# D5.1 Enhancing IoT Cybersecurity

| | | | |
|---|---|---|---|
| **WORKPACKAGE** | WP5 | **PROGRAMME IDENTIFIER** | H2020-ICT-2020-1 |
| **DOCUMENT** | D5.1 | **GRANT AGREEMENT ID** | 957246 |
| **REVISION** | V0.1 | **START DATE OF THE PROJECT** | 01/10/2020 |
| **DELIVERY DATE** | 30/10/2021 | **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), Bosch Sistemas de Frenado S.L.U. (BOSCH), 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** | 30/10/2021 |
| **ACTUAL DATE OF DELIVERY** | 15/11/2021 |
| **DOCUMENT TITLE** | Enhancing IoT Cybersecurity |
| **AUTHOR(S)** | A. Voulkidis (SYN), T. Velivassaki (SYN), S. Bourou (SYN), A. Zalonis (INTRA), D. Skias (INTRA), A. Gonos (OPT) |
| **REVIEWER(S)** | A. Corsi (ENG), Y. Kortesniemi (AALTO) |
| **ABSTRACT** | SEE EXECUTIVE SUMMARY |
| **HISTORY** | SEE DOCUMENT HISTORY |
| **KEYWORDS** | IoT, cybersecurity, Federated Learning, Generative Adversarial Networks, model poisoning, training, dataset |

# Document History

| Version | Date | Contributor(s) | Description |
|---|---|---|---|
| V0.1 | 09/06/2021 | SYN | Table of Contents |
| V0.2 | 21/07/2021 | OPT | Contributions related to IoT vulnerabilities and attacks |
| V0.3 | 14/09/2021 | INTRA | Input for IoT-NGIN cyberthreat modelling methodology |
| V0.4 | 27/09/2021 | SYN, INTRA, OPT | Contribution to state-of-the-art GAN models and tools |
| V0.5 | 12/10/2021 | SYN, INTRA, OPT | Contribution to IoT-NGIN dataset generator; Analysis of LLs use cases |
| V0.6 | 22/10/2021 | SYN, INTRA | Updates and further additions to IoT-NGIN dataset generator |
| V0.7 | 04/11/2021 | SYN, INTRA | Contribution to IoT vulnerabilities crawler |
| V0.8 | 08/11/2021 | SYN | Installation guidance updates for IoT-NGIN dataset generator; Overall updates |
| V0.9 | 09/11/2021 | SYN | Complete draft ready for Peer Review |
| V0.9.1 | 11/11/2021 | ENG | Peer reviewed version |
| V0.9.2 | 12/11/2021 | AALTO | Peer reviewed version |
| V1.0 | 15/11/2021 | SYN | Updates on the basis of review comments; Quality check; Final version |

# Table of Contents

# List of Figures

# List of Tables

# List of Acronyms and Abbreviations

| | |
|---|---|
| AGV | Automated Guided Vehicle |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| AR | Augmented Reality |
| CDF | Cumulative Distribution Function |
| CPU | Central Processing Unit |
| CSV | Comma-Separated Values |
| CTGAN | Conditional Tabular GAN |
| CVE | Common Vulnerability Exposure |
| CVSS | Common Vulnerability Scoring System |
| CWE | Common Weakness Enumeration |
| DCGAN | Deep Convolutional GAN |
| DDOS | Distributed Denial of Service |
| DOS | Denial-of-Service |
| EHR | Electronic Health Records |
| ENISA | European Union Agency for Cybersecurity |
| ERR | Error Rate based Rejection |
| EU | European Union |
| FedAvg | Federated Average |
| FL | Federated Learning |
| GAN | Generative Adversarial Network |
| GDPR | General Data Protection Regulation |
| GPU | Graphics Processing Unit |
| HTTP | Hypertext Transfer Protocol |
| IID | Independent and Identically Distributed |
| IDS | Intrusion Detection System |
| IoC | Indicators of Compromise |

| | |
|---|---|
| IoT | Internet of Things |
| JSON | JavaScript Object Notation |
| LFR | Loss Function based Rejection |
| MAD | Malicious Attack Detector |
| MISP | Malware Information Sharing Platform |
| ML | Machine Learning |
| MQTT | Message Queue Telemetry Transport |
| MTD | Moving Target Defence |
| NIDS | Network Intrusion Detection System |
| NIS Directive | Directive on Security of Network and Information Systems |
| OES | Operators of Essential Services |
| OWASP | Open Web Application Security Project |
| PMU | Phasor Measurement Unit |
| PNR | Passenger Name Records |
| PQA | Power Quality Analyser |
| R2L | Remote to Local |
| RCE | Remote Code Execution |
| SDG | Synthetic Data Gym |
| SDV | Synthetic Data Vault |
| SGD | Stochastic Gradient Descent |
| SSRF | Server-Side Request Forgery |
| U2R | User to Root |
| VGM | Variational Gaussian Mixture Model |
| VNC | Virtual Network Computer |
| UWB | Ultra-Wide Band |
| WGAN | Wasserstein Generative Adversarial Network |
| WGAN-GP | Generative Adversarial Network with Gradient Penalty |
| WP | Work Package |

# Executive Summary

The Internet of Things (IoT) and Artificial Intelligence (AI) combined together have revolutionized intelligence delivery in a multitude of applications. At the same time, the smooth operation of intelligent IoT systems requires consideration of cybersecurity aspects in the design and delivery of such IoT systems. IoT-NGIN provides a set of tools, which aim at protecting IoT systems participating in Federated Learning installations against cyberattacks, such as network or data and model poisoning attacks. The main achievements of IoT-NGIN towards the cybersecurity tools included in the present report can be summarized as follows:

