D3.5: Methods and solutions to achieve security and robustness

D3.5

Methods and solutions to achieve security and robustness

21/07/2021
Revision history

Administration and summary

Project acronym: DECENTER

Document identifier: D3.5: Methods and solutions to achieve security and robustness [M36]

Leading partner: ATOS

Report version: 1.2

Report preparation date: 21.07.2021

Classification: PU (Public)

Nature: R (Report)

Author(s) and contributors: Silvio Cretti (FBK)

Roberto Doriguzzi-Corin (FBK)

Klaus Kim (GLSYS)

Roi Sucasas Font (ATOS)

Alejandro García Bedoya (ATOS)

Status:
- Plan
- Draft
X Final

The DECENTER Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

<table>
<thead>
<tr>
<th>Date</th>
<th>Edited by</th>
<th>Status</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.03.2021</td>
<td>Alejandro García Bedoya</td>
<td>Plan</td>
<td>Table of contents (ToC)</td>
</tr>
<tr>
<td>20.04.2021</td>
<td>Silvio Cretti</td>
<td>Draft</td>
<td>Sections 3.3, 4.3</td>
</tr>
<tr>
<td>04.05.2021</td>
<td>Silvio Cretti</td>
<td>Draft</td>
<td>Section 2</td>
</tr>
<tr>
<td>04.05.2021</td>
<td>Klaus Kim</td>
<td>Draft</td>
<td>Section 2</td>
</tr>
<tr>
<td>05.05.2021</td>
<td>Roberto Doriguzzi Corin</td>
<td>Draft</td>
<td>Sections 3.2, 4.2</td>
</tr>
<tr>
<td>05.05.2021</td>
<td>Roi Sucasas Font</td>
<td>Draft</td>
<td>Section 4.1.1</td>
</tr>
<tr>
<td>05.05.2021</td>
<td>Alejandro García Bedoya</td>
<td>Draft</td>
<td>Section 3.1.1</td>
</tr>
<tr>
<td>11.05.2021</td>
<td>Alejandro García Bedoya</td>
<td>Draft</td>
<td>Section 3.1</td>
</tr>
</tbody>
</table>
D3.5: Methods and solutions to achieve security and robustness

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01.06.2021</td>
<td>Alejandro García Bedoya</td>
<td>Draft</td>
<td>Remainder Sections and full version</td>
</tr>
<tr>
<td>07.06.2021</td>
<td>Roberto Doriguzzi Corin</td>
<td>Draft</td>
<td>Added references in Section 3.2</td>
</tr>
<tr>
<td>07.06.2021</td>
<td>Roberto Doriguzzi Corin</td>
<td>Draft</td>
<td>First review of the document</td>
</tr>
<tr>
<td>16.06.2021</td>
<td>Alejandro García Bedoya</td>
<td>Draft</td>
<td>Fixed the review comments</td>
</tr>
<tr>
<td>09.07.2021</td>
<td>Seungwoo Kum</td>
<td>Draft</td>
<td>Review of the document</td>
</tr>
<tr>
<td>09.07.2021</td>
<td>Alejandro García Bedoya</td>
<td>Draft</td>
<td>Fixed the review comments</td>
</tr>
<tr>
<td>14.07.2021</td>
<td>Alejandro García Bedoya</td>
<td>Final</td>
<td>Final version</td>
</tr>
</tbody>
</table>

Notice that other documents may supersede this document. A list of latest public DECENTER deliverables can be found at the DECENTER Web page at https://www.decenter-project.eu/.

Copyright

This report is © DECENTER Consortium 2018. Its duplication is restricted to the personal use within the Consortium, funding agency and project reviewers.

Acknowledgements

This project has received funding from the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence)

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT/IITP) (No. 1711075689, Decentralised cloud technologies for edge/IoT integration in support of AI applications).

The partners in the project are FONDAZIONE BRUNO KESSLER (FBK), ATOS (ATOS), COMMISSARIAT À L’ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (CEA), COMUNE DI TRENTO (TN), ROBOTNIK (ROB), UNIVERZA V LJUBLJANI (UL), KOREA ELECTRONICS TECHNOLOGY INSTITUTE (KETI), GLUESYS (GLSYS), DALIWORKS (DW), LG U+ (LGUP), SEOUL NATIONAL UNIVERSITY (SNU).

The content of this document is the result of extensive discussions within the DECENTER © Consortium as a whole.

More information

Public DECENTER reports and other information pertaining to the project are available through DECENTER public Web site under http://www.decenter-project.eu.
D3.5: Methods and solutions to achieve security and robustness

Contents

Revision history .................................................................................................................. 2
Copyright ............................................................................................................................. 3
Acknowledgements ............................................................................................................. 3
More information ................................................................................................................ 3
List of figures ...................................................................................................................... 6
List of tables ....................................................................................................................... 7
Executive Summary ............................................................................................................ 8

1 Introduction ...................................................................................................................... 9

2 Cross-border Security ................................................................................................... 10

2.1 Security on Microservice-based Cloud Infrastructure ............................................. 10

2.1.1 Transition from perimeter-based security to zero trust security ...................... 11

2.1.2 Transition from static IP and hardware to shared resources with high utilization .... 11

2.1.3 Transition from application-specific security implementation to shared security requirements integrated into the service stack ......................................................... 11

2.1.4 Transition from a specialized, intermittent deployment process to a standard process with a high frequency of deployment ............................................................ 12

2.1.5 Transition from isolated workload through physical machines or hypervisors to loaded workloads that require strong isolation running on the same machine ............ 12

2.1.6 Security principles ................................................................................................. 12

2.2 Cross-border Federated Scenario ............................................................................ 13

3 Analysis of Security/Robustness solutions ................................................................ 15

3.1 Security of the microservices .................................................................................. 15

3.1.1 Exploratory Data Analysis (EDA) ...................................................................... 15

3.1.2 Pre-process and DL Model .................................................................................. 20

3.2 Security of the Fog nodes ....................................................................................... 23

3.2.1 System architecture ............................................................................................ 23

3.2.2 Convolutional layer .......................................................................................... 24

3.2.3 MaxPooling layer ............................................................................................. 25

3.2.4 Classification layer ........................................................................................... 25

3.2.5 Hyper-parameters tuning .................................................................................. 25

3.2.6 Datasets .............................................................................................................. 25

3.3 Robustness of the platform ..................................................................................... 27


4.1 Security of the microservices .................................................................................. 29
4.1.1 Implementation of the solution L-ADS ................................................................. 29
4.1.2 Evaluation ............................................................................................................. 30
4.2 Security of the Fog nodes ....................................................................................... 33
4.2.1 Implementation .................................................................................................... 33
4.2.2 Detection accuracy .............................................................................................. 33
4.2.3 Intrusion detection at the edge ........................................................................... 34
4.3 Robustness of the platform .................................................................................... 36
5 Conclusion .................................................................................................................. 37
6 References ................................................................................................................. 38
7 Glossary & Abbreviations ......................................................................................... 40
List of figures

Figure 1: Cross border infrastructure ................................................................. 13
Figure 2: Example of CIDDS dataset ................................................................. 16
Figure 3: Flow Chart of Src IP and Dst IP ......................................................... 17
Figure 4: Topology of the CIDDS network ....................................................... 17
Figure 5: Categorical Variable – Proto .............................................................. 18
Figure 6: Categorical Variable - TCP Flags ...................................................... 18
Figure 7: Categorical Variable – Tos ................................................................. 19
Figure 8: Statistical summary of numeric variables .......................................... 19
Figure 9: Correlation of numeric variables ....................................................... 20
Figure 10: Architecture of the Autoencoder ..................................................... 22
Figure 11: MSE function ................................................................................. 22
Figure 12: Workflow of intrusion detection and mitigation processes .............. 24
Figure 13: Single cluster architecture .............................................................. 27
Figure 14: Multi-cluster federated architecture .............................................. 28
Figure 15: L-ADS in Kubernetes ..................................................................... 29
Figure 16: L-ADS logs in the UC2 ................................................................. 31
Figure 17: Workflow of L-ADS in the UC2 ..................................................... 32
Figure 19: Inference performance on the NVIDIA TX2 board ..................... 35
Figure 20: FedFAApp model ........................................................................... 36
List of tables

