data may include activity from the local hosts' logins, file access, privilege escalation, alteration of system privileges, system calls, file system changes and application logs, and many others.

B. System Call Based Intrusion Detection Systems

System call traces in particular are very often used in detecting intrusions with HIDS on program level. System call traces, or the sequence of system calls of a given process, are used to find repeated patterns of system calls, enabling anomaly detection and misuse detection during execution, intrusions could be in the form of sub-sequential traces of intrusive activities. A system call trace refers to the system call sequences that have been ordered, performed by a process that a program ran while executing, for instance, a *write*, *read*, *open*, *wait*, *exit* sequence can be considered a system call trace of length of 4. System calls can be extracted, for example, via the *strace* Linux utility, although there are many other ways to collect system call logs. The system call analysis approach to intrusion detection was first proposed by Forrest et al. [10]. Forrest et al. looked at short sequences of system calls to generate profiles of normal program behaviour. This approach is based on an enumeration sequence-based method known as STIDE (Sequence Time Delay Embedding). In this technique, the sliding window method is used to generate short sequences of system call traces and then a database with the signature of the normal behaviour with the invoked sequences. This algorithm was inspired by the natural immune systems of organisms. The approach is claimed to be simple and efficient to deploy for possible real-time implementation. Parameters of system calls are removed to reduce system load and obtain the best result with system call identifiers only. In Forrest's work, it has been observed that simply running a batch of code will affect the system, and as a result, anomalous behaviour of a process/service should leave its system call traces.

Alternatively, analysis of log files can also be applied for intrusion detection. However, it has several restrictions, as more often than not log files are large and have a significant amount of noisy information. On the other hand, an HIDS based on system call renders a better data granularity. However, with increasing complexity of systems, traditional Host Intrusion Systems have several limitations to process large amounts of system call traces in real-time, one of the ways to counteract this is to create an HIDS that has an early detection ability which classifies whether a program is benign or malicious by analysing a specific number of system calls during the initial execution of the process/program.

More recently, the system call analysis problem has been interpreted as a Natural Language Processing (NLP) problem in many research works. For instance,

Lee [11] tried to improve on the results obtained by Forres by using machine learning algorithms to extract information from normal and abnormal sequences of system call traces. Frequency based methods were also explored by Helman and Bhangoo [12], however, instead of keeping track of frequencies of each system call individually, they recorded frequencies of sequences of system calls by tokenizing the system calls into Ngram sequences, which enabled them to preserve the order of the system calls.

Liao and Vemuri [13] took a slightly different approach to using system calls. Instead of keeping track of sequences of calls, they monitor the frequency of system calls that are issued by a program. By doing this, they avoid having to treat every system call individually and reduce the overhead involved in analysing and storing every system call. A system call is considered as an instance of a *text* and the whole set of calls issued as a *document*.

As we can see, many works consider analysis of system calls to detect intrusion on Linux based systems. Creech et al. observed that the existing open source host intrusion datasets (shown in Fig. 1) are mostly outdated and are not representative of contemporary attacks [7].

C. Federated Learning

It is common, in traditional ML scenarios, to have a centralized server or a cloud platform, where the data from several devices (i.e. smartphones, tablets, laptops, smart carts) is aggregated, and consequently, used to train the central model. Such devices nowadays collect vast amounts of data, especially data that is private, which inherently creates a significant risk by having such data stored on a central server. As a result, it is imperative to find solutions that help maintain user data private and secure. FL was introduced in order to minimize these risks. Google, as a solution, proposed Federated Learning in 2016. It was first applied as part of their efforts to devise a decentralized method for using data from mobile devices to improve the user experience of other AI focused solutions. In FL the training, data is kept on device-level, meaning that the client data is not directly transmitted over the network in order to prevent any data from leaking, therefore ensuring a basic level of privacy. The only data to ever be transferred is the locally computed updates from each device to a central coordinating server, which is a cloud-based distributed service (referred to as a server model) where they are aggregated.

Since its introduction, Federated Learning has revolutionized the field of IoT applications by providing new AI based solutions due to its distributed and private-driven nature. Federated Learning is particularly attractive for building distributed IoT systems due to the recent advances in mobile technologies and the overgrowing concerns of risks related to user privacy, by pushing AI technologies to the network edge in IoT devices. In this case, FL enables the cooperative training of a shared global model, which benefits both network operators and IoT clients in terms of network resource savings (because data is decentralized) and privacy enhancements because user data is not shared to a central entity.

III. Methodology

In this section, we discuss our approach for developing an effective Host Intrusion Detection System based on system call analysis along with its evaluation results. System calls traces were collected from various Linux based systems. Figure 2 illustrates the detailed steps followed by the proposed Host Intrusion Detection System. Additionally, in the next sections, we discuss the evaluation approach of the Machine Learning methodologies that were then applied in the framework.