- Analysis of vulnerabilities and cyberthreats which may challenge the secure operation of federated learning systems, involving IoT and edge devices
- Analysis of cybersecurity needs of the use cases of IoT-NGIN Living Labs, which involve federated learning in training their ML models
- Presentation of a generic cyberthreat modelling methodology proposed and respected by IoT-NGIN for providing on-device Federated Learning security
- Presentation of four cybersecurity tools which will be developed in IoT-NGIN aiming at enhancing cybersecurity in IoT-based Federated Learning systems, namely:
    - Generative Adversarial Network (GAN) based IoT attack dataset generator, useful for generating synthetic datasets of attacks, which can be further used for Machine Learning model training.
    - Malicious Attack Detector (MAD) able to identify anomalous behaviors in network or model update datasets and thus facilitate the detection of potential cyberthreats or attacks.
    - IoT vulnerabilities crawler which allows identifying network vulnerabilities in IoT systems and thus accelerate their mitigation.
    - Moving Target Defence (MTD) network of Honeypots as a useful tool for tracing and analysing attackers' behaviours, induced by IoT systems' vulnerabilities, and thus allowing enhancement of threat modelling and management processes.
- Extensive analysis of state-of-the-art Generative Adversarial Network (GAN) models and tools for synthetic dataset generation.
- Initial version of the Generative Adversarial Network (GAN) based IoT attack dataset generator, including technical design and specifications, as well as initial implementation
- Initial version of the IoT vulnerabilities crawler, including technical design and specifications.

The work presented in this document acts as the basis for the future activities of IoT-NGIN in cybersecurity for on-device federated learning. The future work includes further updates for the presented components, as well as design and development for the rest two. The outcomes of these activities will be reported in D5.2 "Enhancing IoT Cybersecurity (Update)", which is due in the third quarter of 2022.

# 1 Introduction

The Internet of Things (IoT) has launched for good in both business and everyday lives, with numerous distributed and highly diversified "things" sensing different aspects of their environment. Different combinations of devices, sensors and business scope across domains provide the bill of materials for numerous, often, unprecedented, applications, leaving room for both inspiration and innovation. The connected things are continuously increasing in volume, and capabilities, collecting huge amounts of data. IDC predicted in 2020 that *by 2025 there will be 55.7 B connected devices worldwide, 75% of which will be connected to an IoT platform* [1]. IDC also estimates in the same report data generated from connected IoT devices to be 73.1 ZB by 2025, growing from 18.3 ZB in 2019. Moreover, Gartner estimated in 2020 that *47% of organizations intend to increase investments in IoT despite the impact of COVID-19* [2]. The same survey reveals that IoT adoption is primarily driven by the Digital Twin and Artificial Intelligence (AI) technologies.

AI provides the intelligence to an IoT platform that enables translating raw information into useful forecasts and insights that allow triggering actions in business-specific defined workflows. Together, IoT and AI have revolutionized the perception of smartness in connected systems, providing insights to digital pioneers both in real time and in great detail.

Machine Learning (ML) is an AI technology, which allows to automatically identify patterns and detect anomalies in the data collected by IoT devices, such as temperature, pressure, humidity, air quality, vibration, sound, but also images, video and voice. Moreover, other AI technologies such as speech recognition and computer vision allow identifying linguistic or visual patterns, enabling inference decisions, only possible by humans until recently. AI applications for IoT enable companies to avoid unplanned downtime, increase operating efficiency, spawn new products and services, and enhance risk management [3].

At its basic level AI enables the prediction of undesired or risky events, while at a more advanced level is combined with actuation capabilities in IoT systems, which enable automated reaction to such events, without human intervention. Indeed, several ML techniques aim to improve the efficiency of the models in making predictions, such as Deep Learning, Reinforcement Learning, Transfer Learning, as well as Federated Learning, as discussed in D3.1. Federated Learning (FL) aims to build and train global models based on training datasets that are distributed across different remote devices while avoiding data leakage.

Despite the indisputable benefits of the combined use of IoT and AI, cybersecurity concerns may be raised from the extensive use of connected and highly automated systems. As stated in the State of the Union (SOTEU) in 2021 *"if everything is connected, everything can be hacked"* [4]. In SOTEU 2021, the need for a European Cyber Defence Policy, including legislation on common standards under a new European Cyber Resilience Act has been identified. Moreover, the SOTEU 2020 [5] had already identified the need for realizing the Digital Decade in Europe and IoT together with AI can be a driving force.

The legislative framework of the European Commission towards cybersecurity builds upon:

- The EU Cybersecurity strategy [6], which proposes building a European Cyber Shield via a network of Security Operations Centres across the EU, identifying the significance of AI and ML techniques in malicious activities detection in such centres. Also, it suggests providing ultra-secure communication infrastructure, integrating cutting edge technologies, such as Quantum, 5G, AI, edge computing.

- The Directive on Security of Network and Information Systems (the NIS Directive) (EU) 2016/1148 [7], which is at the core of the EU Single Market for cybersecurity. It states the need to take technical and organizational measures to "address risks" posed to systems of both Operators of Essential Services (OES) and providers of digital services. Both IoT and AI are at the core of such operations. A reformed NIS Directive has been proposed to be developed, in order to "*provide the basis for more specific rules that are also necessary for strategically important sectors, including energy, transport and health*" [6]. The draft NIS2 Directive [8] contains a catalogue of measures listing among other things, risk analysis and security concepts, prevention of security incidents and crisis management, which must at least be implemented by the companies.
- The Cybersecurity Act [9], promotes ICT certification at EU level, based on a European Cybersecurity Certification Framework, which will result in ICT products, services and processes in the EU to operate with an adequate cybersecurity level.