Table 1: Comparison of security aspects................................................................. 10
Table 2: Example of standardization – Bytes. ............................................................... 21
Table 3: Example of dummies variables – Proto. .......................................................... 21
Table 4: Summary of the DDoS datasets. .................................................................... 26
Table 5: Summary of the CICIDS2017 dataset............................................................. 26
Table 6: L-ADS performance using the CIDDS dataset.............................................. 32
Table 7: L-ADS performance using the script on the UC2 connections. ...................... 33
Table 8: DDoS detection performance. ....................................................................... 33
Table 9: Intrusion detection performance on the CICIDS2017 test set. ....................... 34
Executive Summary

This document summarizes the achievements made during the project of DECENTER with the aim of proposing methods and solutions to improve the security and the robustness of the DECENTER platform. It is the continuation of the D3.3 [8] where these solutions were introduced.

This document describes a final and detailed analysis of the proposed solutions that have been implemented in the DECENTER fog computing platform. Those solutions are the L-ADS (Live Anomaly Detection System) as a solution for the security of microservices, an IDPS (Intrusion Detection and Prevention System) for the security of fog nodes and a solution to ensure the robustness of the platform. Furthermore, it also contains the security aspects related to the cross-border data security and intra-domain robustness against failures.

Finally, the document presents the final evaluation and implementation of those tools that have been developed with the aim of increasing the security of microservices and fog nodes.
D3.5: Methods and solutions to achieve security and robustness

1 Introduction

The DECENTER fog platform developed throughout the project is supported by the solutions proposed in task T3.4 (Infrastructure security and robustness). They are made to ensure a horizontal secure and vertical robust deployment of the platform.

The security and robustness solutions, introduced in deliverable D3.3 [8], have been developed during the year 2 and year 3 of the DECENTER project. All the developments during the year 2 regarding security and robustness were reported in D3.3.

In this deliverable, we present three main points: Cross-border Security, Analysis of the proposed solutions and the implementation and evaluation of those solutions. The Cross-border security (Section 2) delves into the investigations to ensure the security on microservices-based cloud infrastructure and presents a cross-border federated scenario with the aim of exposing a real-life case where a cluster-to-cluster communication is secured and encrypted. Then, the document continues giving a low-level view of the solutions, completing the information provided by the deliverable D3.3. There are three solutions proposed with the goal of improving the security of the microservices, the security of the fog nodes and the robustness of the DECENTER fog computing platform. Section 3 reports a final and detailed analysis of these solutions. In addition, the security solutions that are based on a DL model shows in detail the architecture with the different kind of layers of the algorithm implemented. However, it should be noted what are the similarities and differences between the security of microservices and the security of fog nodes. Both are based on DL algorithms although the security of microservices uses RNN (Recurrent Neural Network) and the security of fog nodes uses CNN (Convolutional Neural Network). In our case, the fog computing platform contains different nodes, where the security of fog nodes tries to detect anomalies analysing the packets (EBPF and XDP technologies), and inside the nodes are pods, where the security of microservices tries to detect anomalies analysing the characteristics of the connections (Softflowd tool).

The implementation and evaluation of the three solutions (Section 4) show what are the steps followed to obtain these objectives. Additionally, for the readers with a special interest in integration understanding, we encourage to read D5.2 [28] (Final release of the AI-integrated fog computing platform and final setups of pilots for demonstrations) which will be published at the same time of this deliverable, and which contains more information related the integration of the solutions in the DECENTER platform.

About the evaluation, the three components present some results of experimental use. The security of the fog nodes and the security of microservices solutions expose the metrics obtained by the DL algorithms using different datasets.

Finally, the document describes the conclusions reached (Section 5) throughout of the work package of DECENTER project.
D3.5: Methods and solutions to achieve security and robustness

2 Cross-border Security

During this section, we present the investigations related to the cross-border data security and then, we show a cross-border federated scenario.

2.1 Security on Microservice-based Cloud Infrastructure

In a microservice-based cloud environment, we cannot simply rely on a firewall to protect the service network. All resources are distributed across various network environments, and users do not use all devices in the same physical location. Just as users move, use multiple devices, and connect from different locations, microservices move and are deployed in different environments across heterogeneous hosts.

The traditional firewall-centric perimeter-based security model is not very effective in protecting microservices-based cloud architectures. In the microservices cloud-based model, the container separates the binaries required by the application from the underlying host cloud operating system, improving the mobility of the application. Containers are immutable, so they don’t change after deployment. Therefore, it is highly rebuilt and redistributed. The job scales to handle the load by deploying new jobs when the load increases and terminating existing jobs when the load decreases. As containers are frequently restarted, shut down, or rescheduled, so does the frequency of reusing and sharing hardware and networking.

Table 1 compares the security aspects of a traditional infrastructure with that of a cloud-based architecture.

<table>
<thead>
<tr>
<th>Security for traditional infrastructure</th>
<th>Microservice based Cloud Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perimeter-based security (for example, firewall) where internal communications are considered trusted communications</td>
<td>Zero trust security where communication between services is verified and there is no implicit trust relationship to services in the environment</td>
</tr>
<tr>
<td>Static IP and hardware for specific applications</td>
<td>Improve resource utilization, reuse and sharing, including IP and hardware</td>
</tr>
<tr>
<td>IP address-based ID</td>
<td>Service-based ID</td>
</tr>
<tr>
<td>Run the service from a known and expected location</td>
<td>Services can run anywhere in the environment, including hybrid deployments across public clouds and private data centers.</td>
</tr>
<tr>
<td>Security-specific requirements are built into each application and applied separately</td>
<td>Shared security requirements are integrated into the service stack according to a centralized enforcement policy.</td>
</tr>
<tr>
<td>Not many restrictions on how to build and review the service</td>
<td>Security requirements are consistently applied across all services</td>
</tr>
<tr>
<td>Lack of oversight of security components</td>
<td>Centralized view to check security policy and compliance</td>
</tr>
<tr>
<td>Specialized and intermittent deployment</td>
<td>Standardized build and deployment process with high change frequency of individual microservices</td>
</tr>
<tr>
<td>Workloads are usually deployed on VMs or physical hosts and are isolated through physical machines or hypervisors</td>
<td>Loaded workloads and related processes run on a shared operating system, so you need a mechanism to isolate the workload</td>
</tr>
</tbody>
</table>
2.1.1 Transition from perimeter-based security to zero trust security

In the traditional security model, the organization’s applications used external firewalls around private data centers to protect against incoming traffic. In a microservice-based cloud environment, network perimeters must be protected, but perimeter-based models are not sufficient. This doesn't create new security issues, but we face the reality that if a firewall doesn't fully protect the corporate network, it doesn't fully protect the service network. In the zero-trust security model, internal traffic is no longer implicitly trusted, requiring other security controls such as authentication and encryption. At the same time, when moving to microservices, we have the opportunity to rethink our existing security model. By removing dependencies from a single network perimeter (such as a firewall), we can further segment the network by services. We can go one step further and implement microservice-level segmentation without the inherent trust relationships between services. Traffic from microservices contains different levels of trust with different controls, so we no longer have to compare internal and external traffic.

2.1.2 Transition from static IP and hardware to shared resources with high utilization

In the traditional security model, an organization's applications are deployed to specific machines, and those machines' IP addresses do not change very often. In other words, the security tool is able to use a relatively static map of the architecture that connects the application in a predictable way. For example, security policies for tools such as firewalls can use IP addresses as identifiers.