In this work, the proposed system should enable deployment in lightweight IoT devices with significantly less computation power than classical computer systems. To be deployed, the system needs to demand the minimum possible storage, and run on restrained memory resources, the setback of making this optimized for lightweight devices is that there is a possible effect in accuracy due to the ML model's size.

The experiments were done on three virtual machines on a private network, whose requirements were set up to closely emulate the resources of the target devices, with ARM64 architecture and a QEMU 6.1 ARM system.

The starting point of the whole system is the data extraction/tracer module, which the devices are running locally. This module collects and aggregates the system call traces, which are going to be processed and then are going to be the subject of automated analysis by a ML model unit that classifies the input as anomalies or normal, and then raises an alert when an intrusion is detected.

The proposed architectures are a centralized model and a decentralized FL-based approach, where each of the nodes represent IoT clients. The main idea of this approach is to establish a comparison between the traditional IDS approaches and the federated setting, where the centralized architecture serves as baseline. The process of training the ML models is depicted in Figures 8 (centralized and federated approach, respectively). In Figure 7 provides a more detailed view of the preprocessing, training and evaluation stage.

A. Data Acquisition Process

As the approach is based on continuous analysis of system call flows in devices, we need to capture the dynamic properties of processes that are running on the

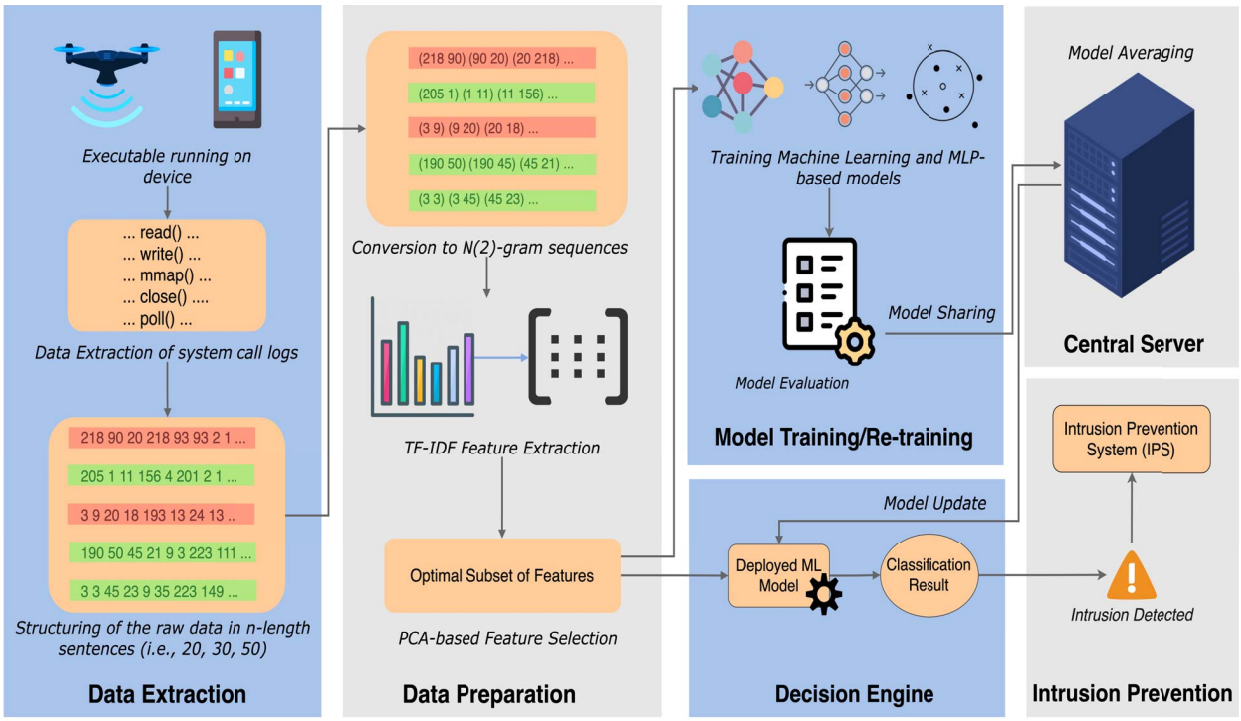

Fig. 2. Overall Architecture of the HIDS.

system. Thus, system call traces of running processes are to be extracted and analysed. The use of system call sequences for generating models that detect normal and anomalous sequences is justified by the fact that security violations on device level are likely to produce abnormal system call invocations. System calls are the only means by which a program operating in user space can enter kernel space and make use of the services provided by the kernel. The user space processes running on the Operating System (OS) make system calls to request services from the kernel.