In addition, the General Data Protection Regulation (GDPR) also requires risk assessment procedures to be in place for those organizations that collect, process and store Personal Identifiable Information (PII). Article 34 to the GDPR states that "the controller has taken subsequent measures which ensure that the high risk to the rights and freedoms of data subjects is no longer likely to materialize" [10].

Considering these, IoT-NGIN provides a set of cybersecurity tools for IoT Federated Learning systems. The main project contributions are towards anomaly detection and threat monitoring, aiming to identify both existing and zero-day network-level attacks suffered by "classical" connected systems, as well as data and model poisoning attacks compromising FL systems. IoT-NGIN leverages ML and Generative Adversarial Networks to train and develop its *Malicious Attack Detection (MAD)* service. Moreover, IoT-NGIN designs and develops a distributed vulnerability scanning service for IoT device, as well as a distributed network of dynamically changing honeypots, allowing to lure and trace attackers' activity in controlled environments, without compromising the production environment's operation. Such threat monitoring allows companies assess and mitigate the cybersecurity risks which are possible in their IoT systems, helping them comply with the EU regulations for cybersecurity.

The present document, entitled "Enhancing IoT Cybersecurity", is the first deliverable of WP5 (D5.1) and reports the activities of Task 5.1 "*Mitigate poisoning attacks in on-device federated ML*" and Task 5.2 "*Adversarial access early attack detection*. Moreover, the activities reported in D5.1 align with the objectives of WP5:

- Develop the first to our knowledge systematic study on local model poisoning attacks to on-device federated ML
- 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.

The present document is a technical report which provides technical specifications and initial version of the Generative Adversarial Network (GAN) based IoT attack dataset generator and the IoT vulnerabilities crawler of IoT-NGIN.

## 1.1 Intended Audience

The target audience includes mainly IoT system providers and ML engineers, including technical staff. Through this report, they can find information about cyberthreats imposed in current and future IoT systems enhanced with AI functionality. Beyond their awareness on the underlying threats, they may identify their relevant cybersecurity requirements in application domains covered by the IoT-NGIN Living Labs. Moreover, the audience may identify through this report underlying business model in defining a threat modelling approach for their systems. The IoT-NGIN threat modelling methodology could potentially raise their awareness and motivate them in adopting a similar approach. Of course, the technical specifications and design of the GAN based dataset generator and the IoT vulnerabilities crawler are of high interest to the technical staff, based on their expertise. Furthermore, IT and ML developers have the chance to download, test and extend the initial versions of these components.

Moreover, the report can be useful to policy makers, who can find technical advantages of the IoT-NGIN threat modelling approach. The cybersecurity benefits and technical details could be promoted as indicative guidance of implementing cybersecurity measures in IoT-based FL systems.

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 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

First of all, the project activities which fed this document consider the use case requirements, as well as the IoT-NGIN meta-architecture and its instantiation, as derived in WP1. The content of this deliverable highly relates to the activities of WP3, regarding the development of the FL framework, which the cybersecurity modules presented in the current document refer to. Also, the content of the deliverable will feed the future activities within the framework of Task 5.1 and Task 5.2 and is planned to be updated in deliverable D5.2 "Enhancing IoT Cybersecurity (Update)", due in Q3 2022. Moreover, the document provides useful feedback to the future integration and validation activities, as well as to the preparation of the trials in the Living Labs (LLs) in WP7. Last, but not least, the outcomes presented in this document may be exploited for impact creation in WP8.

## 1.3 Document overview

The rest of the document is organized as follows.

Section 2 draws the background and motivation for providing cybersecurity tools for IoT-based FL systems.

Section 3 analyses the cybersecurity requirements of the use cases which will trialed in the LLs, identifying their needs for the IoT-NGIN cybersecurity tools for on-device FL.

Section 4 presents the IoT-NGIN cyberthreat modelling methodology and briefly presents the IoT-NGIN cybersecurity tools, mapping them to the steps of this methodology.

Section 5 presents state-of-art techniques GAN-based techniques for synthetic dataset generation and discusses their suitability for generating IDS datasets.

Section 6 provides the technical design of the GAN-based Dataset Generator of IoT-NGIN and provides installation and user guidance for its initial implemented version.

Section 7 provides the technical design of the IoT Vulnerability Crawler of IoT-NGIN, as well as information about the deployment of implemented releases, as soon as they become available.

Section 8 draws conclusions and closes the deliverable.

# 2 Background and Motivation

In this section, the need for adopting cybersecurity solutions while developing or running IoT systems is identified, highlighting potential vulnerabilities and attacks in smart IoT systems. IoT-NGIN aims to address such cyberthreats across domains, starting from ensuring cybersecurity of FL processes in IoT-NGIN LL use cases.

## 2.1 What is on-device Federated Learning

In recent years, the development and extensive use of smart devices has raised increasing interest in accessing, extracting, and sharing valuable information, which is gathered by different types of devices. Already now, massive amounts of data are being collected from different devices in a distributed manner [11] [12]. Such wealth of data is often used to feed Machine Learning (ML) functions and applications, which are able to generate knowledge in an intelligent manner. However, the effectiveness of Machine Learning (ML) models can be significantly improved when training is done synergistically, exploiting multiple information feeds, which, however, raises data privacy concerns.

Federated learning (FL) systems provide a noble path, as they are able to perform Machine Learning without the need to share the private data [13]: FL enables edge devices to collaborate in training a shared model without disclosing the training data to a central server as the training takes place on-device.