However, in a microservice-based cloud environment with shared hosts and frequently changing operations, access between microservices through a firewall cannot be controlled. We should not rely on the fact that certain IP addresses are tied to certain services. Therefore, rather than performing the analysis on the ID of the IP address or hostname, it should be based on the ID of the service.

2.1.3 Transition from application-specific security implementation to shared security requirements integrated into the service stack

In the traditional security model, individual applications had to meet their own security requirements independently of other services. These requirements include identity management, SSL/TLS termination, and data access management. This has made it more difficult to apply changes because implementations are often inconsistent, or security issues have not been addressed in multiple locations.

In a cloud-based environment, components are reused much more frequently between services, and there are choke points to ensure that policies are applied consistently across services. If we have different policies, we can use different security services to apply them. Different policies can be split into separate microservices without the need to separately implement critical security services in every application. For example, one policy is implemented to ensure authorized access to user data, and another policy is implemented to ensure the use of the latest TLS cipher suites.
2.1.4 Transition from a specialized, intermittent deployment process to a standard process with a high frequency of deployment

In the traditional security model, there were not many shared services. Because the code is more distributed, coupled with local development, it is difficult to see the impact of changes that have affected different parts of the application, resulting in less frequent deployment and difficult to adjust. Each component needs to be manually updated by the developer before the change could be made. For example, to update the configuration, we need to connect to the virtual machine via SSH. As a result, the application overall has been around for quite some time. On the security side, as the code becomes more distributed, it has become more difficult to review, and it is more difficult to fix vulnerabilities everywhere. Moving to a cloud-based approach with a high frequency and standardized deployment process enables to implement zero-point security in a software development lifecycle. This makes it easier and more consistent to apply security hygiene, including regular security patching.

2.1.5 Transition from isolated workload through physical machines or hypervisors to loaded workloads that require strong isolation running on the same machine

In the traditional security model, workloads were reserved on their own instances without shared resources. Applications were effectively separated by machine and network perimeter, and workloads could be isolated using only physical host separation, hypervisors, and traditional firewalls.

In a cloud-based environment, workloads are containerized and loaded onto shared hosts and shared resources. Therefore, stronger isolation measures should be applied between workloads. These latter can be separated into microservices that are partially isolated from each other using network control and sandboxing techniques.

2.1.6 Security principles

At the same time as the microservice-based architecture development, we have developed and optimized the following security principles to enhance security.

- **Network protection at the edge:** Isolate workloads from unauthorized traffic and network attacks coming from the Internet. The firewall-based approach is not a new concept for microservice-based cloud but can still be used as a security recommendation. In this environment, the perimeter approach is used to protect the infrastructure as much as possible from unauthorized traffic and possible attacks coming from the Internet, such as volume-based denial of service attacks.
- **No inherent mutual trust relationship between services:** Only known, trusted, and specially authorized callers can utilize the service, preventing attackers using untrusted code from accessing the service. If a service breach occurs, it prevents an attacker from exploiting this vulnerability to expand the attack surface. This mutually distrusted structure helps to limit the scope of the breach's impact.
- **Trusted machines running code from known sources restricts service identities to run only in approved and validated environments with authorization code and configuration only.**
- **Choke point for consistent application of policies across the service:** For example to confirm requests for access to user data when access to the service is derived from verified requests from authorized end users and the administrator's access requires a business justification Choke point.
D3.5: Methods and solutions to achieve security and robustness

- Automated, standardized, and simple change deployment: Easily review the security impact of infrastructure changes and deploy security patches without impact to production.
- Isolation between workloads sharing an operating system: A service breach does not affect the security of other workloads running on the same host.

Security should scale in the same way as service extensions. Services should be secured by default, and services that are not secure should be treated as exceptions. Human intervention should be exceptionally acceptable and auditable, not routine. This allows the service to be authenticated based on the code and configuration deployed to the service on behalf of the user who deployed the service.

Implementing the above security principles together allows containers and microservices running in-house to be deployed and communicated with each other without compromising the nature of cloud-based architectures (e.g. easy workload management, no-ops scaling, effective loading). And can be run at a distance adjacent to each other. All of this can be achieved with minimal burden on individual microservice developers through the security and implementation details of the underlying infrastructure.

### 2.2 Cross-border Federated Scenario

With the support of Task 3.1, we set up a cross-border federated Kubernetes clusters. As depicted in Figure 1, the two independent yet federated clusters were installed in Europe and Korea: more specifically we installed a Member Cluster in Korea and a Host Cluster in Europe (see section 3.3) interconnected with a mesh VPN (Wireguard [6]) capable to guarantee the L3 secure connectivity among all Kubernetes nodes.

![Figure 1: Cross border infrastructure.](image)

A simple cloud native application, able to gather images from a camera and process them, has been deployed on top of such an infrastructure: the application is composed by the following modules:

- A **processor** able to retrieve images from a camera, extract the text embedded in the images (via an Optical Character Recognition library) and finally to store such a text in a repository.
- A **repository** where the extracted text is stored.
- A **webserver** used to access to the data stored in the repository.

The processor component can be customized for different alphabets. We customized one instance of processor for Latin characters and another suitable to read Korean characters.

The deployment of the application is made through the DECENTER Fog Platform with the two
instances of processors deployed locally to the corresponding cameras (European and Korean ones) and with the repository and the webserver located in the European cluster. Thanks to this deployment topology and to the setup of the VPN between the two clusters, only relevant and aggregated information leave the cluster where they are produced (the original images are maintained locally) and the cluster-to-cluster communication (i.e., cross border) is secured and encrypted.
3 Analysis of Security/Robustness solutions

This section provides the final and detailed analysis of the solutions proposed with the goal of improving the security of microservices and fog nodes. Additionally, it includes a detailed description of robustness of the DECENTER fog computing platform.

3.1 Security of the microservices

The microservices are not exempt from possible cyberattacks or different cyber threats, which is the main reason why we try to protect them and give the necessary tools to inform if there are any unusual behaviours on the microservices. Task T3.4 has improved and adapted the solution called L-ADS (Live Anomaly Detection System). This asset tries to identify which connections could be anomalous in a certain environment. However, the initial version of the asset [1] was developed using a Machine Learning algorithm called One Class Support Vector Machine and it has been adapted to detect anomalies on the microservices using a deep learning algorithm called Autoencoder.

The asset L-ADS was introduced in D3.3. It includes the data acquisition, the architecture of the solution, the two versions of the asset, the DL model used to classify the connections and a testing scenario where the asset was able to correctly detect more than the 90% of all connections. In this section we aim to explain in detail the analysis of the CIDDS (Coburg Intrusion Detection Data Sets) dataset presented in D3.3 and how the L-ADS can classify the anomalous connections.

The tool Softflowd is an implementation of the protocol developed by Cisco called Netflow and is the responsible to capture the flows of the traffic too. The data provided by Softflowd has the Netflow structure, it contains the following variables:

- **Date first seen.** Is the start time of the connection.
- **Duration.** Is the total time of the connection.
- **Proto.** Is the protocol used in the connection.
- **Src IP Addr.** Is the source IP address.
- **Src Pt.** Is the source port.
- **Dst IP Addr.** Is the destination IP address.
- **Dst Pt.** Is the destination Port.
- **Flags.** Are the TCP flags of the connection.
- **Packets.** Are the number of packets.
- **Bytes.** Are the number of bytes.
- **Tos.** Is the type of service.
- **Flows.** Are the number of flows in the connection.

Unlike the D3.3, in this deliverable we explain the exploratory data analysis (EDA) that was followed before the DL model development. In addition, the DL model development will be described in more details below.

3.1.1 Exploratory Data Analysis (EDA)

EDA is a process that aims to obtain information analysing the dataset using visualization methods and statistical summaries. As we have explained, we are using the CIDDS dataset,
D3.5: Methods and solutions to achieve security and robustness

Figure 2 is a sample of it, where each row of the dataset corresponds with a connection and the columns are the variables that characterize the connections.