In order to achieve the goal of producing a system that analyses system call logs to detect intrusions, we start by performing log collection, both benign and malign traces. To do this, we are using a Linux tool called *perf* to extract system calls in real time. *Perf* is a Linux tool used for profiling, refined and upgraded by Linux kernel developers.

In the data collection phase, sequences are recorded in the database as part of the normal profile. These profiles usually consist of thousands of short sequences of system calls. Since the majority of the IoT devices are built using the Linux operating system, the benign samples are going to be collected from the system/executables on a Linux based Virtual Machine. These files are going to be executed for one minute using *perf* tool to generate system call traces for the respective process file.

In the generation of anomalous data phase, we simulate an environment of the OS which is under intrusion and capture its behaviour under those circumstances. Which is used in order to train a model that recognizes anomalous device behaviour. To accomplish this task, we resort to various tools to perform a DoS attack and a brute force attack on a SSH port.

The reason of the simulation with the device running a DoS attack on a server is due to the increasing occurrences of Mirai Botnet attacks. In these instances, the IoT devices of a particular network are infected by an attacker, and then are used as means to perform the Distributed Denial of Service (DDoS) attack on a target server. So, this attack allows for the simulation of a how a particular device would behave if it was to be used as a botnet for a DDoS attack.

Brute force via hydra was also used as an attack vector. Brute-force attacks represent the use of a traditional brute-force password-cracking application, *Hydra* to discover users' passwords over Secure Shell (SSH) and File Transfer Protocol (FTP) ports. Most IoT devices still use the default credentials for authentication, and a lot of devices don't even use credentials at all. In IoT, Telnet and SSH brute force attacks still account for nearly 70% of all IoT attacks [9].

Another type of intrusion that was tested was a *nmap* scan, which discovers other devices on a traditional

TCP/IP network, it is common for an attacker to want to exploit a network in which the IoT device belongs to, to do that, the attacker first has to analyse the network in order to find potential targets systems. To accomplish that, a network scan via the *nmap* tool can be done.

B. Data Preprocessing

In this step, we proceed to the system call sequence generation. Each line of the recorded trace file represents a log related to a system call that comprises execution time of the system call, process name, process ID, name of the system call, related parameters, and return value. The system call names and associated process ID were extracted from each of the logs, and converted them into system call numbers by referring to the system call table for the specific Linux version that was used, as each system call has a distinct numeric reference.

The resulting data that was extracted is composed of system calls, each executed system call per line. These sequences need to be treated and processed in order for us to have a structured dataset that can be used for training ML models. In this step, we tokenize the system calls into sequences of variable length. For instance, let's consider the following snapshot of system calls recorded on a normal setting:

open(), mmap(), read(), socket(), mmap(), execve(), open(), read(), close(), brk()

With a sequence size of 5, the following subsequences would be produced by the sliding window method of size 1:

Sequence 1: *open(), mmap(), read(), socket(), mmap()*
Sequence 2: *mmap(), read(), socket(), mmap(), execve()*
Sequence 3: *read(), socket(), mmap(), execve(), open()*
Sequence 4: *socket(), mmap(), execve(), open(), read()*
Sequence 5: *mmap(), execve(), open(), read(), close()*
Sequence 6: *execve(), open(), read(), close(), brk()*

This process is applied to all the system call sequences that are captured by the data extraction executable and used to train a ML model to classify normal and abnormal behaviour.

In case of an anomaly, lets, for instance, consider the following pattern was observed:

socket(), mmap(), execve(), open(), write()

This would generate an alarm with the patterns stored in the normal database. Thus, the anomalously behaving process, which may have loaded another program (by the *execve()* call) and opened a file and written to it (instead of reading from it) would be detected.

After the tokenization process, we'll clean the data, by removing the intersecting rows (sequences) from both normal and attack datasets in order to maximize the distinction between the two class types for the learning process. A row with a normal sequence is set with a label of 0, whereas a row with an intrusive sequence is set with a label of 1.

Having done that, we follow that up with a feature extraction process. This process can be done through various other approaches, in this work we evaluate the following three techniques.

1) Trivial Representation: The most basic representation of a system call trace is to consider it as a string (sequence) of system calls. Let us consider an operating system with total m number of unique system calls, then set of system calls can be represented by $U = s_1, s_2, s_3, s_4, s_m$.