In FL settings, data remain within their administrative domain and training is performed locally. Then, the locally trained ML models are shared among the federated nodes or a central node, leading to the generation of an aggregated model. In this way, data sovereignty is respected via the FL setting, since the training data remain at the devices, while they are never processed at central servers.

Indicatively, Figure 1 illustrates a common FL setup, which consists of a central server $S$ and several dispersed nodes $n_i$, $i \in 1, 2, \dots N$ (the nodes could be e.g. smartphones, laptops, or IoT devices). In this setting, the nodes exchange locally trained model updates with the central server, which performs the aggregation of the individual models. Specifically, in each iteration, the server $S$ sends the global model to each node $n_i$, $i \in 1, 2, \dots N$. Then, the model is trained based on the private local data stored on each node, and the updated model parameters are sent back to the central server $S$, where all the model updates from each different node are aggregated, creating a new global model. This procedure is then repeated until it satisfies a termination condition, such as reaching a specified number of iterations.

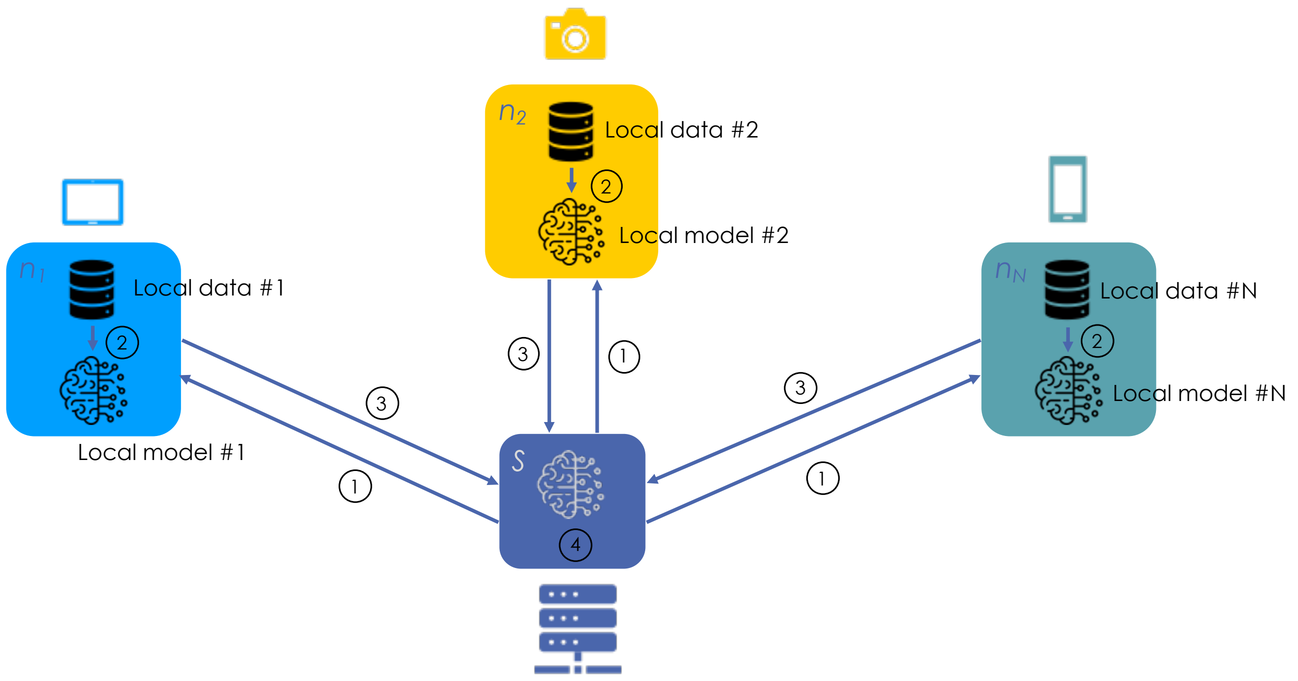


Figure 1: Federated Learning framework. 1. The global model is sent to the devices. 2. The model is updated based on the data. 3. The updated parameters are sent to the server. 4. A new global model is created.

The basic federated averaging algorithm was introduced as Federated Average (FedAvg) in 2017 [14], in which an averaging local Stochastic Gradient Descent (SGD) is used to update the model. The basic FL setup includes the learning of a global model using data stored locally at different remote devices and then only sending the updates of the trained model to a central server. Specifically in the case of FedAvg, the aim of the basic FL setup is to minimize the objective of the global model $w$, which is the sum of the weighted average of the devices' loss:

$$\min_{w} F(w), where\ F(w) := \sum_{k=1}^{n} p_k\, F_k(w) \qquad (1)$$

Where *n* is the total number of devices, $F_k(w)$ is the local objective function for the $k_{th}$ device, while $p_k \geq 0$, specifying the relative impact of each device and $\sum_k p_k = 1$.

Federated Learning has been largely researched in the literature with the aim of optimizing the distributed training process and costs. While optimization of (centralised) ML processes has been focused on achieving low latency and storage requirements, such as in lite versions of TensoFlow [15] and PyTorch [16], the distributed training under the FL setting shifts the computational overhead from the centralized server to the devices, which affects the goals of optimization. The research and development of FL algorithms relies largely on open source FL experimentation frameworks, the most common ones being TensorFlow Federated [17], PySyft [18], LEAF [19] and FedML [20]. In the following, the main challenges related to FL are discussed, accompanied by a brief analysis of state-of-the-art FL techniques in addressing them.