<table>
<thead>
<tr>
<th>Date first seen</th>
<th>Duration</th>
<th>Proto</th>
<th>Src IP Addr</th>
<th>Src Pt</th>
<th>Dst IP Addr</th>
<th>Dst Pt</th>
<th>Packets</th>
<th>Bytes</th>
<th>Flows</th>
<th>Flags</th>
<th>Tos</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-04-12 18:09:18.263</td>
<td>0.002</td>
<td>TCP</td>
<td>192.168.220.12</td>
<td>58447.0</td>
<td>192.168.100.5</td>
<td>440.0</td>
<td>2</td>
<td>17.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-13 02:21:32.127</td>
<td>0.000</td>
<td>TCP</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>192.168.100.7</td>
<td>51227.0</td>
<td>1</td>
<td>7.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-12 12:02:25.045</td>
<td>0.422</td>
<td>TCP</td>
<td>192.168.220.14</td>
<td>37619.6</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>307</td>
<td>1100000.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-13 06:41:24.314</td>
<td>0.000</td>
<td>TCP</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>192.168.220.4</td>
<td>40155.0</td>
<td>1</td>
<td>10.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-17 21:31:15.781</td>
<td>0.000</td>
<td>TCP</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>192.168.220.5</td>
<td>59792.0</td>
<td>1</td>
<td>108.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-19 08:07:47.245</td>
<td>0.000</td>
<td>TCP</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>192.168.220.8</td>
<td>36967.0</td>
<td>1</td>
<td>10.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-16 23:28:44.110</td>
<td>0.000</td>
<td>TCP</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>192.168.220.9</td>
<td>39356.0</td>
<td>1</td>
<td>108.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
<tr>
<td>2017-04-15 00:30:49.002</td>
<td>0.000</td>
<td>TCP</td>
<td>192.168.100.5</td>
<td>445.0</td>
<td>192.168.220.4</td>
<td>57828.0</td>
<td>1</td>
<td>108.0</td>
<td>1</td>
<td>&quot;AP_&quot;</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2: Example of CIDDS dataset.

The first step during the EDA is to identify the different kinds of variables in the dataset. In our case, the variables are categorized as following:

- **Temporal variable.** The *Date first seen* variable shows the date when the connection has been started.
- **Identifier variables.** They identify what are the sender and the receiver in the connection, in this case *Src IP Addr* and *Dst IP Addr*.
- **Categorical variables.** This kind of variables are: *Proto*, *Flags* and *Tos*.
- **Numeric variables.** In this dataset most variables are numeric such as: *Duration, Src Pt, Dst Pt, Packets, Bytes and Flows*.

Due to this categorization of the variables, the processes to analyse the data in those variables are different too.

The current version of the L-ADS does not use the temporal variable as a result of the promising performance of the asset without this temporal variable. Perhaps, in other environments with different datasets would be necessary to introduce it.

The identifier variables not only provide information of what are the sender and the receiver in the connection. If we use the proper graphical tools, we can obtain information of the network topology. The following flow chart (Figure 3) shows in the left side the source IP addresses and in the right side the destination IP addresses. In addition, the lines width depends on the cumulative number of bytes between the source IP and the destination IP.
D3.5: Methods and solutions to achieve security and robustness

Figure 3: Flow Chart of Src IP and Dst IP.

Thanks to this plot, we observe the IP 192.168.100.5 is the IP that sends and receives the largest quantity of bytes. Due to this fact, it allows us to suspect that this IP could be one of the core nodes in the network. Figure 4 shows a representation of the network. Our suspicions were correct, we can observe one of the core nodes is the IP 192.168.100.5.

Figure 4: Topology of the CIDDS network.
D3.5: Methods and solutions to achieve security and robustness

Using these plots, we are able to understand how is the network topology and in the case of suffering an attack, which nodes (or IPs) are critical to protect, and which ones are less important to protect.

About the categorical variables, the simplest plot to show an overview of these variables is a bar plot with the number of occurrences per each category. Figure 5 shows the variable *Proto*.

![Figure 5: Categorical Variable – Proto.](image)

The Figure 6 shows the variable *Flags*.

![Figure 6: Categorical Variable - TCP Flags.](image)
And the variable Tos (Figure 7)

These bar plots let us know about the proportions of each category in the different categorical variables. As we can observe, Figure 5 shows the proportion of protocols and the most commons are TCP and UDP. About the TCP Flags (Figure 6), three (.AP... , ...., .A....) of the thirteen TCP Flags are the majority of the categories. And variable Tos (Figure 7), the value 0 is prevalent in most connections.

To conclude the EDA, the numeric variables can provide information using a statistical summary (Figure 8).

The variable Flows has as minimum the value 1 and it is also the maximum value in this variable. So that, we can eliminate this variable because it only has the value 1, it does not provide information to the algorithm. For the rest of variables, it can be interesting show the correlation between them (Figure 9). However, with these results we cannot draw more conclusions.
3.1.2 Pre-process and DL Model

In deliverable D3.3, we focussed in the two different versions of the L-ADS (see section 7.1.4 of D3.3). On the one hand, L-ADS version 1 can model the problem, generate a single dataset without the importance of the source IP and the destination IP and use a single algorithm to classify the new input connections. L-ADS version 2, on the other hand, can model the problem for each IP, generate a single dataset per IP and the algorithm associated to this IP. Consequently, the second version of the L-ADS considers adding the importance of the IP addresses. However, we did not detail the transformation necessary to obtain the dataset.

During the pre-processing, the initial dataset is transformed to obtain a dataset prepared to the training phase. At this point, the dataset must be complete (without missing data) and all the values on the dataset must be numeric.

The pre-process technique has the same structure as the previous subsection EDA, it will be different depending the kind of variables.

- The temporal variable *Date first seen* has not been modified because it will not be included in the algorithm. This variable will be dropped in the pre-processed dataset because we have not seen any evidence of improvement in the algorithm.
- The identifier variables *Src IP Addr* and *Dst IP Addr* will be dropped in L-ADS version 1 and in L-ADS version 2, they will be used to filter the dataset for each IP and generate the variable *is_source*. After obtaining the filtered dataset for a certain IP, the identifier variables will be removed from the dataset according to the same procedure of L-ADS version 1.
- The numeric variables are standardized to placing them on the same scale. This technique subtracts the mean and divides the values by the standard deviation of each numeric variable. Table 2 shows the standardization in variable *Bytes*. It could be

![Figure 9: Correlation of numeric variables.](image-url)
D3.5: Methods and solutions to achieve security and robustness

useful if we receive the dataset in different scales such as MB (Megabytes), GB (Gigabytes), etc.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>-0.51</td>
</tr>
<tr>
<td>10</td>
<td>-0.64</td>
</tr>
<tr>
<td>108</td>
<td>1.15</td>
</tr>
</tbody>
</table>

In addition, the L-ADS creates four new numeric variables using the initial dataset. They are:

- **Packets_speed.** Packets/Duration. If Duration = 0, Packets_speed = −1
- **Bytes_speed.** Bytes/Duration. If Duration = 0, Bytes_speed = −1
- **Packets_per_flow.** Packets/Flows
- **Bytes_per_flow.** Bytes/Flows

In this case, the last two will not be generated because we dropped the variable *Flows*, it only contains the value 1 and it does not provide any information.

- The categorical variables (*Proto*, *Flags*, *Tos*) are transformed into dummies variables. This procedure transforms a categorical variable generating binary variables one per each categoryTable 3 is an example using the variable *Proto*.

<table>
<thead>
<tr>
<th>Proto</th>
<th>Proto_TCP</th>
<th>Proto_UDP</th>
<th>Proto_ICMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TCP</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ICMP</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

At this point, the L-ADS has pre-processed the dataset to be ready as input in the algorithm.