2) Vector Space Model: Vector Space Model is a very commonly used technique in information retrieval field to represent a document as a set of words. This technique is also known as the "bag of words" technique, in which each word in a given document is assigned to a weight, this determines how much the document is relevant to specific words. In the process, the weight is assigned to a word as the number of times the word appear in the document. In the context of system call representation, system call trace is considered as a document and each system call as one word. Then we can apply a vector space model to represent a given system call trace as a feature vector. To represent the system call traces using vector space model representation, let us consider a feature set, as a set of vectors that corresponds to a set of system call traces of the system. System call trace for an application *i* with this model can be represented as a vector, which represents the number of times the system call appears in the system call trace sequence/row.

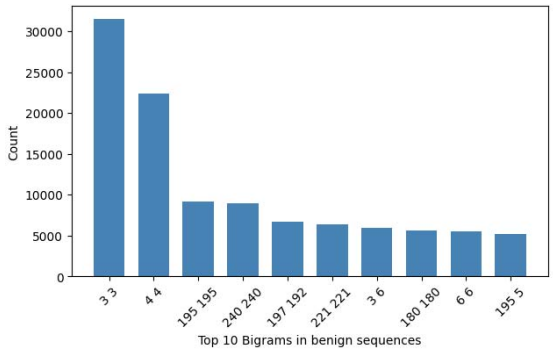

Fig. 3. Top 10 frequent 2-gram for benign sequences

3) Term Frequency–Inverse Document Frequency: In any document corpus, certain words are more common than others. An assumption was made that the same idea applies to logs of system calls, that certain sequences refer to all operations of any program in an operating system and certain sequences refer to specific operations that may or may not be malicious. In document classification, one of the most popular model to derive weights to words that occur in a document is the TF-IDF. Term Frequency (TF) is referent to the occurrences of a certain word in a document, while Inverse Document Frequency (IDF) pertains to the number of times the word occurs in the document corpus. The big issue with the vector space model representation is that the relational order of the features/words in the input data is not kept in the training input information. To keep that data relative to the order of the words in the input sequence, we can to consider tuples of words (or system calls) and not exclusively the single words, as is done in the vector space model. This method is called the n-gram technique, which is defined as a tuple of n words where n is, in general, a small positive integer.

The benefit of applying this type of feature extraction technique is the fact that calculating the weights of the n-grams present in the system call traces is not computationally expensive. However, the n-gram approach can lead to another problem which is the high sparsity of the data matrix, which means we can end up with a high number of features, which can be solved by applying a feature selection method in order to reduce the dimensionality of the input matrix.

In order to perform some data analysis on the dataset, we created the n-gram representational model. In particular, we analysed the top 10 frequent bigram sequences and top 10 common frequent bigram sequences. Figure 3 and 4 show the top-10 most frequent 2-gram sequences for benign and malign traces, respectively. From the graphs, we can observe that the patterns for benign and malware traces are different. For example, the sequences "3 3" (3 corresponds to the *close*) and "4 4" (4 corresponds to the *stat*) is dominant in benign traces, while sequence "14 14" (14 corresponds to *rt_sigprocmask*) and "168 168" (168 corresponds to *swapoff*)is more frequent in malicious traces. In both types of traces have some unique frequent sequences, which can greatly help machine learning models to distinguish between benign and malicious traces.

To better understand the patterns between benign and malware traces, we also counted the top 10 common bigrams sequences in the two types of system call traces (seen in Figure 5). For instance, sequence "10 265" (10 and 265 correspond to *mprotect* and *linkat*, respectively) appears in similar amounts of time in benign and malign sequences. This means that by analysing the top common sequences between the two types of traces, we can find the sequences that show up commonly in the both traces and it is possible to discard them, as they will most likely not be useful in differentiating between the two classes.

C. Feature Selection

Feature retrieval techniques play a major role in differentiating malicious traces from non-malicious traces. Existing feature retrieval techniques mainly focus on different approaches.

Some features of the data may be more important than others for the tasks of anomaly detection or attack classification. Noisy features can exaggerate the minor discrepancies in the dataset, and may reduce the predictive performance of classification models. Thus, feature selection is considered a crucial approach to eliminate irrelevant attributes. Feature selection reduces the training time involved, as few features are used to develop model and for predictability of samples in future. Through feature selection, high–ranked system calls are determined that have greater ability to identify malware from benign executable. Through our study,