### *Challenges*

The minimization problem of FL models (1) imposes several challenges. The first one is related to **communication cost**, which is a crucial bottleneck in FL systems. Specifically, those systems mainly consist of a large number of devices, which makes the communication slower. Another challenge is about the systems **heterogeneity** since the devices on a FL network may have different storage, computational and communication capabilities due to the varying hardware components (CPU, GPU, memory), network connections (Wi-Fi, 3G, 4G, 5G) as well as power (battery life). It is very common on FL set ups that active devices can drop out due to energy or connectivity restrictions, limiting the FL model training to devices with better conditions. However, the FL system should be effectively scaled on all devices, independently of their type. Additionally, **statistical heterogeneity** poses another a challenge: FL systems commonly consist of several devices, which end up collecting data in a non-uniformly distributed way. This contravenes the frequently-held hypothesis of independent and identically distributed (IID) data, which increases the analysis, modelling, and evaluation difficulties. Lastly, **privacy concerns** are important even in FL applications. Even though FL provides more protection to private data of each device by only sharing the model updates, this communication between devices and server can still disclose sensitive information.

### *Addressing communication challenges*

Efficient communication on FL system can be achieved with various strategies. Hence, optimization methods that enhance flexible local updating as well as low client participation are widely used on FL networks. FedAvg works satisfactorily for non-convex problems; however this method does not always guarantee convergence, but it can still diverge in FL systems with heterogeneous data [21]. Additionally, works which try to succeed an effective communication in a FL setup have been proposed. Indicatively, in [22] quantization with structured random rotations was implemented, which restricts the communication between server and devices by applying lossy compression and dropout.

Regarding the system heterogeneity in FL system, several studies have investigated the impact of different resources to the performance of FL model, and different approaches have been developed to handle this issue. The authors in [23] examine the effect of heterogeneity in computational resources, such as CPU, memory, and network resources, on the training time of FL. Techniques such as asynchronous communication, active sampling of participating devices, and fault tolerance can address the system heterogeneity. A novel device selection strategy based on resources is presented by [24], in which the server aggregates as many updates as possible in a pre-defined time range. The work in [25] designs mechanisms that encourage devices with higher-quality data to participate in the learning process based on the system overheads realized on each device. Finally, aiming to create a robust FL system towards dropped devices, the authors in [26] design a secure aggregation protocol that is tolerant of arbitrary dropouts, provided that there are enough model updates from the non-dropped devices. Although the system heterogeneity effect in FL has been quite extensively researched, it still remains an open issue which needs to be consider while designing FL systems.

### *Addressing operation in constrained devices*

Often, the edge devices, in which the training is performed, are low-cost devices without the capacity to handle and train large amounts of data. In that case, a key issue is that only a limited amount of data can be used during the training phase on each edge device. One approach to address this issue is to use cooperative model updates. There, each edge node shares its trained results, while it improves the common model using the collected updates from the other devices. Those intermediate training results can be either exchanged via a server among the devices or they can be transferred directly from one edge device to another. The method in [27] describes an on-device federated learning algorithm, which updates the model collaboratively between the edge devices and it is applied on anomaly detection tasks.

Moreover, another major issue relates to the training process of the FL algorithms, which might be too heavy for some low-capable devices to bear their execution. Interestingly, several well-known frameworks like Tensorflow Federated [28] and LEAF [19] provide advancements in developing FL algorithms, but they do not offer tools for estimating the training's computational overhead; such overhead could provide hints for the suitability of the developed algorithms really for on-device training. As an alternative, FLOWER is proposed for evaluating on-device FL on various smartphones and embedded devices [29] and constitutes a nice solution for on-device FL.

### *Addressing statistical heterogeneity*

Apart from system heterogeneity, FL networks can also suffer from statistical heterogeneity as the training of FL models may be performed with non-identical distributed data across devices. FL methods have been proposed to overcome this challenge. MOCHA [30] is an optimization framework, which allows the personalized learning on devices. In this approach, separate but related models are trained at each device while a shared representation via multi-task learning is also supported. A different approach in [31] introduces a meta-learning method using information from each device separately. Additionally, the FedProx framework [21] can manage data heterogeneity in FL systems. Slightly modifying FedAvg, FedProx guarantees convergence also for FL systems trained with non-identical distributed data.

Another important factor to be considered during the handling of heterogeneous data on FL devices, apart from accuracy, is fairness. Specifically, using the basic objective function, such as (1), the models in some devices may be advantaged, while models of other devices may be disadvantaged, since the training may introduce biases towards devices with large amounts of data. Some studies have introduced FL approaches, which reduce the variance of model performance among the devices. In [32] a variable number of local updates based on local loss is performed by some heuristics. The Agnostic Federated Learning framework in [33] improves the common model for any target distribution composed of a mixture of the client distributions. As a conclusion, works, as the ones presented, have been aimed for addressing statistical heterogeneity, which is still open for the research community, as no single solution has been or is able to be adopted for every case.

***Addressing privacy issues***

To further enhance privacy in FL settings, different privacy preserving ML techniques have been proposed. Differential Privacy (DP) [34] [35] is a framework that randomizes part of the FL learning algorithm's behavior to make it impossible to reveal behavior patterns that correspond either to the model and the learned parameters, or to the training data. Another privacy preserving technique is Homomorphic Encryption (HE) [36], which secures the ML procedure by computing standard mathematical operations on encrypted data. Furthermore, Secure Multiparty Computation (SMC) [37] [38] is a cryptographic technique that empowers distributed parties to jointly compute an arbitrary function without revealing their own private input and output pairs. SMC techniques provide security verification in well-defined simulation frameworks, ensuring complete zero-proof knowledge [39].