The algorithm used in the development of the asset L-ADS is the Autoencoder. This kind of neural network has a certain structure, it has an encoder and a decoder, the architecture of the Autoencoder is represented in Figure 10. Those parts let the neural network compress and decompress the input, aiming to learn how to reconstruct the initial connections. If the algorithm is only train using legit traffic (without attacks), it will learn how to reconstruct normal traffic and in the other side, the reconstructed anomalous connections will be so different with the original anomalous connection. That is the reason to train the algorithm only with normal traffic. Furthermore, the layers of the autoencoder are fully connected RNN (Recurrent Neural Network) and the activation function used in the neural network is “ReLU”. The autoencoder algorithm is trained during 500 epochs. An epoch is the number of complete passes through the training dataset.
The mean squared error (MSE) function (MSE) was introduced in D3.3.

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - Y'_i)^2,
\]

where \( n \) is the number of features, \( Y_i \) is the value of the i-th feature in the connection and \( Y'_i \) is the value of the i-th feature in the reconstructed connection.

The MSE function generates a value for each new connection. This new input connection is evaluated by the autoencoder and then, the MSE function calculates a value that we can consider it as a “normality” value. If the value obtained by the MSE function is higher than a specific threshold, the connection will be categorized as anomalous. The MSE function generates a representation like the following Figure 11.
In our case, the threshold is the blue line in Figure 11. The choice of setting the threshold is one of the critical points during the modelling, it affects directly with the performance obtained by the algorithm. In the case of we are trying to classify a supervised learning problem where the dataset is labelled, the choice of threshold is done using the labels with the aim to obtain the best performance of the algorithm. Otherwise, if the asset is trying to solve an unsupervised learning problem (without labels), the choice of threshold could be with the information extracted of the supervised learning problem if it is possible or otherwise, the threshold is established by experimentation.

3.2 Security of the Fog nodes

Fog nodes are exposed to various cyber-attacks that aim at degrading or preventing their normal operations [7]. In this task, we have investigated how to protect the Fog nodes from different types of network intrusions, starting from Distributed Denial of Service (DDoS) attacks (the results of this study have been published in international journals [9][17]), to other common attacks such as port scans, brute force attacks, data injections, etc.

As detailed in Deliverable D3.3 [8], the objective of this study is to design and implement a lightweight Intrusion Detection and Prevention System (IDPS) that can be executed directly on the Fog nodes. As Fog nodes might possess a limited amount of resources (CPU, memory and storage), the main challenge of this work is to find a good trade-off between attack detection accuracy and resource consumption.

In this Section we introduce the workflow of IDPS, with particular attention on the Deep Learning (DL) traffic classification process. Hence, we detail the layers of the neural network architecture and how it has been tuned to maximise the attack detection accuracy.

3.2.1 System architecture

The overall workflow of the intrusion detection and mitigation is represented in Figure 12. From the left-hand side, the incoming packets are filtered based on the content of a blacklist. When the list is empty, all the packets pass through the filter towards the traffic pre-processing and classification modules. Once the blacklist starts being populated, each packet is compared against the content of the list. The matching packets are dropped. The packet filtering program is implemented in the kernel space of the Fog node’s operating system using eBPF and XDP technologies [10][11][12], available in the kernel of recent Linux Operating systems (kernel version 4.15 and above). Filtering the network traffic in directly the kernel space has two main advantages: (i) syscalls overhead and kernel/user space context switching are avoided, (ii) early malicious traffic filtering before the TCP/IP stack of the machine’s operating system are considered. More details on the packet filtering process are provided in D3.3 and in the research entitled Introducing SmartNICs in server-based data plane processing: The DDoS mitigation use case [17].
D3.5: Methods and solutions to achieve security and robustness

The traffic pre-processing mechanism groups the packets into flows represented in array-like structures of size $nxf$. The $n$ lines are the packets of a flow in chronological order (from top to bottom), while each of the $f$ columns is a packet header’s attribute. Each flow is then processed by Convolutional Neural Network (CNN) composed of Convolutional layer with $k$ filters of height $h$, a MaxPooling layer and a final Classification layer.

The Softmax activation function applied to the classification layer returns a probability score for each of the traffic classes (those available in the training set such as, benign traffic, DDoS attack, SQL injection, etc., presented in Section 3.2.6). The class with the highest probability is assigned to the input flow. If the flow is not benign, its identifier is added to the blacklist.

3.2.2 Convolutional layer

Each input flow is operated by a single convolutional layer with $k$ filters of height $h$ and width equal to the number of packet features $f$. Each filter, also known as a kernel or sliding window, convolves over the input flow with a step of 1 to extract and learn local features that contain useful information for the detection of malicious and benign flows. Each of the $k$ filters generates an activation map $a$ of size $n - h + 1$, such that

$$a_k = ReLU(Conv(F,W_k,b_k))$$

where $W_k, b_k$ are the weight and bias parameters of the $k$-th filter respectively, that are learned during the training stage. To introduce non-linearity among the learned filters, we use the rectified linear activation function $ReLU(x) = \max\{0, x\}$, as per convention for CNNs. All activation maps are stacked, creating an activation matrix $A$ of size $(n - h + 1) \times k$.

There are two main benefits of including a CNN in our architecture. Firstly, it allows the model to benefit from efficiency gains compared to standard neural networks, since the weights in each filter are reused across the whole input. Sharing weights, instead of the full end-to-end connectivity with a standard neural net, makes the model more lightweight and reduces its memory footprint as the number of learnable parameters is greatly reduced. Secondly, during the training phase, the CNN automatically learns the weights and biases of each filter such that the learning of salient characteristics and features is encapsulated inside the resulting model during training. This reduces the time-consuming feature engineering and ranking involved in statistical and traditional machine learning methods, which relies on expert human knowledge.
3.2.3 MaxPooling layer

For max pooling, we down-sample along the first dimension of the activation matrix $A$, which represents the temporal nature of the input. A pool of size $m$ produces an output matrix of size $(n - h + 1)/m \times k$, which contains the largest $m$ activations of each learned filter. In this way, the model disregards the less useful information that produced smaller activations, instead of paying attention to the larger activations. This also means that we dispose of the positional information of the activation, i.e., where it occurred in the original flow, giving a more compressed feature encoding, and, in turn, reducing the complexity of the network.

3.2.4 Classification layer

A dense layer takes the vector of size $k$ activations and returns a vector $v$ of size $N$, where $N$ is the number of traffic classes. Vector $v$ is passed to the Softmax activation function, which turns $v$ into a vector of probabilities whose elements sum to one. As a final step, the classification process selects the class with the highest probability and assigns it to the input flow.

3.2.5 Hyper-parameters tuning

Tuning the hyper-parameters is an important step to optimise the model's accuracy, as their values influence the model complexity and the learning process. This operation has been executed at the beginning of this study on the security of Fog nodes, where we only focused on DDoS attacks [9]. Nevertheless, we re-use those results to also tune the final architecture (Figure 12) that we designed to support all the traffic classes presented in Section 3.2.6.

Hence, based on the analysis presented in [9], we set the values of $n$ (max number of packet/flow), $k$ (number of convolutional filters), $h$ (height of the filters) and $m$ (pool size) to 100, 64, 3 and 98 respectively. The pool size equals to 98, indicates that for each filter’s output activation map, we keep only the activation with the highest value.

3.2.6 Datasets

Our deep learning models have been validated with benign and malicious network traffic provided by the Canadian Institute for Cybersecurity of the University of New Brunswick (UNB) [13]. We used the datasets ISCX2012 [14], CICIDS2017 [15] and CICIDS2018 [16] to train and assess the neural network in terms of classification accuracy and inference time. UNB researchers generated these datasets based on traffic profiles to accurately represent the abstract properties of human and attack behaviours. One profile characterises the normal network activities HTTP, SMTP, SSH, IMAP, POP3, and FTP), other profiles describe a variety of attack scenarios based on recent security reports.