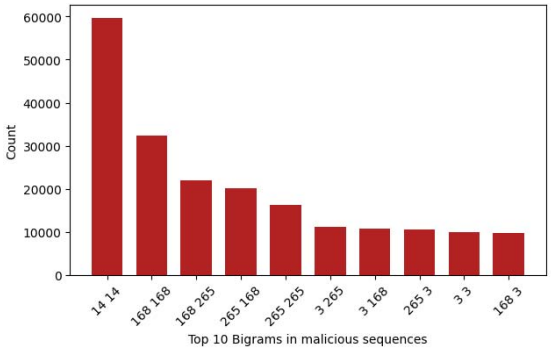

Fig. 4. Top 10 frequent 2-gram for malign sequences

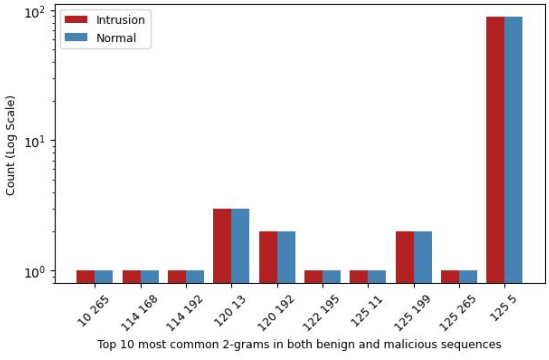

Fig. 5. Top 10 most common 2-grams between benign and malign sequences

we also assess the impact of varying feature length on detection accuracy and performance. For dimension reduction, Principal Component Analysis (PCA) can be applied. This method allows the classification framework to use fewer features and helps to make the detection system faster and less demanding in terms of memory resources.

D. Datasets

In the evaluation of ML models, we used two datasets. Firstly, the ADFA-LD, which is a collection of system call traces. A single trace is a sequence of system call numbers, over some arbitrary time period. The traces were collected from an Ubuntu Linux 11.04 operating system. The ADFA-LD labelled dataset is the successor of the KDD collection that applied the latest publicly available exploits and methods for its data generation.

In addition to the ADFA-LD, we also generated our own version of an HIDS dataset in order to validate the deployed models on the system in real time. This dataset based on system calls, with normal traces and attack traces that were described in the data acquisition process, and validated the developed intrusion system with that dataset with a different set of attacks.

E. Training Approach

In the training phase, the class were balanced as 50% benign traffic and 50% attack traffic, making the dataset perfectly balanced for binary classification.

It is important to note that this rebalancing of the data can lead to different problems, since in real world scenarios the amount of benign data surpasses significantly malicious data. Both train and test sets are indeed affected by each change, and the goal is to compare the impact on the model performance when the classes are balanced.

F. Centralized Approach

In the centralized setting, we'll apply the following traditional ML algorithms:

K-Nearest Neighbour (KNN) classifies an observation by counting the majority classes of the nearest neighbours. This model enhances its performance by minimizing intra-variability within a group while maximizing inter-variability between different groups. For simplicity, we choose the number of neighbours to be 3. Still, any odd number of k is recommended to avoid a tie situation.

The *Decision Trees (DT)* algorithm relies on a set of rules, derived from the training process, to partition data into groups that are as homogeneous as possible. The goal is to generate a Decision Tree to classify normal sequences from intrusion sequences at the lowest possible error rate. That way, it can be generalized to the testing set as well as future unseen data.

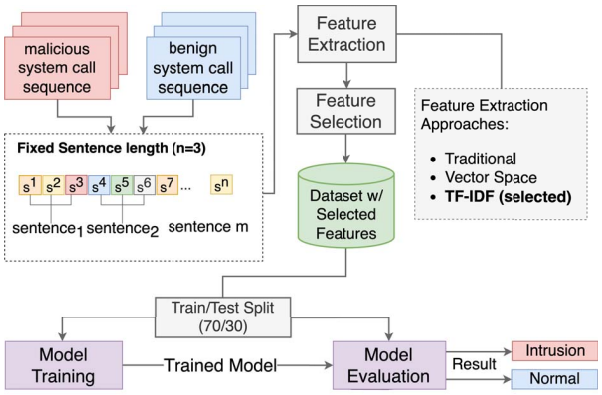

Fig. 6. Steps for the data preprocessing and training process

Random Forest (RF) distinguishes normal sequences from intrusion sequences using ensemble learning method. RF creates multiple Decision Trees to classify the same observation. The final decision is based on the wisdom of the crowd that is the majority predicted class of that particular observation.

Support Vector Machine (SVM) is a supervised machine learning algorithm which is good at detecting anomaly by separating normal behaviour from the other using different kernel types.

Naïve Bayes (NB), This algorithm s known for classifying data based on conditional probabilities without making any assumption. Given lots of labelled sequences, we hope that this model can efficiently distinguish intrusion sequences from normal sequences. A predicted label is determined based on the highest probability of a class.

We use *Neural Networks (NN)*, more specifically MLPs, both in the centralized and federated scenarios, as it is parametric and model aggregation can be performed. Neural networks are efficient at learning underlying complex relationships because of its composite architecture. The model is composed of four layers: an input layer, two ReLU layers. and a sigmoid output layer with two output nodes, where each one represents a probability of a sequence belonging to either class.