## 2.2 How is IoT cybersecurity threatened

In addition to the inherent challenges of FL discussed above, smooth FL operation can be challenged by inherent constraints or vulnerabilities of the federated devices, which are induced by the distributed nature of the FL system. IoT devices operating unattended are subject to cybersecurity or physical attacks, which may render them malicious participants within the FL system. In the following, the cybersecurity challenges around FL are presented, discussing the vulnerabilities of IoT devices in subsection 2.2.1, which may lead to the attacks against the FL system, presented in subsection 2.2.2.

The European Union Agency for Cybersecurity (ENISA) defines security vulnerability as "*a weakness an adversary could take advantage of to compromise the confidentiality, availability, or integrity of a resource*" [40]. While there are known vulnerabilities, which may thus be appropriately addressed, zero-day vulnerabilities can be more damaging, as they have not been publicly disclosed and are kept private. Vulnerabilities provide breeding grounds for potential attackers, who deploy specially crafted code known as *exploit*, which takes advantage of vulnerabilities to compromise a resource.

### 2.2.1 IoT vulnerabilities

Security vulnerabilities in IoT devices could allow adversaries to knock devices offline, or even worse, to take control of them remotely as a steppingstone in order to gain wider access to the affected network.

From the point of view of the IoT web applications, Open Web Application Security Project (OWASP), which is a non-profit foundation, has compiled a list of the top 10 web application security risks for the 2021. This list[1] includes:

- Broken access control: Failures typically lead to unauthorized disclosure, modification, or destruction of data or performing a business function outside the user's limits.
- Cryptographic Failures: Failures related to cryptography (or lack thereof) often lead to exposure of sensitive data. Such failures include the use of Hard-coded passwords or broken cryptographic algorithms.

[1] https://owasp.org/www-project-top-ten/

- Injection: The attacker can provide malicious input (inject it) to a web application and change the operation of the application by forcing it to execute certain commands.
- Insecure Design: Focuses on risks related to design and architectural flaws, with a call for more use of threat modelling, secure design patterns, and reference architectures
- Security misconfigurations: Vulnerabilities might be introduced if unnecessary features are enabled or installed (ports, privileges etc,)
- Vulnerable and Outdated Components
- Identification and Authentication Failures: Failures include the permission of weak or well-known passwords, or ineffective multi-factor authentication etc.
- Software and Data Integrity Failures: Failures relate to code and infrastructure that does not protect against integrity violations such as when an application relies upon plugins, libraries, or modules from untrusted sources, repositories, or content delivery networks
- Security Logging and Monitoring Failures: Systematic logging is critical for detecting and responding to breaches.
- Server-Side Request Forgery (SSRF): SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL.

This list was compiled based on reported vulnerabilities, which were classified using the "Common Weakness Enumerations (CWE)". CWE refers to the types of software weaknesses, rather than specific instances of vulnerabilities within products or systems. This list acts as a "dictionary" of the top and most common software vulnerabilities identified.

"Malware Information Sharing Platform (MISP)" is another very well-known platform that facilitates the sharing, storing, and correlating of threat intelligence, including vulnerability information. It provides an adjustable taxonomy and an "Indicators of Compromise (IOCs)" database, i.e. a database of forensic data that identify potentially malicious activity on a system or network.

An additional source of vulnerabilities is the "Common Vulnerability and Exposures (CVE)" database hosted at MITRE. It is one of the largest publicly available source of vulnerability information. CVE provides a common enumeration that promotes shareability and consists of a list of information security vulnerabilities and exposures that aims to provide common names for publicly known problems. MITRE's CVE Program's mission is to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities[2]. A CVE also holds a Common Vulnerability Scoring System (CVSS), which is a vulnerability metric for communicating the characteristics and severity of software vulnerabilities. MITRE also publishes the CWE top 25 Most Dangerous Software Weaknesses on an annual basis[3]. For example, one of the recent and common CVEs for 2021 is the CVE-2021-3156: Sudo Privilege Escalation to Root. It holds a CVSS score of 7.8, which is considered high. This vulnerability is due to improper parsing of command line parameters and an attacker can exploit it via "sudoedit -s" and a command-line argument that ends with a single backslash character. To mitigate this vulnerability one can apply audit logs. Since brute-forcing is required to trigger the vulnerability, audit logs will be flooded with events relating to sudoedit.

[2] https://cve.mitre.org/cve/

[3] https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

An additional common CVE vulnerability is Citrix CVE-2019-19781. This particular CVE has a CVSS score of 9.8 which is considered critical. This CVE is a Remote Code Execution (RCE) vulnerability. An arbitrary code execution vulnerability is a security flaw in software or hardware allowing arbitrary code execution.

## 2.2.2 Attacks in Federated Learning systems

FL systems improve privacy preservation since the data remains locally on device, while only the model updates from each training are shared to the server. However, surveys have shown that FL may not always provide adequate privacy guarantees. Specifically, the sharing of model updates between the server and the IoT devices during the training phase can reveal sensitive information [41], [42]. In some cases, only a small portion of original gradients can already disclose important knowledge about the local data [43]. In addition, the malicious attackers want to exploit vulnerabilities to access the sensitive, private data of IoT devices or to control ML networks and manipulate the outputs. Therefore, to reinforce the FL features of IoT systems appropriately, all potential security, privacy, and network attacks on FL networks must be investigated.