In our initial experiments, we used the benign and DDoS traffic profiles from all the three datasets to tune the hyper-parameters and to evaluate the performance of the approach on such a common cyber-attack. Table 4 shows the parts of the three datasets used in this work. In the table, the column Traffic trace specifies the name of the trace, according to [14], [15] and [16]. Specifically, the ISCX2012-Tue15 trace contains a DDoS attack based on an IRC botnet. The CIC2017-Fri7PM trace contains a HTTP DDoS generated with LOIC, while the CSECIC2018- Wed21 trace contains a HTTP DDoS generated with HOIC. With respect to the
D3.5: Methods and solutions to achieve security and robustness

original file, the trace CIC2017-Fri7PM is reduced to timeslot 3.30PM-5.00PM to exclude malicious packets related to other cyber-attacks (port scans and back-doors).

Table 4: Summary of the DDoS datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Traffic trace</th>
<th>#Flows</th>
<th>#Benign</th>
<th>#DDoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISCX2012</td>
<td>Tue15</td>
<td>571698</td>
<td>534320</td>
<td>37378</td>
</tr>
<tr>
<td>CICIDS2017</td>
<td>Fri7PM</td>
<td>225745</td>
<td>97718</td>
<td>128027</td>
</tr>
<tr>
<td>CICIDS2018</td>
<td>Wed21</td>
<td>1048575</td>
<td>360832</td>
<td>687743</td>
</tr>
</tbody>
</table>

The flows of the three datasets have been balanced in order to have the same number of benign and DDoS flows, with the same number of DDoS flows extracted from each dataset. The resulting merged dataset (that we called UNB201X) of ~224000 flows have been shuffled and divided into training (90%) and test (10%) sets, with 10% of the training set used for validation during the hyper-parameter tuning process.

We have also used the other attacks of the CICIDS2017 dataset to validate the system. Besides DDoS and benign traffic, CICIDS2017 includes other common network malicious activities such as brute force and port scan attacks, the Command & Control traffic of a botnet and others listed in Table 5. CICIDS2017 has been split into training (90%) and test (10%) sets, and by ensuring that the training set contains the 90% of the samples of each class. The training set is only used for training the model, while the test set is used for the evaluation presented in Section 4.2. The traffic classes have not been balanced, due to the small amount of samples available for the Botnet and Web Attack classes.

Table 5: Summary of the CICIDS2017 dataset.

<table>
<thead>
<tr>
<th>Traffic class</th>
<th>#Flows</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>747148</td>
<td>Synthetic benign traffic produced using the profiles of normal network activities captured from the UNB network. It includes distribution models for applications and protocols, such as HTTP(S), SMTP, SSH, IMAP, POP3, FTP, NetBios and DNS.</td>
</tr>
<tr>
<td>DoS/DDoS</td>
<td>365601</td>
<td>The CICIDS2017 dataset contains DoS and DDoS traffic generated using publicly available attacking tools such as Hulk, GoldenEye, Slowloris, Slowhttptest, Heartleech and LOIC.</td>
</tr>
<tr>
<td>FTP Patator</td>
<td>12289</td>
<td>Brute force password-guessing attack against File Transfer Protocol (FTP) servers.</td>
</tr>
<tr>
<td>SSH Patator</td>
<td>11808</td>
<td>Brute force password-guessing attack against Secure Shell (SSH) servers.</td>
</tr>
<tr>
<td>Port Scan</td>
<td>160471</td>
<td>Port scan attack against a number of machines with network scanner Nmap.</td>
</tr>
<tr>
<td>Botnet</td>
<td>972</td>
<td>Communication between a Command &amp; Control (C&amp;C) server and a number of victim machines (bots) infected with malware. The traffic includes remote shell, file upload/download, key logging and other activities.</td>
</tr>
<tr>
<td>Infiltration</td>
<td>65218</td>
<td>Infiltration attempts to gain unauthorised access to the victim machine. The attack traffic is generated using Nmap and Metasploit used to discover the vulnerabilities of the victim machine, previously infected with malware.</td>
</tr>
<tr>
<td>Web Attack</td>
<td>3861</td>
<td>XSS, SQL injection and brute force attacks against a vulnerable PHP/MySQL web application and generated using the Selenium framework.</td>
</tr>
</tbody>
</table>
The dataset is split into training set (90%, 1230631 samples) and test set (10%, 136737 samples), with 10% of the training set used for validation.

### 3.3 Robustness of the platform

Concerning robustness aspects, the task T3.4 has worked to make the Fog Computing Platform more robust and resilient. Kubernetes is already well suited for traditional cloud and data centres. But considering more distributed and heterogeneous environments, like the ones foreseen by the fog computing paradigm, failures in the network connectivity among the different nodes of a given cluster become more relevant and can tamper the cluster stability. In these environments, having one cluster spread across different geographical locations can increase the risk of failures due to connectivity problems between the master and the workers nodes. Instead, having many Kubernetes clusters each covering a single geographical region, allows reconducting the problem of management of faults to many clusters located in the traditional data centres, simplifying the overall complexity.

As explained in D3.3 and D3.4, one of the major limitations of using a single distributed Kubernetes cluster in a fog context is that the control plane is centralized in one region and potential faults to the network between the region hosting the control plane and the other regions can disrupt the entire cluster. More generally the single cluster architecture is distributed but not decentralized limiting the resilience to the faults of the entire system. In order to make the Fog Platform more decentralized and resilient, we propose an evolution of the architecture adopting the concept of federation: each region corresponds to a cluster of the Fog Platform (i.e., a Kubernetes cluster), independent but federated with the other clusters. The pattern chosen to implement such a federation of clusters leverages the Kubernetes Cluster Federation (KubeFed) project [26][26]. The following figures show a comparison between the single cluster and the federated multi-cluster architecture.

![Single cluster architecture](image-url)

*Figure 13: Single cluster architecture.*
In Figure 13, a single cluster is depicted with a Master Region where the Kubernetes and Fog Platform control plane are installed. In Figure 14, a federated multi-cluster configuration is shown: the Master Region hosts the Kubernetes federation control plane and the Fog Platform federation control plane, while the regions (Region 1 and Region 2) are traditional Kubernetes clusters hosting the Kubernetes and Fog Platform control planes. Moreover, as Figure 13 a cluster can have sub-regions, Region 2 has sub-regions Sub Region 1 and Sub Region 2. Therefore, following the KubeFed naming convention, in case of Figure 14 we have one Host Cluster (Master Region) and two Member Clusters (Region 1 and Region 2).

With this solution the architecture is decentralized, and robustness and resilience are improved: even though the Host Cluster experiences a failure, the different Member Clusters can continue to operate and process the local workload.

In this scenario two different levels of workload orchestration are employed: one at the federation level, in order to select the cluster candidate to host a given workload, and another at cluster level in order to select the sub-region/worker nodes where to deploy the workload.
4 Implementation and Evaluation in the Fog Computing Platform

This section explains what the steps are following to complete the implementation of all solution in the DECENTER platform. Details of the integration are also reported more precisely in D5.2 [28]. Additionally, this section gives information related to confirm the validity of the proposed solutions.

4.1 Security of the microservices

4.1.1 Implementation of the solution L-ADS

The implementation of the solution (L-ADS) in the Fog Computing Platform has been a complex process. Initially, the asset presented in [1] was developed using a virtual machine. Due to the Fog Computing Platform is based on Kubernetes, one of the best options to complete the integration was containerized the L-ADS asset as a docker container. With this migration the L-ADS would be easier to deploy in different environments.

The L-ADS solution is composed by two applications: **Softflowd** and **L-ADS**. These two applications have been “dockerized” so that they can be deployed in container orchestrators like Kubernetes or Docker. In this case both L-ADS and Softflowd have been deployed in a Kubernetes environment (Figure 15).