G. Federated Learning Approach

In this section, we seek to propose the FL that are developed and evaluated in the described scenario.

FedAvg as a baseline Federated Learning algorithm, *FedAvg* was chosen for implementation, as it is simple and widely used in FL scenarios. It is a simple algorithm, which takes the parameters of all participating clients in an epoch, and averages the values, resulting on a global model.

Weighted Federated Averaging, from the implementation of *FedAvg* we constructed a weighted version of it

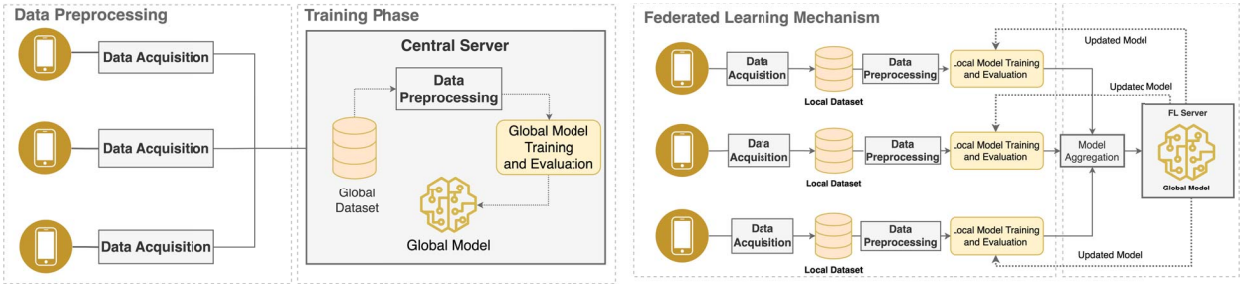

Fig. 7. Training process, centralized setting and federated setting

that scales the client weights according to the amount of data points that each client has, the more data points the client has, the more weight the model parameters have in the final calculation of the global model, and vice versa when the client has fewer data points.

IV. Evaluation Approach

To demonstrate the effectiveness of both the baseline and federated approach, and to evaluate the approaches, some performance metrics were selected to achieve that. The performance of an IDS can be measured by the following metrics.

True Positive Rate (TPR) - TPR is calculated as the ratio between the number of correctly predicted attacks and the total number of attacks. The TPR can be expressed mathematically as:

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

False Positive Rate (FPR) - FPR is the ratio between the number of normal instances incorrectly classified as an attack and the total number of normal instances, expressed as:

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

Accuracy - The accuracy measures how accurate the IDS is in detecting normal or anomalous traffic behaviour. It is described as the percentage of all those correctly predicted instances to all instances:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Precision - Precision measures the accuracy of positive predictions:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

F1-score - Weighted average of the precision and recall, defined as follows:

$$F1 - score = \frac{2 * TP}{2 * TP + FP + FN} \tag{5}$$

Number of Rounds - Amount of training rounds needed in order to reach a certain training accuracy threshold. Since our aim is to reproduce a model that is capable of reach convergence quickly, needing less communication rounds with the server.

V. Experimental Results

Tables I, II and III show the performance of different machine learning algorithms on ADFA-LD dataset in regard to performance indicator accuracy, recall, false positive rate and true positive rate. Our goal is to find the best candidate model with high accuracy, high recall but low FPR and FNR. High performance measures are bolded so that we can easily identify the best candidate model.

The approach of vector space and TF-IDF representation have shown to be better suited for model training, since in general the results with those approaches have showed to be higher than with the traditional representation.

TABLE I
Trivial Representation (ADFA-LD)

Algorithm	*Accuracy*	*Precision*	*TPR*	*FPR*	*F1*
KNN	0.86	0.38	0.75	0.12	0.88
Decision Tree	0.95	0.64	0.93	0.07	0.95
Random Forest	**0.97**	**0.82**	**0.97**	**0.02**	**0.98**
SVM	0.71	0.68	0.64	0.22	0.71
Naive Bayes	0.53	0.50	0.48	0.46	0.42
MLP	0.72	0.22	0.80	0.33	0.78

TABLE II
Vector Space Representation (ADFA-LD)

Algorithm	*Accuracy*	*Precision*	*TPR*	*FPR*	*F1*
KNN	0.99	0.99	0.98	0.00	0.99
Decision Tree	0.96	0.94	0.94	0.06	0.95
Random Forest	**0.98**	**0.97**	**0.97**	**0.02**	**0.98**
SVM	0.93	0.91	0.89	0.02	0.93
Naive Bayes	0.86	0.80	0.84	0.13	0.86
MLP	0.98	0.95	0.98	0.02	0.98

Decision Tree, Random Forest, KNN and MLP are the best candidate algorithms since they achieved higher accuracy, f1-score and precision, yet at a lower false positive rate.

