**Secure management of IoT devices lifecycle through identities, trust and distributed ledgers**

**D4.4 Federated threat analysis models for continuous risk assessment**

**Document Summary Information**

<table>
<thead>
<tr>
<th>Grant Agreement No</th>
<th>Acronym</th>
<th>ERATOSTHENES</th>
</tr>
</thead>
<tbody>
<tr>
<td>101020416</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Full Title</th>
<th>Secure management of IoT devices lifecycle through identities, trust and distributed ledgers</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Start Date</th>
<th>Duration</th>
<th>42 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/10/2021</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project URL</th>
<th><a href="http://www.eratosthenes-project.eu">www.eratosthenes-project.eu</a></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>D4.4 Federated threat analysis models for continuous risk assessment</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Work Package</th>
<th>WP4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Contractual due date</th>
<th>Actual submission date</th>
<th>27/06/2023</th>
</tr>
</thead>
<tbody>
<tr>
<td>30/06/2023</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Dissemination Level</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsible author</th>
<th>Lead Beneficiary</th>
<th>Lead Beneficiary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pablo Ramirez Hereza, Juan Manuel Vera Díaz</td>
<td>ATOS</td>
<td>ATOS</td>
</tr>
</tbody>
</table>

| Authors | |
|---------| |
| Pablo Ramirez Hereza, Juan Manuel Vera Díaz | |

<table>
<thead>
<tr>
<th>Internal reviewers</th>
<th>INLE, DBC</th>
</tr>
</thead>
</table>

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No 101020416.
Revision history (including peer reviewing & quality control)

<table>
<thead>
<tr>
<th>Version</th>
<th>Issue Date</th>
<th>% Complete</th>
<th>Changes</th>
<th>Contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.0</td>
<td>30/03/2023</td>
<td>0%</td>
<td>Initial Deliverable Structure</td>
<td>Juan Manuel Vera Díaz, Pablo Ramírez Hereza (ATOS)</td>
</tr>
<tr>
<td>V0.1</td>
<td>25/04/2023</td>
<td>15%</td>
<td>Added Content in Chapters 1 and 2</td>
<td>Juan Manuel Vera Díaz (ATOS)</td>
</tr>
<tr>
<td>V0.2</td>
<td>23/05/2023</td>
<td>70%</td>
<td>Added Content in Chapter 3, and sections 4.3, 5.1, 5.3.1 and 6.2</td>
<td>Juan Manuel Vera Díaz (ATOS)</td>
</tr>
<tr>
<td>V0.3</td>
<td>25/05/2023</td>
<td>85%</td>
<td>Added Content in sections 4.1 and 4.2</td>
<td>Pablo Ramírez Hereza (ATOS)</td>
</tr>
<tr>
<td>V0.4</td>
<td>27/05/2023</td>
<td>100%</td>
<td>Added Content in sections 5.2, 5.3 and 5.4. Final conclusions added</td>
<td>Pablo Ramírez Hereza, Juan Manuel Vera Díaz (ATOS)</td>
</tr>
<tr>
<td>V0.5</td>
<td>29/05/2023</td>
<td>100%</td>
<td>Review 1</td>
<td>Pablo Ramírez Hereza, Juan Manuel Vera Díaz (ATOS)</td>
</tr>
<tr>
<td>V0.6-0.7</td>
<td>27/06/2023</td>
<td>100%</td>
<td>INLE &amp; DBC reviews completed</td>
<td>Pablo Ramírez Hereza, Juan Manuel Vera Díaz (ATOS)</td>
</tr>
</tbody>
</table>

Disclaimer

The content of the publication herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the ERATOSTHENES consortium make no warranty of any kind with regard to this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the ERATOSTHENES Consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the ERATOSTHENES Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

© ERATOSTHENES Consortium, 2020-2025. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.
Table of Contents

1 Executive Summary ....................................................................................................................... 6
2 Introduction .................................................................................................................................. 7
   2.1 Mapping ERATOSTHENES Outputs ......................................................................................... 7
   2.2 Deliverable Overview and Report Structure ............................................................................. 9
3 Architecture Orientation and Industrial Requirements ................................................................. 10
   3.1 Architectural Positioning and design decisions ...................................................................... 10
      3.1.1 Overall Architecture of FedLPy ....................................................................................... 11
      3.1.2 Training-Inference Flow Diagram ................................................................................... 12
      3.1.3 Interconnection with ERATOSTHENES tools ................................................................ 13
   3.2 Business, Industrial Positioning and End-User Requirements ................................................. 15
   3.3 Methodology .......................................................................................................................... 16
4 Federated Learning Framework for Threat Analysis ..................................................................... 17
   4.1 The FedLPy package .............................................................................................................. 17
      4.1.1 Federated Learning Module ............................................................................................. 17
      4.1.2 Differential Privacy Module ........................................................................................... 18
      4.1.3 Common module ............................................................................................................. 20
   4.2 FedLPy-based Server & Client .................................................................................................. 20
      4.2.1 Server ............................................................................................................................... 20
      4.2.2 Client ................................................................................................................................ 21
      4.2.3 Configuration Files ........................................................................................................... 22
   4.3 Deployment ............................................................................................................................. 23
      4.3.1 Deployment Script ............................................................................................................. 23
5 Research and Scientific Innovation ............................................................................................... 28
   5.1 Description of the databases .................................................................................................... 28
      5.1.1 Datasets ............................................................................................................................ 28
      5.1.2 Dataset Preprocessing ...................................................................................................... 32
   5.2 Evaluation protocol .................................................................................................................. 32
      5.2.1 Overall description ........................................................................................................... 33
      5.2.2 Federated Threat Analysis test bed ................................................................................... 33
      5.2.3 Evaluation metrics ........................................................................................................... 33
   5.3 Initial models ............................................................................................................................ 34
   5.4 Preliminary results .................................................................................................................... 35
      5.4.1 Federated Threat Analysis ............................................................................................... 35
      5.4.2 Privacy-preserving Federated Threat Analysis with DP-SGD ........................................... 36
6 Conclusions .................................................................................................................................. 38
   6.1 Main Conclusions .................................................................................................................... 38
   6.2 Future Improvement Lines ......................................................................................................... 38
7 References ...................................................................................................................................... 39

List of Figures

Figure 1: Current ERATOSTHENES architecture. ........................................................................... 10
Figure 2: ERATOSTHENES federated learning training and continual inference overall architecture. ........................................................................................................... 11
Figure 3: Flow chart example of a Federated Learning Server-Clients communication. ................... 13
Figure 4: Different approaches to interconnect ADI sub-module and FedLPy. (a) ADI as the initial FedLPy model (b) ADI as a FedLPy client. (c) ADI independent of FedLPy. ........................................................................................................ 14

Figure 5: FedLPy Structure ............................................................................................................................... 17

Figure 6: Private Aggregation of Teachers ensembles schematized. This image is based on [2].......................... 19

Figure 7: Front End initial version included in FedLPy .................................................................................... 22

Figure 8: General IoT network deployment PoC ................................................................................................. 33

Figure 9: Confusion matrix for Federated Threat Analysis with Binary FCNN..................................................... 36

Figure 10: Confusion matrix for Federated Threat Analysis with Multiclass FCNN ............................................ 36

Figure 11: Confusion matrix for Federated Threat Analysis with DP-SGD-based Binary FCNN .......................... 37

Figure 12: Confusion matrix for Federated Threat Analysis with DP-SGD-based Binary FCNN ....................... 37

List of Tables

Table 1: Adherence to ERATOSTHENES GA Deliverable & Tasks Descriptions.............................................. 7

Table 2: Requirements from D1.2 .................................................................................................................... 15

Table 3: UNSW-NB15 considered attacks extracted from [14]........................................................................ 28

Table 4: Aposemat IoT-23 considered attacks extracted from [15]................................................................ 29

Table 5: ToN IoT considered attacks extracted from [16].............................................................................. 30

Table 6: Merged dataset considered attacks .................................................................................................. 31

Table 7: Common features used for the merged database ............................................................................. 32

Table 8: Architecture of Binary FCNN model ................................................................................................. 34

Table 9: Architecture of MultiClass FCNN model .......................................................................................... 35

Table 10: FCNN performance for binary classification ................................................................................... 36

Table 11: Federated Threat Analysis with Multiclass FCNN ........................................................................ 36

Table 12: DP-SGD-based FCNN performance for binary classification............................................................ 37

Table 13: DP-SGD-based FCNN performance for binary classification........................................................... 37
Glossary of terms and abbreviations used

<table>
<thead>
<tr>
<th>Abbreviation / Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>Anomaly Detection Inspector</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>DP</td>
<td>Differential Privacy</td>
</tr>
<tr>
<td>DP-SGD</td>
<td>Differential Privacy Stochastic Gradient Descent</td>
</tr>
<tr>
<td>FC</td>
<td>Fully Connected</td>
</tr>
<tr>
<td>FCNN</td>
<td>Fully Connected Neural Network</td>
</tr>
<tr>
<td>FML</td>
<td>Federated Machine Learning</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MVP</td>
<td>Minimum Viable Product</td>
</tr>
<tr>
<td>OS</td>
<td>Operative System</td>
</tr>
<tr>
<td>PATE</td>
<td>Private Aggregation of Teacher Ensembles</td>
</tr>
<tr>
<td>PoC</td>
<td>Proof of Concept</td>
</tr>
<tr>
<td>PPML</td>
<td>Privacy-Preserving Machine Learning</td>
</tr>
<tr>
<td>TMB</td>
<td>Trust Manager &amp; Broker</td>
</tr>
</tbody>
</table>

© ERATOSTHENES, 2020
1 Executive Summary

The Federated Threat Analysis block is a fundamental component of the ERATOSTHENES project since it provides Artificial Intelligence (AI) based models to allow a continuous network risk assessment as a defence service for detecting potential attacks within the Internet of Things (IoT) networks. To consider the different network traffic flows that the edge devices can observe, a federated learning system is implemented in this task, thus composing a global aggregated model that works for each of the nodes with good performance.