The tool Softflowd is listening the traffic of the rest of the pods and it sends the traffic through the port 9000. In addition, Softflowd sends the traffic using the Netflow structure that the L-ADS is able to read and use as input of the Autoencoder. To do that, the L-ADS application needs to expose the port 9000, which is used by the Softflowd application to send there all the information gathered from the network where this application is listening. As both applications are tightly coupled, they have been defined in the same “pod” definition used to deploy them in Kubernetes:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    io.kompose.service: lads-softflowd
name: lads-softflowd
namespace: decenter
spec:
  containers:
  - image: atosdecenter/lads
    name: lads
    ports:
    - containerPort: 9000
```

Figure 15: L-ADS in Kubernetes.
4.1.2 Evaluation

The Robotnik’s Use Case (UC2) of DECENTER has been developed aiming to improve the logistics of the robots in the DECENTER platform. L-ADS was integrated for the purpose of provide security on the microservices which are running in the UC2.

The following Figure 16 shows the proper functioning of the L-ADS on the UC2.
D3.5: Methods and solutions to achieve security and robustness

Figure 16: L-ADS logs in the UC2.

To better understand the process following by the tool, the Figure 17 represents the workflow of the L-ADS in the UC2.
Firstly, the L-ADS receive data from the different pods that are running in the UC2 by the tool Softflowd. The process of data-capturing lasted for 2 hours. After that, the L-ADS trains the deep learning algorithm (Autoencoder) using the captured data from the pods. And finally, the L-ADS is ready to start making the predictions for the new input data received by Softflowd.

During the test, the L-ADS did not detect any anomaly. This could be due to fact that UC2 is a controlled environment where there are no attacks during the simulation. Additionally, in the absence of a labelled dataset with different attacks, the threshold estimation is a complicated process.

The L-ADS was proposed as a solution to detect possible anomalies in a certain environment. However, it is difficult find a valid dataset which could be truly represent the connections such as the UC explained above.

The following Table 6 shows the results using the CIDDS dataset, using the two versions of L-ADS.

<table>
<thead>
<tr>
<th>L-ADS Version</th>
<th>Accuracy</th>
<th>F1-Score (Benign)</th>
<th>F1-Score (Anomalous)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Supervised)</td>
<td>0.97</td>
<td>0.78</td>
<td>0.33</td>
</tr>
<tr>
<td>2 (Supervised)</td>
<td>0.94</td>
<td>0.81</td>
<td>0.96</td>
</tr>
<tr>
<td>1 (Unsupervised)</td>
<td>0.91</td>
<td>0.86</td>
<td>0.28</td>
</tr>
<tr>
<td>2 (Unsupervised)</td>
<td>0.94</td>
<td>0.81</td>
<td>0.96</td>
</tr>
</tbody>
</table>

In order to get a labelled dataset, we created a script that modifies some values of the dataset that it is used as input of the algorithm. More precisely, the script modifies the 10% of connections of the dataset, and each modified connection could change one or two columns.
D3.5: Methods and solutions to achieve security and robustness

(it depends on a 50% likelihood). If the chosen column by the script is numeric, the initial values are changed by the 95th percentile of the same column. Otherwise, if the chosen column is categorical, the values are changed by the category with less occurrences.

With the help of this script, we analyse the performance of the L-ADS in the UC2. The results obtained (Table 7) are better than those obtained using the CIDDS.

Table 7: L-ADS performance using the script on the UC2 connections.

<table>
<thead>
<tr>
<th>L-ADS Version</th>
<th>Accuracy</th>
<th>F1-Score (Benign)</th>
<th>F1-Score (Anomalous)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Supervised)</td>
<td>0.98</td>
<td>0.99</td>
<td>0.84</td>
</tr>
<tr>
<td>2 (Supervised)</td>
<td>0.99</td>
<td>0.99</td>
<td>0.98</td>
</tr>
</tbody>
</table>

There is, therefore, a reasonable prospect that the results are not conclusive by the fact that the script does not modify the connections in the same way as the behaviour of any attack.

4.2 Security of the Fog nodes

In this section, we introduce the IDPS implementation details and we present a detailed evaluation of the proposed approach with the datasets presented in Section 3.2.6. Evaluation metrics of Precision, Recall and F1 Score have been used for performance measurement and for comparison with state-of-the-art model.

4.2.1 Implementation

As per convention in the literature, we report the metrics Precision (or Positive Predictive Value (PPV)), Recall (or True Positive Rate (TPR)) and F1 Score (F1), with a focus on the latter. PPV is the ratio between the correctly detected DDoS samples and all the detected DDoS samples (true and false). TPR represents the percentage of DDoS samples that are correctly classified as such. The F1 Score is an overall measure of a model’s performance; that is the harmonic mean of the PPV and TPR. These metrics are formally defined as follows:

\[
PPV = \frac{TP}{TP + FP} \quad TPR = \frac{TP}{TP + FN} \quad F1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR}
\]

where TP=True Positives, FP=False Positives, FN=False Negatives.

The IDPS model has been implemented in Python v3.6 using the Keras API v2.2.4 [24] on top of Tensorflow 1.13.1 [25].

4.2.2 Detection accuracy

In order to validate our approach, we measure the performance of the proposed IDPS in classifying unseen traffic flows as benign or malicious. Table 8 and Table 9 summarize the results obtained on the various test sets. As illustrated, the very high performance is maintained across the range of test datasets indicating the robustness of the system design.

Table 8: DDoS detection performance.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PPV</th>
<th>TPR</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISCX2012</td>
<td>0.9827</td>
<td>0.9952</td>
<td>0.9889</td>
</tr>
<tr>
<td>CICIDS2017</td>
<td>0.9939</td>
<td>0.9994</td>
<td>0.9966</td>
</tr>
<tr>
<td>CICIDS2018</td>
<td>0.9984</td>
<td>0.9989</td>
<td>0.9987</td>
</tr>
<tr>
<td>UNB201X</td>
<td>0.9914</td>
<td>0.9979</td>
<td>0.9946</td>
</tr>
</tbody>
</table>
Table 9: Intrusion detection performance on the CICIDS2017 test set.

<table>
<thead>
<tr>
<th>Traffic class</th>
<th>PPV</th>
<th>TPR</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>0.9922</td>
<td>0.9895</td>
<td>0.9908</td>
</tr>
<tr>
<td>DoS/DDoS</td>
<td>0.9790</td>
<td>0.9861</td>
<td>0.9825</td>
</tr>
<tr>
<td>FTP Patator</td>
<td>0.9919</td>
<td>0.9959</td>
<td>0.9939</td>
</tr>
<tr>
<td>SSH Patator</td>
<td>0.9957</td>
<td>0.9881</td>
<td>0.9919</td>
</tr>
<tr>
<td>Port Scan</td>
<td>0.9969</td>
<td>0.9973</td>
<td>0.9971</td>
</tr>
<tr>
<td>Botnet</td>
<td>0.9293</td>
<td>0.9485</td>
<td>0.9388</td>
</tr>
<tr>
<td>Infiltration</td>
<td>0.9963</td>
<td>0.9816</td>
<td>0.9889</td>
</tr>
<tr>
<td>Web Attack</td>
<td>0.8846</td>
<td>0.9534</td>
<td>0.9177</td>
</tr>
<tr>
<td><strong>Weighted Avg</strong></td>
<td><strong>0.9891</strong></td>
<td><strong>0.9890</strong></td>
<td><strong>0.9890</strong></td>
</tr>
</tbody>
</table>

Results show that thanks to the properties of its CNN, the proposed model learns to distinguish between patterns of malicious behaviour and benign flows. Regarding the selected metrics, only Botnet and Web Attack classes return slightly low scores. This is due to the scarce amount of data available for these two types of intrusions (Table 5).

Given the properties of convolutional methods, these patterns are recognised regardless of the position they occupy in a flow, demonstrating that our spatial representation of a flow is robust. Irrespective of whether the attack event appears at the start or the end of the input, the CNN will produce the same representation in its output. Although the temporal dynamics in network attacks might suggest that alternative DL architectures may seem more suitable (e.g. Long Short-Term Memory (LSTM)), our novel pre-processing method combined with the CNN removes the requirement for the model to maintain temporal context of each whole flow as the data is pushed through the network. In comparison, LSTMs are known to be very difficult to train, and their performance is inherently slower for long sequences compared to CNNs.

4.2.3 Intrusion detection at the edge