In terms of False Positives, the results reveal that Random Forest, SVM, KNN, and MLP models outperform other classifiers by a good margin, ranging from 0% to 0.02%. The best result achieved when looking at model accuracy, Random Forest was able to obtain a value of 99%, with vector space representation and TF-IDF.

To verify that the Federated Learning (FL) approach fits in this scenario properly, it is necessary to compare it with traditional solutions. In the federated approach, we took two ways of evaluating the algorithms.

In the first one, the data is distributed equally among peers, making the data iid. In the second approach, the data is randomly distributed among the clients, in other words, non-iid. Table IV show the metrics obtained with the experimentations done across all the scenarios.

The performance of the two federated algorithms was similar, especially when iid data was used, this can be explained due to the fact that since the distribution of the data across the clients is almost similar, which will make the algorithms of Weighted Federated Averaging behave similarly to the standard Federated Averaging algorithm. When the data is not equally distributed, however, we can see that Weighted Federated Averaging has a slightly better performance than Federated Averaging, and is able to reach better accuracy with the scaling of the weights.

The evaluation results with the generated dataset have shown to be of good performance, in both centralized (Table IV) settings, obtaining performance results better than ADFA-LD. This, however, can be due to the fact that the data used, as mentioned previously, is very limited, and the interpolation between normal traces and attack traces is almost non-existent in this dataset, contrary to ADFA-LD, so the model is able to distinguish between the two classes very easily within the data that was captured.

TABLE III
TERM FREQUENCY INVERSE-DOCUMENT FREQUENCY (ADFA-LD)

Algorithm	*Accuracy*	*Precision*	*TPR*	*FPR*	*F1*
KNN	0.91	0.93	1.00	0.21	0.90
Decision Tree	0.98	0.98	0.98	0.02	0.98
Random Forest	**0.99**	**0.99**	**99**	**0.01**	**0.99**
SVM	0.96	0.95	0.94	0.02	0.96
Naive Bayes	0.76	0.76	0.78	0.25	0.76
MLP	0.98	0.99	0.98	0.01	0.98

TABLE IV
FEDERATED LEARNING (ADFA-LD)

Algorithm	*Accuracy*	*Precision*	*TPR*	*FPR*	*F1*
FedAvg (iid)	**0.96**	0.97	**0.97**	**0.03**	**0.96**
FedAvg (non-iid)	0.92	0.91	0.91	0.06	0.92
WFedAvg (iid)	**0.96**	**0.97**	**0.97**	**0.03**	**0.96**
WFedAvg (non-ßiid)	0.95	0.93	0.94	0.03	0.95

TABLE V
TERM FREQUENCY INVERSE-DOCUMENT FREQUENCY (GENERATED DATASET)

Algorithm	*Accuracy*	*Precision*	*TPR*	*FPR*	*F1*
KNN	0.68	0.63	0.62	0.32	0.65
Decision Tree	0.96	0.93	0.99	0.05	0.97
Random Forest	**0.99**	**0.98**	**1.00**	**0.01**	**0.99**
SVM	0.99	0.71	0.00	0.00	0.50
Naive Bayes	1.00	0.90	0.93	0.00	0.96
MLP	**1.00**	**0.94**	**0.92**	**0.00**	**0.96**

TABLE VI
FEDERATED LEARNING (GENERATED DATASET)

Algorithm	*Accuracy*	*Precision*	*TPR*	*FPR*	*F1*
WFedAvg (iid)	**0.995**	**0.99**	**1.00**	**0.00**	**1.00**
WFedAvg (non-iid)	0.95	0.93	0.94	0.03	0.95

VI. MODEL DEPLOYMENT

After evaluating the results and selecting the best model for our objectives, we developed a client-server architecture for the deployment of our intrusion detection system and federated learning supported approach.

The architecture includes a Linux deployable executable as a client interacting with a Federated server developed in Python and TensorFlow. Using the local executable, clients can monitor system call traces of other apps in their system, train local models with local data and share those models with the server for model aggregation. The server receives model parameters from each of the clients, averages them, and sends the results back to each of the clients for model updates.

A. Client Program

Figure 8 illustrates the interface of the developed service as a client node and the general output of the classification phase. By simply starting up the service, the program starts by executing the *perf trace* command in the background of the system and starts with the system call trace classification process. It is possible, as well, to monitor a set of programs, by specifying the PID of the processes that we want to trace or filter out, in the environment variables. Additionally, the client is responsible for collecting the local data and re-training the ML model from time to time, which results in an updated model that is then sent to the global server for model aggregation.