This deliverable (D4.4) describes the techniques implemented for the proposed Federated Machine Learning (FML) system, resulting in the Python package FedLPy as the main output of T4.4. It also presents the first results of the analysis of potentially malicious events from simulated IoT network attack scenarios, as well as the deployment setup to be carried out.

The principal objective of this deliverable is to present the design and implementation of the Federated Threat Analysis component that will be integrated into the ERATOSTHENES architecture and to set the direction for future extensions of the component to meet all the pre-defined functionalities.
2 Introduction

This chapter maps the work done in task 4.4 named "Federated threat analysis models for continuous assessment" with each of the chapters and sections that compose the deliverable. Furthermore, its structure is briefly described.

2.1 Mapping ERATOSTHENES Outputs

The purpose of this section is to map ERATOSTHENES Grand Agreement commitments, both within the formal Deliverable and Task description, against the project’s respective outputs and work performed.

Table 1: Adherence to ERATOSTHENES GA Deliverable & Tasks Descriptions

<table>
<thead>
<tr>
<th>ERATOSTHENES GA Component Title</th>
<th>ERATOSTHENES GA Component Outline</th>
<th>Respective Document Chapter(s)</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELIVERABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4.4 Federated threat analysis models for continuous risk assessment</td>
<td>This deliverable includes the initial outcomes from Task 4.4 about the development of Federated threat analysis models for continuous assessment as the defence service for detecting different attacks and exploits inside IoT networks</td>
<td>Section 3.1 and Section 4.1</td>
<td>Section 3.1 defines the architecture of the federated system and explains each of its components. Section 4.1 describes the FedLPy package developed.</td>
</tr>
<tr>
<td>TASKS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4.4 Federated threat analysis models for continuous assessment</td>
<td>This task will create a defence service capable of detecting different attacks and exploits inside an IoT network using an AI model that will be developed. For a first prototype, the model will be created based on current existing datasets about traffic network and IoT attacks: (a) the UNSW-NB15 Dataset (recently updated, 2018) with about 2 million registers and 9 different types of attacks; (b) the Aposemat IoT-23 (from 2020) with 20GB of network traffic and 20 different malware detection. A similar approach, like the one followed by these two datasets, could be used to capture extra data sets from pilots. TCPDUMP to get network traffic and tools</td>
<td>Section 4.2, Section 4.3, Chapter 5</td>
<td>Section 4.2 defines the implementation of a federated server/client based on the FedLPy package. Section 4.3 describes the deployment setup that will be followed for the deployment in a regular IoT infrastructure. Chapter 5 provides a description of the state-of-the-art public datasets used for training and evaluating the first models.</td>
</tr>
</tbody>
</table>
such as Argus and Zeek to analyze the traffic. A particularity for this model is to make it capable of running on the edge, making usage of extra computing from special devices to enhance AI tasks: TPU, GPUs, etc. The training process would be implemented with two alternative approaches: training in the cloud and running in the edge (with frameworks like TensorFlow-Lite), or a more advanced option based on federated learning. Using federated learning the IoT network, running the model, sharing results with a central node, using the feedback of the network to provide new model's improved versions. The data shared between the nodes are just model's metadata and never traffic data. This is usually an important point to prevent data privacy. Finally, in order to provide extra security and auditable models, data sets and data models will be incorporated into the Blockchain. In case of an incident, the model owner would be audited about the version of the model and the data set was used for the training. This auditing service is part of the requests from the EC about AI and Trust. This integration will be developed jointly with task 4.3 and the results will be reported in D4.4, in D4.7 (updated version) and in the final version D4.8.
2.2 Deliverable Overview and Report Structure

Deliverable D4.4 presents the initial version of the ERATOSTHENES Federated Machine Learning (FML) Network Threat Analysis package implemented. This framework is called *FedLP*, and it is the main output of Tasks 4.4. This document is organized as follows:

- Chapter 3 presents the architecture of the Federated Threat Analysis components. Section 3.1 presents an overall view of the federated system architecture and a flow diagram for the training and inference processes. Furthermore, Section 3.2 discusses the compliance of the developed solution to the requirements defined in D1.2. Finally, Section 3.3 explains the methodology and rationale behind the design decisions made during the development of the component.

- In Chapter 4, more details on the actual implementation of the component are provided, and how the implemented solution technically supports the requirements. Section 4.1, describes each of the *FedLP* modules (federated learning, differential privacy, common). Section 4.2 explains the core of the federated learning system, where the communication logic between the server and the clients is defined, as well as the details of the training following this approach and the first results achieved. Finally, the deployment process of the federated learning system is presented in section 4.3.

- Chapter 5 delves into the initial experiments conducted throughout the development of *FedLP*. The chapter begins with Section 5.1, which offers insights into the state-of-the-art databases used and the pre-processing steps taken to adapt them to the AI models employed. Then, Section 5.2 provides comprehensive information about the models implemented. Lastly, Section 5.3 presents and discusses the results obtained.

- Finally, Chapter 6 presents the main conclusions extracted from this work and the potential future actions.
3 Architecture Orientation and Industrial Requirements

This chapter describes the location of the *FedLPy* component in the ERATOSTHENES architecture and schematizes the package construction, detailing the main modules, business and industrial positioning, end-user requirements and the methodology used to develop the proposal.

3.1 Architectural Positioning and design decisions

The Federated Threat Analysis block of ERATOSTHENES provides Artificial Intelligence (AI) to detect potential attacks within the IoT network. Applying a federated approach implies (i) a federated learning process of the ML detection algorithms and (ii) an inference process, where the IoT edge devices perform the detection in their local flow. This approach implies multiple advantages in comparison with traditional centralized machine learning, where data from the IoT devices are sent to a server where detection is made. These advantages are:

1. **Scalability**: Data decentralization in FML and on-device inference allows harnessing the computing power of numerous devices. In this way, FML can train complex models with large-scale datasets. Additionally, on-device inference allows parallelizing the inference process and prevents the server from behaving as a bottleneck when there is a high inference load.
2. **Efficiency**: FML eliminates the need to transfer raw data from devices to a central server. This reduces the bandwidth and processing requirements of the server.
3. **Privacy and security**: Data decentralization in FML enables collaboration and learning from distributed datasets without compromising private information as the sensitive information remains on local devices. In this way, FML reduces the risk of data leakage and unauthorized access.

Given the current ERATOSTHENES architecture represented in Figure 1, the Federated Threat Analysis block is closely related to the Monitoring, Intrusion Detection System (IDS) module located in the Trust Manager & Broker (TMB). For this reason, the Server-side of the FML approach is expected to be deployed in the TMB, while the client-side will be deployed in the IoT devices of the network.

![Hyperledger Fabric Channel (Multi-domain)](image)

**Figure 1**: Current ERATOSTHENES architecture.

This section describes the general architecture of the *FedLPy* and discusses how this block is interconnected with the ERATOSTHENES tools.
3.1.1 Overall Architecture of FedLPy

The Federated Threat Analysis Block is synthesized in the FedLPy component. From a high-level point of view, FedLPy can be divided into a server-side and edge node-side, as shown in Figure 2, where the server-node communication is only used to send the model weights during the federated training phase. In addition, each edge node communicates with the Trust Manager & Broker (TMB) of the ERATOSTHENES network so that these IoT devices can alert the system that they may be under attack during the inference phase.

![Figure 2: ERATOSTHENES federated learning training and continual inference overall architecture.](image-url)

3.1.1.1 Training Phase

In the training phase of the FedLPy component, both the server side and the edge node side are important. Each of these blocks is composed of several layers of functionality.

On the side of the edge nodes, three different functional layers are distinguished:

1. **Model Training Layer**: Here are all the necessary functions for training machine learning models. This layer oversees loading the last weights of the model stored in the device (coming from the server), which are adjusted with the local data of the device. Note that this data can be static (data never changes) or dynamic (data can be updated over time).

2. **Differential Privacy Layer**: This layer includes the necessary techniques to ensure that the weights sent to and received from the server are protected so that a malicious agent cannot extract the training data from the different edge devices. Those techniques can be Differential Privacy Stochastic Gradient Descent (DP-SGD) [1] or Private Aggregation of Teacher Ensembles (PATE) [2] among others.

3. **Federated Learning Client Layer**: It is in charge of establishing communication with the Federated Learning Server, sending the model weights of each device, receiving the aggregated weights, and storing them locally.

On the server side, there are also two layers:

1. **Federated Learning Server Layer**: This layer is responsible for receiving each of the weights of the models generated by each of the IoT devices and passing them on to the next layer.

2. **Model Aggregation Layer**: This layer is responsible for, from all the weights collected from the different edge nodes, generating an overall model considering the insights of each of these models.
3.1.1.2 Inference Phase

The inference phase of the FedLPy component is only performed on the edge node side. This block consists of the following two layers:

1. **Continual Inference Layer:** This layer is responsible for estimating sample by sample the probability of each type of attack that the IoT device is suffering. This process continued over time and only stopped to update the global model weights.

2. **Web Report Layer:** Currently, a report of events that occurred during runtime is displayed. However, it is intended that the device send an alert to the TMB to notify the attack and that this component acts accordingly.