In this section, we demonstrate that our intrusion detection solution can be deployed and effectively executed on devices with limited computing and memory resources, by running the DL model on an NVIDIA Jetson TX2 development board [20], equipped with a quad-core ARM Cortex-A57@2GHz CPU, 8 GB of RAM and a 256-core Pascal@1300 MHz Graphics Processing Unit (GPU). For the experiments, we used Tensorflow 1.9.0 with GPU support enabled by cuDNN, a GPU-accelerated library for deep neural networks [21].

We analyse the applicability of our approach to online edge computing environments by estimating the prediction performance in terms of samples processed per second. As we are aware that edge nodes do not necessarily mount a GPU device, we conduct the experiments with and without the GPU support on the UNB201X test set and discuss the results.

With respect to this, one relevant parameter is the batch size, which configures how many samples are processed by the CNN in parallel at each iteration. Such a parameter influences the speed of the detection, as it determines the number of iterations and, consequently, the number of memory reads required by the CNN to process all the samples in the test set (or the samples collected in a time window, in the case of online detection).
D3.5: Methods and solutions to achieve security and robustness

Figure 18: Inference performance on the NVIDIA TX2 board.

Figure 18 shows the performance of the CNN on the development board in terms of processed samples/second. As the shape of each sample is \([n, f] = [100, 11]\), i.e. each sample can contain the features of up to 100 packets, we can estimate that the maximum number of packets per second (pps) that the device can process without the GPU and using a batch size of 1024 samples is approximately 1.9 Mpps. As an example, the content of the UNB201X test set is 602,547 packets distributed over around 22400 samples, which represents a processing requirement of 500 Kpps without the GPU, and 600 Kpps when the GPU is enabled.

The second measurement regarding resource-constrained systems is the memory requirement to store all the samples collected over a time window. The memory occupancy per sample is 8,800 bytes, i.e. 100-11 = 1100 floating point values of 8 bytes each. As per Figure 18, the CNN can process around 23K samples/second with the help of the GPU and using a batch size of 1024. To cope with such a processing speed, the device could require up to 20GB RAM. However, this value greatly exceeds the typical amount of memory available on edge nodes, in general (e.g., 1 GB on Raspberry Pi 3 [22], 2 GB on the ODROID- XU board [23]), and on our device, in particular. Indeed, the memory resources of nodes can represent the real bottleneck in an edge computing scenario.

Therefore, assuming that our edge node is equipped with 1GB RAM, the maximum number of samples that can be stored in RAM is approximately 100K (without considering RAM used by the operating system and applications).

We have calculated that this memory size would be sufficient for an attack such as the HTTP-based DDoS attack in the CICIDS2018 dataset, for which we measured approximately 30K samples. For more aggressive attacks, however, a strategy to overcome the memory limitation would be to configure the CNN model with lower value \(n\), hence lower memory footprint for each sample. For instance, setting the value to 10 can reduce the memory requirement by a factor of 10, with a low cost in detection accuracy (F1 score 0.9928 on the UNB201X test set, compared to the highest score obtained with \(n = 100\), i.e. 0.9946).

The measurements based on our test datasets demonstrate that the proposed DL-based IDPS is usable on resource-constrained devices such as the Fog nodes both with respect to processing and memory requirements. These results are promising for effective deployment.
D3.5: Methods and solutions to achieve security and robustness of the IDPS in a variety of edge computing scenarios, including those where the nodes execute latency-sensitive services.

4.3 **Robustness of the platform**

The implementation of the federated solution sketched in D3.3 requests an extension of the Fog Platform orchestrator considering two level of orchestration. Such an extension is based on Kubernetes Custom Resource Definition (CRD): the CRD *FedFAApp* is the representation of a “federated” cloud-native application composed by a set of application pieces, called “application chunks”, and data flows between these different chunks. Each chunk is a set of one or more microservices that should be placed on a given cluster of the federation. Such a placement is managed by the Fog Platform **federated** orchestrator. An application chunk is represented by a federated version of the *FADepl* CRD (see D3.3 and D3.4) that in turn is ingested by the corresponding Fog Platform orchestrator for a fine-tuned and intelligent (based on resource availability and constraints imposed) placement inside the selected cluster. Figure 19: *FedFAApp* model shows the *FedFAApp* models.

Finally, the applicative communication between the different clusters has been implemented by Istio service mesh [27].
5 Conclusion

In this report, we have described the main achievements performed of the task T3.4 during the year 3 of the DECENTER project. The report presents the work and the investigations done by the different involved partners and it accomplish the proposed objectives for the last two years of the DECENTER project.

As a result of this work, it has been investigated on different security concerns to ensure the security on federated interactions when there are multiples providers. Additionally, there is a real-life scenario where it is guaranteeing the data transfers between clusters when it is a communication established.

As consequence of the final and detailed analysis of the tools, it has been developed two solutions based on DL models. The security of the microservices solution is the L-ADS (Live Anomaly Detection System), it has been tested using two different datasets (CIDDS and the UC2 dataset that has been modified by the script). Performance results obtained by the asset make us believe that L-ADS classifies connections in a certain environment with an accuracy of more than 94 percent. About the solution to ensure the security of the fog nodes, it has been tested in four different dataset and it has obtained results that are promising. The performance of the algorithm was computed using different metrics such as the PPV, TPR and F1-Score but we can highlight the minimum value of F1-Score in all dataset was 0.9889. All this leads to a solution with a high capacity to correctly classify unseen connections with the aim to distinguish if those connections are benign or anomalous.

Finally, the report describes the robustness solution of the DECENTER platform that has been previously reported in D3.3 and D3.4. It guarantees the robustness against failures in a decentralized cloud infrastructure. In this case, all the solutions have been implemented in the DECENTER fog computing platform.
6 References


[8] DECENTER D3.3. “Second release of the fog computing platform”


Methods and solutions to achieve security and robustness


[27] Istio, https://istio.io/

[28] DECENTE D5.2. “Final release of the AI-integrated fog computing platform and final setups of pilots for demonstrations”.

39
# 7 Glossary & Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>CP</td>
<td>Control Plane</td>
</tr>
<tr>
<td>CRD</td>
<td>Custom Resource Definition</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>eBPF</td>
<td>Berkeley Packet Filter</td>
</tr>
<tr>
<td>EDA</td>
<td>Exploratory Data Analysis</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HOIC</td>
<td>High Orbit Ion Cannon</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDPS</td>
<td>Intrusion Detection and Prevention System</td>
</tr>
<tr>
<td>IMAP</td>
<td>Internet Message Access Protocol</td>
</tr>
<tr>
<td>IRC</td>
<td>Internet Relay Chat</td>
</tr>
<tr>
<td>k8s</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>L-ADS</td>
<td>Live Anomaly Detection System</td>
</tr>
<tr>
<td>LOIC</td>
<td>Low Orbit Ion Cannon</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>PPV</td>
<td>Positive Predictive Value</td>
</tr>
<tr>
<td>POP3</td>
<td>Post Office Protocol version 3</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SSL/TLS</td>
<td>Secure Sockets Layer/Transport Layer Security</td>
</tr>
<tr>
<td>TPR</td>
<td>True Positive Rate</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>XDP</td>
<td>eXpress Data Path</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>