### 2.2.2.1 Poisoning attacks

Attacks commonly occur during the training phase of FL networks, which are referred to as poisoning attacks, impacting either the dataset or the local model and, consequently, distorting the global ML model accuracy and/or performance. The general goal of poisoning attacks is to modify the behavior of FL model in an undesirable way. The attacks during training of FL networks are divided in two types, namely *Data Poisoning* and *Model Poisoning* attacks. The data poisoning attacks aim to compromise the integrity of training data, while the model poisoning attacks target partial or full model replacement during training. In an FL system, attacks can be executed by either the central server or the participating devices in FL system. Figure 2 shows that data poisoning is performed at local data collection, while model poisoning is sourced at the local model training process.

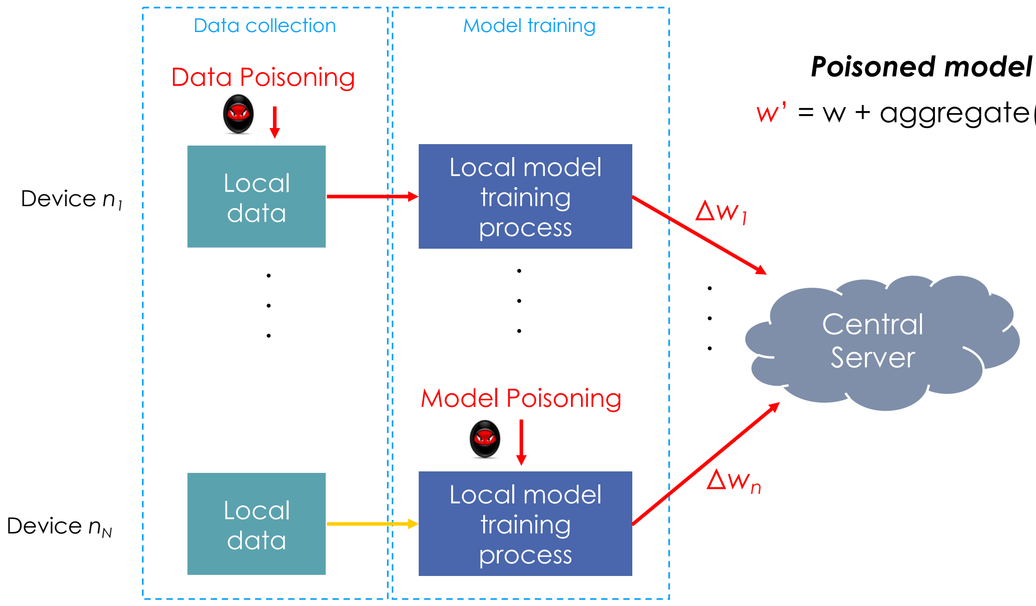

Data collection
Model training
Data Poisoning
Model Poisoning
Device $n_1$
Device $n_N$
Local data
Local data
Local model training process
Local model training process
$\Delta w_1$
$\Delta w_n$
Central Server
**Poisoned model parameter**
$w' = w + \text{aggregate}(\Delta w_1 + \ldots + \Delta w_n)$

Figure 2: Data vs. Model poisoning attacks on FL system.

#### 2.2.2.1.1 Data poisoning

Data poisoning attacks can be executed by any FL device, while the effect on the final trained ML model depends on the extent that the participating devices have been involved in attacks as well as the amount of poisoned data. Data poisoning attacks are usually less effective than model poisoning attacks. However, research in [44] demonstrates that poisoning training data at edge-case with low probability are more efficient.

Data poisoning attacks are further divided in two categories: clean-label and dirty-label. **Clean-label** [45] attacks are the attacks, in which the attacker cannot change the label of training data since there is a procedure certifying the correct class of the data, and thus, only the data is modified. The poisoning of data samples must be imperceptible in this kind of attack. On the other hand, the **dirty-label attacks** [46] can alter both the data and the label, which can be done by introducing several modified data with the desired target class, causing model misclassification. Dirty-labels attacks are also referred as backdoor attacks. An example of dirty-label poisoning is the label-flipping attack [47], where (as the name indicates) labels of honest training data belonging to one class are flipped to another class, without any modification of data features.

#### 2.2.2.1.2 Model poisoning

Model poisoning attacks aim to corrupt the updates of the local model before sending them to the central server or to insert hidden backdoors into the lobal model [48]. Research performed in [49] demonstrates that the model poisoning attacks are much more efficient than data poisoning attacks in FL systems. Furthermore, in FL system, the aggregator is not familiar with the modes of local updates, making it incompetent to detect anomalies and verify the correctness of local updates. The performance of model poisoning requires sophisticated techniques and high computational resources.

The aim of model poisoning attacks is to provoke the global ML model to miss-classify a set of data without being perceived, thus with high confidence. Threats of model poisoning attacks on FL, introduced by a single malicious agent as well as strategies to perform this kind of attack are analyzed in [50]. Additionally, [49] and [51] explore model poisoning attacks by sending a poisoned subset of updates to the server in every given iteration. The poisoned updates can be introduced to the global model by hidden backdoors. The work in [48] demonstrates that even a single-shot attack could be adequate to insert a backdoor to a model. The method developed in [52] uses Generative Adversarial Network (GAN) models to generate model poisoning attacks. Specifically, malicious participants pretend to be benign, while utilizing the GAN architecture to generate training data with incorrect label to damage the other clients of FL system. Current methods that defend systems from poisoning attacks are not entirely applicable in FL systems. To mitigate those attacks from FL, promising approach would be anomaly detection in the central server.