3.1.2 Training-Inference Flow Diagram

Since one of the most important functionalities of task 4.4 of the ERATOSTHENES project is to develop a system based on federated learning techniques, this subsection details the communication flow between the server and the different clients.

Figure 3 shows an example of a connection of three clients with the server. It should be noted that these connections can be extended to as many clients as desired. The federated training scheme is divided into four different parts:

1. **Model Distribution:** The server sends the latest version of the global model to each of the clients that will take part in the training. The server will remain in idle mode until it receives the updated weights from each of the clients.

2. **Local Training:** Each of the clients performs a classic training (fit and val processes) with their data and the weights of the received model. This process can be repeated as many times as desired.

3. **Aggregation Process:** The clients send the parameters resulting from the training process to the server and the server aggregates them in such a way that it computes a global model that considers the insights acquired from each of the nodes.

4. **Evaluation of the Aggregated Model:** The server sends the resulting model back to the clients so that they can evaluate the accuracy of the model with their private data. This result is returned to the server by each of the nodes, where they are weighted aggregated.
3.1.3 Interconnection with ERATOSTHENES tools

Regarding how the FedLPy component will interact with the Monitoring IDS module. Three approaches have been studied:

1. **The Anomaly Detection Inspector provides the initial FedLPy model:** The ADI’s model developed in task 4.4 and described in Deliverable 4.5 will provide the initial model of the FedLPy component. In the learning phase, this model is sent to the clients that update it with their own local data. In the inference phase, the final aggregated model is used on each device to detect threats in their local traffic flows. When a threat is detected, the FedLPy client will raise an alert to the IDS.

2. **The Anomaly Detection Inspector as a FedLPy client:** The ADI submodule will be considered as a FedLPy client. Thus, in the learning phase, the ADI’s model will be aggregated with the rest of the local models. In the inference phase, the ADI will detect threats in the network traffic, while the rest of the FedLPy clients will detect threats in their traffic. When a threat is detected, either by the ADI or the rest of the FedLPy clients, an alert will be sent to the IDS Threat/Rule Manager.
3. **The Anomaly Detection Inspector is independent of FedLPy**: This approach would imply two network scans, one in the edges and one in the TMB from the ADI developed in task 4.5. Both components would send independent alerts to the Monitoring IDS Threat sub-module.

These approaches have been schematized and represented in Figure 4. Note that for the sake of simplicity, edges from the clients to the server corresponding to the local parameters sharing have been omitted. To benefit from the main advantages of the Federated Threat Analysis Scheme, it has been decided that the first approach is the most efficient. However, the other options are maintained as alternatives for future study.

![Figure 4: Different approaches to interconnect ADI sub-module and FedLPy. (a) ADI as the initial FedLPy model (b) ADI as a FedLPy client. (c) ADI independent of FedLPy.](image-url)
3.2 Business, Industrial Positioning and End-User Requirements

The development of communication networks has evolved significantly in recent years. The latest trends are to implement networks of IoT devices interconnected with each other creating complex grids with many elements sending information over the network. Nevertheless, due to their simplicity and low computational resources, these elements are the most vulnerable within a network and are the principal targets of those agents who attempt to attack the system. Thus, monitoring the network traffic flows of these devices should be done to assess possible malicious risk events.

According to NIST [3], continuous risk monitoring is defined as “maintaining ongoing awareness of vulnerabilities and threats to support organizational risk management decisions”. The main reasons for implementing a continuous risk assessment system in a network are: (i) maintaining situational awareness of all devices throughout the network and (ii) maintaining an understanding of the threats and then acting accordingly.

“Continuous security monitoring is a type of security solution that automates security monitoring across multiple devices” [4, 5]. This type of monitoring enables continuous assessment of the network to stay ahead of cyber threats by providing real-time visibility of devices operating on the network. To address the task of continuous risk assessment, several approaches exist. However, systems based on artificial intelligence, and more specifically machine learning, are one of the most widely used today.

The ERATOSTHENES project brings a continuous threat monitoring system based on machine learning, which exploits the topology of IoT networks to implement federated learning techniques [6, 7]. In this way, the privacy of the data of each device (network traffic flows in this case) is ensured, allowing all of them to share the same threat detection model without having to share the data packets used to fit it.

From D1.2 "Use Cases, Requirements and Methodological Framework", the main objective of this task is the need to implement a system based on machine learning capable of allowing connected devices to analyse the received network traffic to detect attacks and report them, allowing constant monitoring of the network at the device level. Table 2 shows the end-user requirements related to this objective.

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Description</th>
<th>Importance</th>
<th>Rationale</th>
<th>Component's Fulfilment of the Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1_FR_05</td>
<td>The infrastructure is monitored, and network traffic is analysed to detect intrusions</td>
<td>M</td>
<td>The integrity of the communications between the vehicles and the infrastructure is ensured by some intrusion/anomaly detection</td>
<td>FedLPy continual risk assessment module from edge devices (vehicles and/or infrastructure) by using IDS models.</td>
</tr>
<tr>
<td>P1_FR_07</td>
<td>The devices must be able to detect an attack and report it</td>
<td>S</td>
<td>Devices can detect attacks so that appropriate action can be taken.</td>
<td>FedLPy continual risk assessment module from edge devices (vehicles and/or infrastructure) by using IDS models.</td>
</tr>
<tr>
<td>P1_NFR_02</td>
<td>Improve detection of malicious messaging from compromised/unsecured actors</td>
<td>M</td>
<td>Improved detection of not identified, suspicious or considered not</td>
<td>FedLPy package and its Federated Learning Training implementation</td>
</tr>
</tbody>
</table>
### 3.3 Methodology

In task 4.4, the development strategy followed is based on the Agile and Minimum Viable Product (MVP) principles, in such a way that interaction between the involved partners and the continuous development of the product are encouraged.

Starting from the architecture and end-user requirements defined in the early stages of the project, the functionalities of the *FedLPy* component are defined. Additionally, simple PoCs are implemented to demonstrate to the involved partners.

To date, the *FedLPy* component is being developed to include all the required functionalities, as well as working together with pilots to refine the different improvements of the module. The pilots have been asked for a dataset with data captured from the use cases to fit the component to the specific needs of these scenarios.

It is important to mention that, since the details of the pilot infrastructure specifications are currently being defined, as well as the technical specifications and the user stories, this document reports the analyses made with the available information from the pilots. Since user stories are defined from a high-level point of view, without knowledge of what kind of attacks are expected to happen or what type of dataframes will be sent through the network, these forced us to develop the presented work by using open databases to define and train the machine learning models proposed in the context of the ERATOSTHENES projects instead of data collected from Pilot’s real-world environments. This lack of specificity in the model definition may lead to poor results in the final demonstrations.
4 Federated Learning Framework for Threat Analysis

This chapter presents the component developed for task 4.4 of the ERATOSTHENES, known as *FedLPy*. With this solution, it is possible to implement an FML system allowing its scalability, since it has followed a modular approach. It should be noted that the *FedLPy* component is developed in *Python* while the system deployment is done using the *Docker* engine.

4.1 The *FedLPy* package

This section provides an overview of the different modules that constitute the *FedLPy* package and offers a wide description of their distinct functionalities. As shown in Figure 5, the *FedLPy* package consists of three closely related modules: the Federated Learning module, the Differential Privacy module, and the Common Functions module.

![FedLPy Structure](image)

Figure 5: *FedLPy* Structure

The following subsections offer a more detailed description of these modules, with an explanation of the main functions that allow them to work properly.

4.1.1 Federated Learning Module

The Federated Learning module is the core unit of *FedLPy* as it encapsulates all the functions required to instantiate and handle the communication between the *FedLPy* clients and the *FedLPy* server explained in section 3.1. This module makes use of the python package *Flwr* [8] to instantiate and handle the communication between the federated learning agents, *FedLPy* Differential Privacy (section 4.1.2) module to apply Privacy Preserving Machine Learning algorithms, and Common module (section 4.1.3) to locally train the clients’ models. *Flwr* is an open-source Python package developed for Federated Learning that offers a friendly framework characterized by: (i) **Efficiency**, it establishes an efficient communication protocol between server and clients based on gRPC; (ii) **Extensibility**, the modular structure of the package allows to extend it generating new Aggregation techniques and Federated Learning agents; (iii) **Flexibility**, it can be deployed in real and heterogeneous devices; (iv) **Compatibility** with state-of-the-art ML frameworks, such as TensorFlow or PyTorch; and (v) **Scalability**, as it can handle a large number of concurrent clients.

This module oversees instantiating the federated agents described in section 3.1.1:

1. **FedLPy Server**: this agent orchestrates the federated training process by the definition of a **training strategy** and an **aggregation technique**. Additionally, it audits the performance of the global model with a public dataset located in its internal storage. Both, the training strategy, and the aggregation technique are described in sections 4.1.1.1 and 4.1.1.2 respectively.

2. **FedLPy Client**: this agent performs the local training of the AI model, shares the AI updates with the server, and updates its local model with the aggregated model from the Server.

Additionally, the Federated Learning module offers some user-friendly functionalities to instantiate both agents:

- *Fl_server_training*: Given an evaluation dataset, an initial model and the configuration file parameters related to the server settings, this function instantiates a *FedLPy*.

- *Fl_client_training*: Given the Ip address and the port number of the Server, the initial ML model, the local dataset, and the privacy-preserving parameters of the configuration, this function instantiates a *FedLPy* Client in a device.
4.1.1.1 Training Strategy

The *FedLPy* server specifies the values of some configuration parameters that define the operation of the complete training process. These parameters can be divided into:

- **Local training configuration.** This category includes all the parameters that the *FedLPy* server sends to the *FedLPy* clients related to the local training. Among these parameters we find the *batch size* (32 by default) and the number of *local epochs* (10 by default) for the local training.