B. Global Server

The global server is located on a remote server, which the clients connect to from time to time for model updates. When a client connects to the server, the

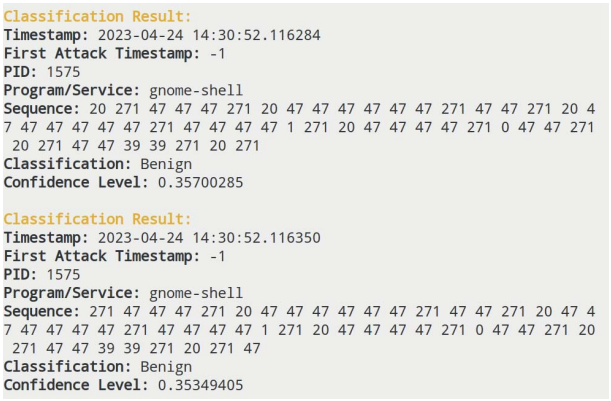

Fig. 8. Client HIDS Interface

server accepts the connection and initializes a connection socket. Once the data is transferred from the client, the server receives the data, and it performs the weighted federated averaging technique that was described in a previous section. After that, the server returns the model parameters to each client via a .JSON file, which will serve as the new model parameters for input classification.

VII. Conclusion

In this work, the input attributes are system calls which are generated from malicious samples occurred in real time IoT attacks made in IoT devices. So the designed a ML model using a simple Neural Network architecture is proved to be efficient in classifying system calls into benign and real time malicious samples and it is evaluated in terms of performance metrics like accuracy, precision and recall.

The evaluated algorithms that had a good performance with the experiments done are based on supervised learning techniques, this means that the developed models can solely detect attacks similar to what they have learned from the dataset. As a consequence, a limitation of this work is the number and relevance of attacks that have been learned, and the lack of real IoT based malware being used, since most IoT threats are not open-source.

This work can be further improved by performing multi-class classification on system calls. In multi-class classification these malware samples are further divided into many classes such as virus, trojans, worms, ransomware and botnets. This kind of classification helps to enhance the analysis of malware samples in an elaborative manner and it also gives insight view for the behaviour of each malware samples. Furthermore, other efficient machine learning algorithms can be considered, such as deep learning techniques, like CNN, RNN and LSTM for both binary and multi class classification. Another feature that would also be interesting to explore is the scalability of the analysis engine, since the recorded traces are analysed independently of each other in each device, the solution can be migrated to a monitoring server (or multiple) to mitigate high performance overhead on the devices.

This work overall obtained promising results. However, future work could be focused on the learning step processing time for the federated learning, since the models have to be updated frequently to account for new threats.

Acknowledgment

This work was partially carried out in the scope of the ARCADIAN-IoT - Autonomous Trust, Security and Privacy Management Framework for IoT, Grant Agreement Number: 101020259. H2020-SU-DS02-2020.

This work was also partially funded by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and byuropean Social Fund, through the Regional Operational Program Centro 2020.

References

[1] Deepti Sehrawat and Nasib Singh Gill. Smart sensors: Analysis of different types of iot sensors.
[2] Nashwan Othman and Ilhan Aydin. A face recognition method in the internet of things for security applications in smart homes and cities.
[3] Farzad Samie, Lars Bauer, and Jörg Henkel. Iot technologies for embedded computing: A survey. 10 2016.
[4] G. Karunarathne, K. Kulawansa, and Mohamed Firdhous. Wireless communication technologies in internet of things: A critical evaluation, 2018.
[5] M. M. Die, “Mmd-0056-2016-linux/mirai, how an old elf malcode is recycled,” 2016.
[6] ZarpelÃco BB, Miani RS, Kawakani CT, de Alvarenga SC (2017) A survey of intrusion detection in internet of things.
[7] Creech, Gideon and Jiankun Hu. “Generation of a new IDS test dataset: Time to retire the KDD collection.” 2013 IEEE Wireless Communications and Networking Conference (WCNC) (2013): 4487-4492.
[8] M. Nobakht, V. Sivaraman and R. Boreli, ”A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow,” 2016 11th International Conference on Availability, Reliability and Security (ARES), 2016
[9] The commonality of the brute force iot attack, https://brilliancesecuritymagazine. com/cybersecurity/the-commonality-of-the-brute-force-iot-attack/
[10] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, 1996.
[11] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.
[12] P. Helman and J. Bhangoo. A statistically based system for prioritizing information exploration under uncertainty. Trans. Sys. Man Cyber, 1997.
[13] Yihua Liao and Rao Vemuri. Using text categorization techniques for intrusion detection, 2002.