- **Federated Network configuration.** This category includes all the parameters related to the configuration of the federated learning network. The main parameters are:
  
  - *Number of rounds*: Number of times all the steps of the federated learning server-client communication (see diagram in figure 2) are performed to train the final model. By default, this parameter is set to 10.
  - *Min fit clients*: Minimum number of clients that reached the server to start the training procedure. By default, this parameter is set two 2.
  - *Fraction fit*: Percentage of clients that are going to be used to train the model in a Federated way. By default, this parameter is set to 100%.
  - *Min evaluate clients*: Minimum number of clients that reached the
  - *Fraction evaluate*: Percentage of clients that are going to be used to evaluate the final model. By default, this parameter is set to 100%.

In the current version of *FedLPy*, these parameters are fixed with the default values. These default values are specified to accommodate the Federated Learning approach to the needs of ERATOSTHENES pilot 1. However, in future interactions, these parameters may be modified directly from configuration files (see section 4.2.3) in other to fit other pilot’s scenarios.

4.1.1.2 Aggregation technique

The aggregation technique defined by the Server determines how the local model updates are aggregated to conform to the global model that is sent to the clients. In the current version of *FedLPy*, the server applies the *Federated Averaging* [9] or *FedAvg* technique which is one of the most used in the literature [10, 11].

\[
W^t ← \frac{1}{m} \sum_{i=1}^{m} \frac{n_i}{n} W_i^t
\]  

(1)

As it is described in (1), *FedAvg* generates the global model in a specific round \( (W^t) \) by computing the weighted average of the parameters \( (W_i) \) of the local models. The weights are computed as the number of data samples in each client \( (n_i) \), divided by the total number of training samples \( (n) \). However, the *flwr* package defines a large number of aggregation strategies, so in future iterations, an analysis of the best-performing strategy under the ERATOSTHENES’ scope will be performed.

4.1.2 Differential Privacy Module

The implementation of Federated Learning represents an improvement in privacy compared to the traditional centralized machine learning scheme because clients never share their private data. However, this approach is still susceptible to different types of attacks, such as those aiming to extract information from private data from the model parameters. Privacy-Preserving Machine Learning (PPML) emerges to prevent such information leaks and maintain client privacy at any stage of the machine learning pipeline. Among these algorithms, differentially private algorithms have gained significant relevance in research [12]. Formally, a differentially private algorithm aims to guarantee that the output of a given algorithm remains almost indistinguishable, regardless of whether a data sample is extracted or included from the database. This property provides privacy guarantees robust against different types of attacks, including sophisticated statistical inference techniques. The central idea of differential privacy is to introduce a controlled amount of noise or randomness to difficult the extraction of features from the training sensitive training database. It is important to remark that this noise needs to be carefully calibrated to strike a balance between privacy protection and accuracy in the results.
The differential privacy module includes two PPML algorithms:

1. **Differentially Private Stochastic Gradient Descent (DP-SGD)** [1] aims to provide privacy guarantees while optimizing the model parameters using gradient-based optimization. To do this, DP-SGD modifies the traditional SGD algorithm including two intermediate steps: Gradient clipping and noise addition. Thus, the final algorithm consists of the following stages: (i) Draw a minibatch from the training dataset. Then, for each value in the minibatch: (ii) gradients of the loss function are computed and (iii) clipped with L2 Normalization. Finally, (iv) the resulting gradients are aggregated and added to a Gaussian noise with a specified magnitude to guarantee differential privacy. The parameters of the model are updated with the gradients as traditional SGD.

The differential privacy module of *FedLPy* allows training a TensorFlow Deep Learning model with the function `train_dl_dpsgd`. This function receives the initial model, the training dataset, and the main parameters of the private learning. Among these parameters, we highlight:

- **Microbatch size**: Number of samples per micro-batch.
- **Noise multiplier**: determines the amount of random noise that is added to the gradients.
- **L2 norm clip**: limits the sensitivity of the gradients.

These parameters determine the existent trade-off between privacy and the utility of the trained model. This function uses the TensorFlow Privacy [13] package to implement this optimizer. TensorFlow Privacy is an open-source Python package, developed by Google Research, that includes differentially private versions of traditional machine learning optimizers.

2. **Private Aggregation of Teacher Ensembles (PATE)** [2] is a framework to address the challenge of privacy-preserving machine learning very used in scenarios where data is distributed. PATE offers a solution based on the definition of three components: teachers’ models, aggregation technique, and student. The steps of the PATE algorithm are: (i) teacher models are trained with disjoint portions of sensitive data and generate a so-called teacher ensemble (ii) predictions over an unlabelled public database are made by the teachers of the ensemble (iii) votes/predictions of the teachers are aggregated into a single result using a noisy aggregation technique (iv) final prediction of the whole ensemble are considered as the labels of the unlabelled dataset (v) the student model is trained with the complete dataset. In this way, the final student model trained is public. A flow diagram representing this process is shown in Figure 6.

![Figure 6: Private Aggregation of Teachers ensembles schematized. This image is based on [2].](image-url)
parameters of PATE and returns the student public model. It is important to remark that FedLPy includes only the LNMax aggregation technique, however, in future versions, we will study the possibility of including other approaches. LNMax aggregation technique counts the votes of each teacher and adds calibrated Laplacian noise to the vote histogram. Finally, it selects the label with the most votes as the ensemble’s prediction [2]. Thus, the parameters final parameters that determine the operation of the PATE algorithm are the number of teachers used for training and the noise multiplier. In the same way as DP-SGD, the noise multiplier determines the amount of random noise added to the labels in the LNMax algorithm and determines the trade-off between the usability and privacy of the approach. In the current version of FedLPy, both parameters can be modified by the user using configuration files (section 4.2.3).

4.1.3 Common module

Finally, the common module of FedLPy includes all the cross-cutting functionalities of the package. It is important to remark that this is the module most closely related to the ML pipeline. For this reason, this module is expected to suffer continual changes during the testing process. The functions of this module can be divided into 3 blocks:

1. **Model definitions**: The script `FedLPy.common.models.py` will compile the definition of the different Deep Learning models tested in the process of development of FedLPy and ERATOSTHENES use cases.
2. **Training process**: The script `FedLPy.common.models.py` compiles functionalities to build, fit and evaluate a TensorFlow model given a training dataset and a model definition. The current version of FedLPy only contains a parametrized version of the Deep Learning model described in Section 5.2.
3. **Databases**: The script `FedLPy.common.databases.py` will encapsulate all the functions related to data loading and data treatment needed throughout ERATOSTHENES use cases and pilots. In the current version of FedLPy, we only find the dataset used to achieve the preliminary results. A wide description of this database and its respective pre-processing procedure is described in Section 5.1.

4.2 FedLPy-based Server & Client

After describing the main capabilities of the FedLPy tool, these sections describe how this package is used to implement and deploy a Federated Learning scheme. In this section, we will define all the software components that must be included in a device to finally deploy a FedLPy Client or FedLPy Server on it. Then, section 4.3 focuses on the deployment process itself using Docker.

4.2.1 Server

In order to define the Server, the selected device must be provided with the path structure defined below. In the context of ERATOSTHENES, this process will be implemented in the Trust Manager & Broker.

```
/Server/
    FedLPy/
    model/
    storage/
    config.ini
    server.py
```

Where:

- **The FedLPy directory** contains the complete FedLPy toolbox described in the previous sections or the wheels file generated for the installation of the package.
- The **Model folder** holds the parameters of the initial Deep Learning parameters of the model selected for detecting attacks in the IoT network. Note that this model will be trained federally and will be used on devices.
- **Storage directory** keeps the local database of the device. As it was mentioned before, currently FedLPy only handles the public database described in section 5. However, in future releases, it is expected to support Pilot databases.
- **Config.ini file** is the server configuration file, which allows the user to define in a straightforward way the parameters that determine the behaviour of the federated learning approach. A description of this configuration file is provided in section 4.2.3.

- Finally, the `server.py` is a Python script developed to act as the main file. This script makes use of FedLPy functionalities to perform initialize a FedLPy server and initialize the federated learning process. First, this file reads the configuration file (config.ini) and extracts the values of the parameters defined by the user. Then, it instantiates and loads the ML model to be trained in a federated way. Finally, it automatically schedules the initialization of the FedLPy server that will wait for a minimum number of clients to connect. By default, the `script.py` schedules a federated re-training every hour. However, this can be adjusted to the use case needs.

### 4.2.2 Client

In contrast with the server, the ERATOSTHENES IoT devices will act as federated clients that will train the models and perform threat detection in their local traffic. For this reason, the client functionality is divided into two parts. On the one hand, the Client Backend oversees performing the training phase. As we have seen in Figure 3, this process includes training and sending the model updates to the server and updating the local model with the aggregated model received. On the other hand, the Client front end performs the inference phase, which includes the detection of anomalies in the local traffic and sending alerts to the ERATOSTHENES Threat Manager of the IDS module located inside the Trust Manager Broker. Thus, to define a federated client, the device in question must have the following structure:

```
/Client/
    └── Backend/
          └── model/
                  ├── client_backend.py
                  └── config.ini

    └── Frontend/
          └── model/
                  └── client_frontend.py
          └── templates/index.html
```

In the client Backend, we find the following components:

- **Config.ini** is the configuration file required to set the specific FedLPy parameters for the client. A description of this configuration file is provided in section 4.2.3. Specifically, the client backend module will focus on Database used, server IP address and network Port, and the client id.

- **The Model folder** contains the parameters of the initial Deep Learning parameters of the model selected for detecting attacks in the IoT network. Note that this model will be trained federally and will be used on devices.

- **Client_backend.py** is the Python script developed to act as the main client Back-End file. Thus, with this script, the client device first reads the FedLPy configuration file to set the federated learning parameters. Then, the initial model is loaded and defined, before being trained locally. Finally, the script automatically schedules the initialization of the federated client. It is important to note that the client will try to communicate with the FedLPy Server, which must be previously initialized. For this reason, the client and server scheduling must be correctly synchronized.

On the other hand, the client Front-End is composed of the following components:

- **Config.ini** is the configuration file required to set the specific FedLPy parameters for the client. A description of this configuration file is provided in section 4.2.3. Specifically, the client frontend focuses on Database used, web publishing IP address and network Port, and client id.

- **The Model folder** contains the parameters of the Deep Learning model selected for detecting attacks in the IoT network resulting from the Federated Learning Training process.
- **Client\_frontend.py** is the Python script developed in order to perform all the functionalities defined to the web reporting block of client devices.
- **The Templates folder** contains the HTML file with the template of the web page used for the reporting process.

Figure 7 shows the most updated version of the web reporting page. It can be seen on it that a table with the accuracy and true positives and true negative values as well as the threat name and the number of samples analyzed. Also, the confusion matrix of the obtained results is shown (Note that this visualization frame has been captured during an early stage of the model learning and it does not show actual results. Its only purpose is to show the frontend web layout used in the model's continual assessment phase).

### Results for Client 2

<table>
<thead>
<tr>
<th>Threat</th>
<th>Acc (%)</th>
<th>Tp (%)</th>
<th>Tn (%)</th>
<th>AUC</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSE</td>
<td>78.79</td>
<td>0.11</td>
<td>100.0</td>
<td></td>
<td>2027</td>
</tr>
<tr>
<td>UC</td>
<td>93.34</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td>112</td>
</tr>
<tr>
<td>MSE</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>PSE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8</td>
</tr>
<tr>
<td>MSE</td>
<td>78.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td>2048</td>
</tr>
<tr>
<td>ROC</td>
<td>93.34</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>AUC</td>
<td>90.31</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>MSE</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td>1070</td>
</tr>
<tr>
<td>Attacker</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSE</td>
<td>78.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSE</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSE</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSE</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSE</td>
<td>70.32</td>
<td>0.22</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Figure 7: Front End initial version included in FedLPy](image)

### 4.2.3 Configuration Files

*FedLPy* users can modify parameters that define the Server and Client nodes' operation, throughout configuration files. For this reason, a proper definition and knowledge of these files is necessary for a successful deployment of the federated learning architecture.

An example of an *ERATOSTHENES* Federated Learning configuration file is as:

```plaintext
[SERVER]
NumberOfRounds = 1
IpAddress = localhost:8080

[CLIENT]
Id = 2
FrontEndPORT = 8090

[PRIVACY]
Algorithm = DPSGD
L2NormClip = 1.0
NoiseMultiplier = 1.3
LearningRate = 0.01
BatchSize = 64
```
MicroBatches = 16

[GLOBAL]
model = DENSE_CLASSIFIER
dataset_path = storage/IoT23/
DataSet = IoT23

As can be seen above, deployment configuration files have four main sections:

1. **Server Configuration**: This section includes the server configuration parameters, and it is tagged as [SERVER]. These parameters are:
   a. *NumberOfRounds*: Defines the number of rounds that a federated learning will perform.
   b. *IpAddress*: Server IP address and port where clients should connect.

2. **Client Configuration**: Configuration parameters are under the [CLIENT] label and they are:
   a. *Id*: The Client identifier.
   b. *FrontEndPORT*: Port where users can access HTTP web reporting.

3. **Privacy Configuration**: All configuration parameters regarding the privacy-preserving techniques are defined after the [PRIVACY] tag. The key parameter is the *Algorithm*, which determines whether the algorithm used is DP-SGD or PATE. The configuration file will present different parameters depending on the algorithm used for enhancing the privacy of the ML models. If the algorithm is DP-SGD then the main parameters are *L2NormClip*, *NoiseMultiplier* and *MicroBatches*: which correspond with the parameters described in section 4.1.2. In contrast, if the algorithm selected is the PATE, the main parameters are the *NoiseMultiplier* and *NTeachers*: which correspond to the main parameters described previously.

4. **Global Configuration**: This configuration section collects all the parameters that will be used by both components, Server and Client.
   a. *model*: A model identifier which the system will be trained with.
   b. *dataset_path*: Relative or absolute path from which data will be extracted from.
   c. *DataSet*: Dataset identifier that is used for the federated learning training.

Next *FedLPy* versions will explore the inclusion of new parameters needed to provide a higher configurable user experience when deploying a federated learning system, as well as refactoring some of the already implemented parameters.

### 4.3 Deployment

#### 4.3.1 Deployment Script

As the *FedLPy* component must be instantiated in several components, which must communicate with each other and with the ERATOSTHENES network (see Figure 2), the deployment process will be automated using the Docker-Compose tool. Furthermore, Docker images are generated with all the necessary functionalities for the Client and the Server components. This approach allows users to better deploy and scale federated learning services in the context of the *ERATOSTHENES* context.

#### 4.3.1.1 Dockerfiles

Each existing module within the architecture defined for Task 4.4 is encapsulated into a Docker image based on an Ubuntu Focal OS base image. Therefore, three different images are generated:

1. **FML server image**
   - This container runs the server.py script where schedules a *FedLPy* Server component according to the configuration provided. The Dockerfile used for this image generation is:
     ```
     FROM ubuntu:focal
     LABEL maintainer="Juan Manuel Vera Diaz" contact="juan.vera@atos.net"
     name="Federated Learning Client Frontend Container"
     WORKDIR /usr/app
     COPY . .
     RUN mkdir model && \
         mkdir storage && \
     ```
apt-get update &&
apt-get install tree &&
apt-get install -y software-properties-common &&
add-apt-repository ppa:deadsnakes/ppa &&
apt-get update &&
apt-get install -y python3-pip &&
pip3 install pip==22.3.1 &&
python3 -V &&
pip3 -V &&
pip3 install FedLPy-0.2.1-py3-none-any.whl &&
rm -rf FedLPy-0.2.1-py3-none-any.whl &&
pip3 install protobuf==3.19.5 &&
pip3 install schedule &&
tree ./

CMD ["python3", "server.py"]

The working directory tree within the Docker container is structured as:

```
/usr/app/
  ├── model/
  │   └── storage/ 
  │       ├── config.ini     
  │       └── server.py
  └── server.py
```

We build the images in a way they can be used whether in AMD or ARM 64 bits architectures. The commands used for creating these images are:

```
docker build --force-rm -t <docker-images-repo>/server/amd64:v0.2.1 --platform linux/arm64 -f server.
docker build --force-rm -t <docker-images-repo>/server/arm64:v0.2.1 --platform linux/arm64 -f server.
```

2. **FML client image**

client_backend.py script is run within this container where the FedLPy Client component is implemented and synchronously scheduled with Server. The Docker file used for this image generation is the following:

```
FROM ubuntu:focal
LABEL maintainer="Juan Manuel Vera Diaz" contact="juan.vera@atos.net"
name="Federated Learning Client Backend Container"
WORKDIR /usr/app
COPY . .
RUN mkdir model &&
mkdir storage &&
apt-get update &&
apt-get install tree &&
apt-get install -y software-properties-common &&
add-apt-repository ppa:deadsnakes/ppa &&
apt-get update &&
apt-get install -y python3-pip &&
pip3 install pip==22.3.1 &&
python3 --version &&
pip3 --version &&
pip3 install FedLPy-0.2.1-py3-none-any.whl &&
rm -rf FedLPy-0.2.1-py3-none-any.whl &&
pip3 install protobuf==3.19.5 &&
pip3 install schedule &&
tree ./
```
CMD ["python3", "client_backend.py"]

The working directory tree within the Docker container is structured as:

```
/usr/app/
    ├── model/
    ├── storage/
    │   └── config.ini
    └── client_backend.py
```

It is important to note that the model folder will be a binding directory from the edge device in order to share the model between the FML module and the inference module.

We build the images in a way they can be used whether in AMD or ARM 64 bits architectures. The commands used for creating these images are:

```
docker build --force-rm -t <docker-images-repo>/client/backend/amd64:v0.2.1 --platform linux/amd64 -f client_backend .
docker build --force-rm -t <docker-images-repo>/client/backend/arm64:v0.2.1 --platform linux/arm64 -f client_backend .
```

3. Inference client image

This container includes the functionalities for the continual network risk assessment based on the Client FedLPy component. At this point, this container prompts a web summary of the accuracy results obtained by the global model according to the storage of local data within the edge node. The Docker file used for this image generation is the following.

```
FROM ubuntu:focal
LABEL maintainer="Juan Manuel Vera Diaz" contact="juan.vera@atos.net"
name="Federated Learning Client Frontend Container"
WORKDIR /usr/app
COPY . .
RUN mkdir model && \
    mkdir storage && \
    apt-get update && \
    apt-get install tree && \
    apt-get install -y software-properties-common && \
    add-apt-repository ppa:deadsnakes/ppa && \
    apt-get update && \
    apt-get install -y python3-pip && \
    pip3 install pip==22.3.1 && \
    python3 --version && \
    pip3 --version && \
    pip3 install FedLPy-0.2.1-py3-none-any.whl && \
    rm -rf FedLPy-0.2.1-py3-none-any.whl && \
    pip3 install protobuf==3.19.5 && \
    pip3 install dash && \
    tree ./

CMD ["python3", "client_frontend.py"]
```

The working directory tree within the Docker container is structured as:

```
/usr/app/
    ├── model/
    ├── storage/
    └── templates/
```
Note that the model folder will be shared with the FML docker model folder to allow the inference module to obtain the last aggregated model served from the FML Server module to the FML Client module. In this way, a continual network risk assessment can be implemented.

We build the images in a way they can be used whether in AMD or ARM 64 bits architectures. The commands used for creating these images are:

```sh
docker build --force-rm -t <docker-images-repo>/client/frontend/amd64:v0.2.1 --platform linux/amd64 -f client_frontend.
docker build --force-rm -t <docker-images-repo>/client/frontend/arm64:v0.2.1 --platform linux/arm64 -f client_frontend.
```

### 4.3.1.2 Docker Compose Files

Since configuration files must be copied in and some local folders must also be mounted within the Docker containers, allowing sharing of model files between them, `.yml` files (server and client) are generated to be able to automatically deploy the server component and the client component individually, enabling all the necessary functionalities according to the use case.

1. **Server docker-compose file.**
   - This docker-compose file automatizes the deployment of the server node in such a way it mounts the necessary folders and files within the server docker image and opens the network's ports that will be used and run the container. This file is as follows:

   ```yaml
   version: "3.9"
   services:
     server:
       image: registry.atosresearch.eu:18494/server/amd64:v0.2.1
       volumes:
       - ./models:/usr/app/model
       - ./config/config.ini:/usr/app/config.ini
       - ./storage:/usr/app/storage
       container_name: server
       ports:
       - 8080:8080
   
   The local folder structure that is needed in the Server device is the following:

   ```
   ./
   ├── model/
   │   └── client.ini
   ├── storage/
   │   └── server-compose.yml
   └── config/
   ```

   - The model folder contains the model aggregation results that will be served to edge nodes. The storage folder contains static data used for the evaluation of the aggregated model and the config folder contains the configuration file used for the Server component deployment.

   - The command used to deploy this component is:

   ```sh
docker compose -f server-compose.yml up
```

2. **Client docker-compose file.**
Both, the inference and the FML modules are deployed by using the same docker compose file in the edge nodes. This file will mount necessary folders and files, open networks ports and finally run the containers as follows:

```
version: "3.9"

services:  
  client_backend:  
    image: registry.atosresearch.eu:18494/client/backend/arm64:v0.1.1  
    volumes:  
      - ./models:/usr/app/model  
      - ./config/config.ini:/usr/app/config.ini  
      - ./storage:/usr/app/storage  
    network_mode: "host"  
    container_name: client_backend
  
  client_frontend:  
    image: registry.atosresearch.eu:18494/client/frontend/arm64:v0.1.1  
    volumes:  
      - ./models:/usr/app/model  
      - ./config/config.ini:/usr/app/config.ini  
      - ./storage:/usr/app/storage  
    container_name: client_frontend  
    ports:  
      - 8090:8090
```

The local folder structure that is needed within the edge node device is the following:
```
./  
├── model/  
│   ├── storage/  
│   └── config/  
└── client-compose.yml
```

The model folder contains the aggregated model served by the server. The storage folder contains static or dynamic data used for training the local model and the config folder contains the configuration file used for the Client component deployment.

The command used to deploy this component is:

```
docker compose -f client-compose.yml up
```
5 Research and Scientific Innovation

In this chapter, the research production generated during the development of task 4.4 is presented. Here several contributions are presented: (i) the study of widely used state-of-the-art databases, as well as the way to generate a joint database considering the characteristics of the previous ones, (ii) a description of the evaluation protocol that will be followed during the testing process of FedLPy, (iii) the description of the initial model used and (iv) the presentation of the first results obtained within the context of the ERATOSTHENES project, (v) the description of a PoC for the deployment of a Federated Learning Framework making use of FedLPy.

It is important to remark that, even if data from Pilots is not expected to follow the same structure or distribution as the public databases we are using, the work described here defines a training and evaluation procedure reliable, complete, and reproducible, which will be exploited during the Pilot.

5.1 Description of the databases

As the FedLPy-based solution proposed for the ERATOSTHENES project relies on machine learning techniques, data is crucial for the proper optimization of the proposed method. In this section, the datasets used for this purpose are presented, as well as the preprocessing procedure implemented in order to adapt these data to the implemented models.

5.1.1 Datasets

5.1.1.1 UNSW-NB15

This dataset was created by the University of New South Wales (UNSW) in 2015 [14]. It was generated using the IXIA tool where normal network activity was captured, and attacks were simulated. This database is composed of the traffic flows captured in the following different formats:

1. 100GB of pcap’s raw traffic flows from two recording days.
2. 2 million records of post-processed traffic flow features stored in CSV files.
3. Files generated by Argus tool from the pcaps traffic flows.
4. Files generated by BRO tool from pcaps traffic flows.

To this point, only the CSV files are used for training the models. However, in future updates of the implemented solution, the use of raw files of network traffic flows is being considered. As this problem can be addressed as a classification task, Table 3 describes the considered attacks (categories) that appear in the UNSW-NB15 dataset. Note that in [14] a full detailed list of the 47 different input features can be found.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzers</td>
<td>Attempting to cause a program or network suspended by feeding it with random data.</td>
</tr>
<tr>
<td>Analysis</td>
<td>Contains attacks of port scan, spam, and HTML file penetrations.</td>
</tr>
<tr>
<td>Backdoors</td>
<td>System security is bypassed stealthily to access a computer or its data.</td>
</tr>
<tr>
<td>DoS</td>
<td>Aims to corrupt resources of the IoT services and network systems.</td>
</tr>
<tr>
<td>Exploits</td>
<td>The attacker leverages a known security problem or vulnerability.</td>
</tr>
<tr>
<td>Generic</td>
<td>A technique works against all block-ciphers, without consideration of the structure of the block_cipher.</td>
</tr>
</tbody>
</table>

Table 3: UNSW-NB15 considered attacks extracted from [14].
Reconnaissance — attacks that gather information.
Shellcode — A small piece of code used as the payload in the exploitation of software vulnerability.
Worms — Attacker replicates itself in order to spread to other computers.

### 5.1.1.2 Aposemat IoT-23

Aposemat IoT-23 is a dataset [15] of network traffic from IoT devices extracted by the Stratosphere lab and funded by Avast Software. It has 20 malware captures and 3 captures of normal IoT device traffic. Both malicious and benign scenarios were run in a controlled network environment. First, in each malicious scenario, a specific malware is executed in a Raspberry Pi. Second, the real network traffic for the benign scenarios was obtained by capturing the network traffic of three IoT devices: a Philips HUE smart LED lamp, an Amazon Echo home and a Somfy smart door lock. These traffic flows are stored in two different formats:

1. Raw network traffic flow pcap files
2. Zeek tool network analyzer created conn.log labelled files.

When using this database, only the conn.log files are been used. Nevertheless, it is planned to use the raw pcap files in further updates of the project. Table 4 shows a summary of the considered attacks found in this dataset. In [15] a comprehensive list of the 21 input features used during training phases can be found.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Sub-attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack</td>
<td>---</td>
<td>Generic label for anomalies not identified.</td>
</tr>
<tr>
<td>C&amp;C</td>
<td>---</td>
<td>Control and command. Attack that takes control of the device to perform other attacks in the feature.</td>
</tr>
<tr>
<td>File Download</td>
<td></td>
<td>The server that controls the infected device sends it a file</td>
</tr>
<tr>
<td>Mirai</td>
<td></td>
<td>The attack is performed by the Mirai bot network.</td>
</tr>
<tr>
<td>Torii</td>
<td></td>
<td>The attack is performed by the Torii bot network.</td>
</tr>
<tr>
<td>Heartbeat</td>
<td></td>
<td>The server that controls the infected device sends periodic messages to check the status of the device.</td>
</tr>
<tr>
<td>Heart-Beat - Attack</td>
<td></td>
<td>Attack comes periodically from a suspicious source with an unknown method.</td>
</tr>
<tr>
<td>Heart-Beat File Download</td>
<td></td>
<td>The checking is performed with a file.</td>
</tr>
<tr>
<td>DDoS</td>
<td>---</td>
<td>Attacker replicates itself to spread to other computers.</td>
</tr>
<tr>
<td>Okiru</td>
<td>---</td>
<td>Attack performed by the Okiru bot network.</td>
</tr>
<tr>
<td>PartOfAHorizontal</td>
<td>---</td>
<td>Information is gathered from a device for future attacks.</td>
</tr>
</tbody>
</table>
5.1.1.3 ToN IoT

In this database “data from Telemetry datasets of IoT sensors. Also operating systems datasets from Windows 7 and 10 OS as well as Ubuntu 14 and 18 TLS OS and Network traffic datasets are included” [16]. The datasets were collected from a realistic and large-scale network designed at the Cyber Range and IoT Labs, the School of Engineering and Information technology (SEIT), UNSW Canberra and the Australian Defense Force Academy (ADFA).

The traffic flows captured in this dataset are storage in the following formats:

1. Raw pcap files that include all the packets with normal and malicious events.
2. The BRO tool generated files of these traffic flows.
3. 23 CSV files with post-processed traffic flow data.

The CSV files are used in the training of the ERATOSTHENES federated learning models and the different possible considered attacks are summarized in Table 5. Note that a detailed description of the 44 input features of this database can be found in [16].

Table 5: ToN IoT considered attacks extracted from [16].

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning</td>
<td>Aims to collect information from the victim system.</td>
</tr>
<tr>
<td>DoS</td>
<td>Aims to corrupt resources of the IoT services and network systems.</td>
</tr>
<tr>
<td>Ransomware</td>
<td>Prevents normal users to access systems or services by encrypting them until the pay a ransom.</td>
</tr>
<tr>
<td>Backdoor</td>
<td>Attackers get around security measures and get user access to a system.</td>
</tr>
<tr>
<td>Injection</td>
<td>Attackers inject fake input data into applications.</td>
</tr>
<tr>
<td>XSS</td>
<td>Attackers employ a web application to transmit malicious code to different users.</td>
</tr>
<tr>
<td>Password cracking</td>
<td>Password hacking, such as brute force.</td>
</tr>
<tr>
<td>Man-In-The-Middle</td>
<td>Attackers place themselves between users and applications and masquerade as one of the parties.</td>
</tr>
</tbody>
</table>

5.1.1.4 Merged Dataset

Under the context of the ERATOSTHENES project, the three above-described datasets are merged.

Table 6 shows all the possible considered attacks. In addition, Table 7 describes the common features that are used during the training phases.
### Federated threat analysis models for continuous risk assessment

Table 6: Merged dataset considered attacks.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>All</td>
<td>Traffic without anomalies.</td>
</tr>
<tr>
<td>C&amp;C</td>
<td>Aposemat IoT-23</td>
<td>Control and command. Attack that takes control of the device to perform other attacks in the feature.</td>
</tr>
<tr>
<td>C&amp;C-HeartBeat</td>
<td>Aposemat IoT-23</td>
<td>Aims to collect information from the victim system.</td>
</tr>
<tr>
<td>Mirai</td>
<td>Aposemat IoT-23</td>
<td>The attack is performed by the Mirai bot network.</td>
</tr>
<tr>
<td>Torii</td>
<td>Aposemat IoT-23</td>
<td>The attack is performed by the Torii bot network.</td>
</tr>
<tr>
<td>DoS</td>
<td>All</td>
<td>Attempting to make server or network resources unavailable to users by temporarily interrupting the service of a host connected to the Internet.</td>
</tr>
<tr>
<td>FileDownload</td>
<td>UNSW-NB15 Aposemat IoT23</td>
<td>The server that controls the infected device sends periodic messages to check the status of the device by using a file.</td>
</tr>
<tr>
<td>Okiru</td>
<td>Aposemat IoT23</td>
<td>Attack performed by the Okiru bot network.</td>
</tr>
<tr>
<td>Port Scan</td>
<td>All</td>
<td>Aims to collect information from the victim system.</td>
</tr>
<tr>
<td>Attack</td>
<td>UNSW-NB15</td>
<td>Generic label for anomalies not identified.</td>
</tr>
<tr>
<td>Ransomware</td>
<td>ToN IoT</td>
<td>Prevents normal users to access systems or services by encrypting them until the pay a ransom.</td>
</tr>
<tr>
<td>Backdoor</td>
<td>UNSW-NB15 ToN IoT</td>
<td>Attackers get around security measures and get user access to a system.</td>
</tr>
<tr>
<td>Exploits</td>
<td>UNSW-NB15</td>
<td>The attacker leverages a known security problem or vulnerability.</td>
</tr>
<tr>
<td>Generic</td>
<td>UNSW-NB15</td>
<td>A technique works against all block-ciphers, without consideration of the structure of the block-cipher.</td>
</tr>
<tr>
<td>Worms</td>
<td>UNSW-NB15</td>
<td>Attacker replicates itself to spread to other computers.</td>
</tr>
<tr>
<td>Injection</td>
<td>UNSW-NB15 ToN IoT</td>
<td>Attackers inject fake input data into applications.</td>
</tr>
<tr>
<td>XSS</td>
<td>ToN IoT</td>
<td>Attackers employ a web application to transmit malicious code to different users.</td>
</tr>
<tr>
<td>Password cracking</td>
<td>ToN IoT</td>
<td>Password hacking, such as brute force.</td>
</tr>
<tr>
<td>Shellcode</td>
<td>UNSW-NB15</td>
<td>A small piece of code used as the payload in the exploitation of software vulnerability.</td>
</tr>
</tbody>
</table>
Table 7: Common features used for the merged database.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id.orig_p</td>
<td>The origin network port used during the connection.</td>
</tr>
<tr>
<td>id.resp_p</td>
<td>The receiver network port used during the connection.</td>
</tr>
<tr>
<td>proto</td>
<td>The protocol used for the connection.</td>
</tr>
<tr>
<td>service</td>
<td>The network service used for the connection.</td>
</tr>
<tr>
<td>duration</td>
<td>The duration of the connection.</td>
</tr>
<tr>
<td>orig_bytes</td>
<td>The number of bytes sent by the origin node.</td>
</tr>
<tr>
<td>resp_bytes</td>
<td>The number of bytes received by the receiver node.</td>
</tr>
</tbody>
</table>

5.1.2 Dataset Preprocessing

In order to adapt the input data to the models used in the proposed solution for this project's continuous risk assessment task, it is necessary to implement a data pre-processing stage. With this, homogeneity of the data is achieved, thus improving the optimization process of the model parameters.

The steps followed in this pre-processing phase are:

1. **Protocol Feature Grouping:** Those web communication protocols with limited presence in the database are grouped together to form a new category called "Others". Thus, the possible protocols that conform the database are TCP, UDP and Others.
2. **Service Feature Grouping:** Similarly, those web services with low presence are grouped in the "Others" category. Therefore, the possible services that appear in the database are DNS, FTP, HTTP, SMTP, SSH and Others.
3. **Variable Categorical Encoding:** Categorical variables, such as features protocol or service, are encoded numerically with integers.
4. **Standardization:** All features are statistically normalized according to the following function:

   \[
   f(x) = \frac{x - \mu}{\sigma}
   \]

   where \( x \) is the vector of features in the database, \( \mu \) is the vector of means of each of the features in the dataset, and \( \sigma \) is the vector of standard deviations of those features.

5. **Data Balancing:** This data pre-processing step is not mandatory. When applied, it ensures that in each subset of data generated (training, validation, and testing subsets) all classes are forced to have the same representation. This balancing process is carried out by means of under-sampling techniques of the classes with more representation.

5.2 Evaluation protocol

The aim of the initial experiments conducted is twofold. On the one hand, to verify the proper functioning of the developed FedLPy component. On the other, to evaluate the performance of the initial models by simulating real
scenarios of threat detection in IoT networks. Although the results presented in this deliverable are preliminary, it has been decided to continue the evaluation process initiated here in subsequent iterations of the project.

5.2.1 Overall description

The evaluation process of ML models for threat detection in IoT networks will consist of two benchmarks carried out in the same test bed. The first benchmark involves the comparison of the performance of state-of-the-art threat detection models trained and evaluated in a federated way. Then, the performance of the best models will be compared when training with privacy-preserving techniques (DP-SGD and PATE) tuned to achieve the required privacy and accuracy guarantees for the specific case.

It is important to remark that the current version of FedLPy does not support Machine Learning models apart from Deep Learning-based models. However, the impressive performance of some traditional ML models (such as Decision Trees) with centralized training, generates the need to integrate such algorithms into FedLPy for subsequent analysis of their federated performance. A description of the results obtained by these models with centralized training can be seen in Deliverable 4.5.

5.2.2 Federated Threat Analysis test bed

In order to evaluate the performance of the Federate Machine Learning algorithms implemented with FedLPy, we have designed a test bed based on a regular IoT network scheme. This test bed is based on the simulation of a Federated Threat detector composed of two FedLPy clients and a FedLPy server. To simulate the local traffic data of each client, the merged dataset is divided into two datasets of the same size. Following standard machine learning practices each local dataset is split into training, validation, and test sets. Approximately 70% of data is allocated for training, 20% for validation and 10% for testing.

This test bed has been deployed as shown in Figure 8, with two raspberry pi 3 (ARM64 processor architecture) playing the role of FML edge client nodes, while a laptop with AMD64 processor architecture does the FML server function.

The deployment has been carried out following the configuration steps explained in section 4.3 and choosing the docker images of the necessary processor architecture.

5.2.3 Evaluation metrics

The threat detection task is initially considered a supervised binary classification task. Thus, the models will return if a traffic flow is malicious (1) or benign (0). However, it can also be considered a multiclass classification task, where the model must return the class of attack detected. Thus, the performance of the models is assessed using the following metrics:
- **Confusion matrix**: representation that provides a comprehensive breakdown of the predicted and real class for a given dataset. Directly from the confusion matrix, we can extract:
  - **True Positives (TP)**: positive samples predicted as positive.
  - **False Positives (FP)**: negative samples predicted as positive.
  - **True Negatives (TN)**: negative samples predicted as negative.
  - **False Negatives (FN)**: positive samples predicted as negative.

- **Accuracy**: proportion of correctly classified samples

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]

- **Precision**: proportion of true positives out of the total predicted positives.

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

- **Recall**: proportion of true positives out of the total actual positives.

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

- **F1-score**: harmonic mean of the precision and recall.

\[
F1 = 2 \cdot \frac{\text{Precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]

Note that some of these metrics, specifically f1-score, precision and recall metrics, are specific for binary classification tasks. For this reason, in the evaluation of multiclass models these metrics will be computed for each class independently, treating that class as positive and the remaining classes as negative.

### 5.3 Initial models

This section describes the initial models that have been implemented and evaluated. Additional models will be evaluated in the next steps of the task. The first model is a Fully Connected Neural Network (FCNN) trained for binary classification, while the second model is the previous FCNN adapted to multiclass traffic threat detection. Table 8 and Table 9 represent the architecture of both models, note that the difference between both resides in the output layer.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Activation fn</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input_1 (InputLayer)</td>
<td>[None, 9]</td>
<td>ReLU</td>
<td>0</td>
</tr>
<tr>
<td>FC (Dense)</td>
<td>(None, 128)</td>
<td>ReLU</td>
<td>1280</td>
</tr>
<tr>
<td>FC_1 (Dense)</td>
<td>(None, 64)</td>
<td>ReLU</td>
<td>8256</td>
</tr>
<tr>
<td>FC_2 (Dense)</td>
<td>(None, 16)</td>
<td>ReLU</td>
<td>1040</td>
</tr>
<tr>
<td>FC_3 (Dense)</td>
<td>(None, 1)</td>
<td>Sigmoid</td>
<td>17</td>
</tr>
</tbody>
</table>

Total params: 10,593
Trainable params: 10,593
Non-trainable params: 0
For training and evaluating the second model, only attack classes with more examples than 10,000 are considered initially, the rest of the attacks have been grouped in a category called “Others”. Thus, the classes considered by the model are Port Scan, DoS, XSS, password, injection, Backdoor, Other attacks and Benign traffic.

Table 9: Architecture of MultiClass FCNN model

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Activation fn</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input_1 (InputLayer)</td>
<td>[(None, 9)]</td>
<td>ReLU</td>
<td>0</td>
</tr>
<tr>
<td>FC (Dense)</td>
<td>(None, 128)</td>
<td>ReLU</td>
<td>1280</td>
</tr>
<tr>
<td>FC_1 (Dense)</td>
<td>(None, 64)</td>
<td>ReLU</td>
<td>8256</td>
</tr>
<tr>
<td>FC_2 (Dense)</td>
<td>(None, 16)</td>
<td>ReLU</td>
<td>1040</td>
</tr>
<tr>
<td>FC_3 (Dense)</td>
<td>(None, 8)</td>
<td>Softmax</td>
<td>136</td>
</tr>
<tr>
<td>Total params: 10,712</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trainable params: 10,712</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-trainable params: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.4 Preliminary results

This section shows the preliminary results obtained for the initial models in the test bed described in section 5.2. To do this, results have been divided into three sections. In the first section, the evaluation metrics obtained for both Federated models are represented. In the second, the performance of the federated models when they are trained using the DP-SGD algorithm, is represented. In every case, the configuration parameters of the Federated Learning approach have been set to the default values provided by FedLPy.

It is important to note that no relevant results have been obtained with the implementation of PATE in FedLPy. This privacy-preserving machine learning technique will be further analyzed in future versions of the package and in future steps of the project.

5.4.1 Federated Threat Analysis

Threat Analysis as a Binary classification

The first experiment performed within the context of the Federated Threat Analysis topic addresses the problem of traffic flow binary classification or anomaly detection. In order to do so, the binary FCNN model summarized in Table 8 is training with the composed merged database. Subsequently, the metrics obtained by the model in the testing phase are shown in Table 10, where all of them are above 90%. It is important to note that the precision of the Binary FCCN model is over 98% and the F1 score (which considers the data imbalance issues) is around the 94.5%. Additionally, Figure 9 shows the non-normalized confusion matrix obtained for this task, where about a 98.34% of the attacks are correctly detected.
4.4 Federated threat analysis models for continuous risk assessment

### Table 10: FCNN performance for binary classification

<table>
<thead>
<tr>
<th>Metric</th>
<th>Binary FCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>94.575%</td>
</tr>
<tr>
<td>Precision</td>
<td>98.206%</td>
</tr>
<tr>
<td>Recall</td>
<td>90.808%</td>
</tr>
<tr>
<td>F1 score</td>
<td>94.362%</td>
</tr>
</tbody>
</table>

![Figure 9: Confusion matrix for Federated Threat Analysis with Binary FCNN.](image)

**Threat Analysis as a Multiclass Classification**

The second experiment performed to assess the FML training procedure consists of addressing the multiclass threat analysis issue. Therefore, the model should retrieve the type of attack the edge node is suffering instead of detecting that an anomaly event is happening within the traffic flow.

As shown in Table 11, the results obtained in this task are worse than the mentioned above with a reduction of 20% of precision and a 10% of F1 score. This is because in this task the dimensionality of the problem increases, so it becomes a harder problem to be solved. Nevertheless, according to the non-normalized confusion matrix shown in Figure 10, attacks are always detected with over 60% of precision.

### Table 11: Federated Threat Analysis with Multiclass FCNN

<table>
<thead>
<tr>
<th>Metric</th>
<th>MC FCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>82.978%</td>
</tr>
<tr>
<td>Precision</td>
<td>78.161%</td>
</tr>
<tr>
<td>Recall</td>
<td>91.527%</td>
</tr>
<tr>
<td>F1 score</td>
<td>84.318%</td>
</tr>
</tbody>
</table>

![Figure 10: Confusion matrix for Federated Threat Analysis with Multiclass FCNN.](image)

5.4.2 Privacy-preserving Federated Threat Analysis with DP-SGD

Experiments described in the previous section have been repeated training the models with DP-SGD. Thus, this section summarizes the results of models described in Table 12 and Table 13, trained with DP-SGD. The configuration of the privacy-preserving algorithm has been set with the default values defined in *FedLPy*.
Threat Analysis as a Binary Classification

Table 12: DP-SGD-based FCNN performance for binary classification.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Binary FCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>80.465%</td>
</tr>
<tr>
<td>Precision</td>
<td>81.151%</td>
</tr>
<tr>
<td>Recall</td>
<td>79.371%</td>
</tr>
<tr>
<td>F1 score</td>
<td>80.251%</td>
</tr>
</tbody>
</table>

Figure 11: Confusion matrix for Federated Threat Analysis with DP-SGD-based Binary FCNN.

Threat Analysis as a Multiclass Classification

Table 13: DP-SGD-based FCNN performance for binary classification.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Binary FCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>57.759%</td>
</tr>
<tr>
<td>Precision</td>
<td>57.759%</td>
</tr>
<tr>
<td>Recall</td>
<td>57.759%</td>
</tr>
<tr>
<td>F1 score</td>
<td>57.759%</td>
</tr>
</tbody>
</table>

Figure 12: Confusion matrix for Federated Threat Analysis with DP-SGD-based Binary FCNN.

As can be seen in Figure 11 and Figure 12, the inclusion of a differential privacy algorithm during the learning phase of the model leads to a degradation in their performance in terms of accuracy. This is aligned with the principles of differential privacy theory, which requires the introduction of greater noise in the training phase to achieve privacy guarantees. Consequently, the utility of the final model diminishes. Thus, the precision of the Binary FCNN model has decreased from 98% to 80% by introducing the DP-SGD algorithm.

This behaviour is amplified for the multiclass classification of threats, where the accuracy has decreased by almost a 25%, from 83% to 57%. For this reason, as future steps, it is established to analyze the trade-off between privacy and performance and configure the privacy-preserving techniques in order to achieve the privacy requirements for the users in ERATOSTHENES pilots.
6 Conclusions

This chapter presents the conclusions drawn from the work done to date for task 4.4 named “Federated threat analysis models for continuous assessment”. Moreover, possible improvement lines are presented for future versions of the components to be implemented for the following deliverables.

6.1 Main Conclusions

This document provides a description of the utility and usage of the component resulting from task 4.4, called FedLPy. The current version of this package allows the development and deployment of a Privacy-Preserving Federated Machine Learning framework for the Network Threat Analysis in ERATOSTHENES.

As main contributions for this deliverable can be highlighted in the following:

- The FedLPy package functionalities have been defined in order to allow users to easily develop Privacy-Preserving Federated Machine Learning systems.
- The first version of the FedLPy module has been implemented according to the functionalities defined.
- An easy-to-deploy methodology has been implemented, so the deployment process can be done in an automatic manner.
- A binary and a multiclass model have been designed relying on Deep Learning techniques in order to address the Federated Threat Analysis for Continual risk Assessment task within the ERATOSTHENES project.
- The first results have been obtained regarding the non-secured Federated Machine Learning Training, as well as results according to Differential Privacy-based approaches.

Although the results shown in this first deliverable are highly satisfactory, further updates to the FedLPy package and the deployment procedure are needed to be addressed in order to achieve the set requirement within the context of the project.

6.2 Future Improvement Lines

As a result of the conclusions detailed above, for the following months of work, the next lines of improvement are proposed:

1. Include the FML server within the ERATOSTHENES TMB component: This modification will allow the FML server to know prior information about the nodes connected to the ERATOSTHENES network, leveraging those insights to better schedule trainings.
2. Connect FedLPy to the IDS module from ERATOSTHENES TMB following the architecture defined in sections 3.1.2 and 3.1.3. This step will be performed in collaboration with task 4.5.
3. Machine Learning Benchmarking: An extensive comparison between the performance of different machine learning models for the task at hand is scheduled to be performed. All the models evaluated will be integrated into FedLPy in order to reproduce them with Pilot 1 databases.
4. Agree on the same threat detection model among the project tasks: This would be beneficial for all project tasks involved in detecting cyber threats in IoT networks since already trained models could be used as a starting point for the FML system. Also, models from other tasks can benefit from model updates provided by the FML system.
5. Optimize the system: Once representative data of the network traffic flows from pilots are available, where the performance of the proposed FML system will be evaluated. The proposed models, as well as the deployment configurations, can be optimized to achieve the highest possible performance in terms of network threat detection.
7 References