### 2.2.2.2 Inference attacks

Inference attacks are another type of threats that can occur on FL systems. In those attacks, adversaries do not attempt to violate the model itself. However, they aim either to lead the model to produce wrong results or gather details about the model characteristics, which can be achieved due to the communication channels in the FL settings. The adversaries could use the uploaded model parameters to partially disclose information about the training data of each FL client (or node). Specifically, the attacker can use the model updates to infer private information, like membership, class representatives, and the attributes of a training subset. These kinds of attacks can be grouped into two categories, the white-box attacks, in which the adversary has full access to the FL model, and the black-box attacks, in which the attacker can only obtain the predicted output.

**Membership inference attacks** [53] intends to reveal information by examining if specific data samples exist on the private training set of a single client or of any client. For this purpose, shadow models are built, creating a similar dataset to the original one. Attackers in FL systems can perform both active and passive membership inference attacks [54]. In the case of passive attacks, the adversary monitors the model updates and executes inference without altering the local training or the global aggregation procedure. On the other side, in the more powerful case of active attacks, the attackers can tamper with the training protocol, thus sharing malicious updates and forcing the FL to disclose more information about FL client's local data.

Additionally, other inference attacks that make FL systems vulnerable, are **property inference attacks** and **model inversion attacks**. Property inference or reconstruction attacks have as goal to define specific sensitive properties' values given that the attacker knows all the non-sensitive attributes that describe the classes of the FL model [55]. The attackers can conclude characteristic about participant's private training data thought the inference of sensitive attributes from the observations of local model gradients at different training rounds. In **model inversion attacks**, the training data is reconstructed using the FL model updates [56]. This kind of attacks raised a lot of concerns about privacy. In recent years, the research community has started dealing with the development of GAN models with regards to the inversion task to better extract appropriate knowledge about attacks [57], [58].

### 2.2.2.3 Network attacks

A FL system can be considered a network, which, therefore, is vulnerable to different network attacks. Networks are vulnerable to different kinds of attacks, by which adversaries aim to gain access to the system and carry out actions related to their malicious intentions. In this section the most well-known network attacks are analyzed.

**In a Byzantine attack** the malicious participants have complete control of their authenticated devices and perform arbitrarily behavior to disrupt the network. Generally, in a byzantine attack, the adversarial device cannot be distinguished from the benign ones. Other cybersecurity attacks include **sybil attacks**, where the attacker simulates multiple participants or several false identities with the aim of accessing the system and adding more powerful attacks on it. The severity of sybil attacks increases when the malicious actor manages to control more devices.

**Denial-of-Service (DoS)** [59] is a type of attack, in which the malicious actor makes the whole network or some devices unavailable for the users. This is achieved by flooding the computing or memory resources of the targeted machine until normal traffic cannot be processed; therefore, the user access to the machine is denied. DoS attack is one of the most frequently occurring cases of network intrusion. **Distributed denial of service (DDoS)** attack aims to disrupt the normal traffic of a network by overwhelming the network itself or the surrounding infrastructure with fraudulent internet traffic from multiple sources. **Probe** is another type of attack, in which the hacker scans a machine or a device and gathers network information to identify vulnerabilities or weak points that can be used to harm the system [60]. **Remote to Local (R2L)** is a type of attack, in which a remote user sends packets through the internet to a machine or a device, in which she/he does not have access [61]. The attacker aims to reveal the network vulnerabilities and take advantages of the privileges that a local user would have. The **User to Root (U2R)** type refers to an attack, in which the attacker imitates a normal user, while attempting to gain root access and super user privileges to the system [62].

Yet another type of attack are **backdoors**, with which the malicious users can avoid normal security requirements and authentication procedures without being discovered, aiming to access a target network system. Via these attacks, the attacker can reach the resources of the system, like servers or participants, and change the settings and properties. **Botnet** is a type of cyberattack, which is accomplished by a group of malware-injected devices controlled remotely by the attacker. These attacks are more sophisticated than others, due to their ability to scale up to cause greater damage to the system. Finally, **Man in the middle** attacks happen when attackers interrupt traffic within the network and secretly relay it - possibly altered- between the communicating parties, which believe that they are in a private communication. This attack can occur when malicious actors find a way to deceive the communication protocols, even if they are highly secured.

## 2.3 How can ML contribute to cybersecurity in on-device FL

The presence of attackers in an FL system degrades the training procedure, which introduces security risks on the learning outcomes. Therefore, the detection of malicious nodes as well as the poisoned model updates is of high importance in FL settings. However, it is common, that the attackers pretend to be benign nodes among the system, making the detection of them a challenging procedure. In recent years, the research community has expressed an intense interest in defense mechanisms to mitigate poisoning attacks, and thus, to reinforce the security of FL settings.

Some of the defense techniques exploit analytics and statistics to recognize abnormalities in events that do not follow an expected pattern. To achieve this, a profile of normal behavior is needed to detect attacks as a diversion from the normality. In FL systems, data and model poisoning attacks can be detected by different anomaly detection methods. For instance, Krum method [51] uses Euclidian Distance to identify malicious users based on the fact that their parameter updates remarkably vary from others. Additionally, the method in [63] detects the abnormal clients' updates at the server. The authors in [64] attempt to handle the adversarial attacks on FL by introducing a spectral anomaly detection with a variational autoencoders framework, which is used by the central server to identify and remove the malicious model updates based on their low-dimensional embeddings. Furthermore, [65] presents two defense mechanisms against local poisoning attacks, the loss function based rejection (LFR) and the error rate based rejection (ERR).

Recently, FoolsGold [66] was designed to detect sybil poisoning attacks by identifying malicious clients through calculation of the similarities between different updates. Various backdoor attacks in FL systems can be detected by the FederatedReverse method [67], which contains four parts, namely reverse engineering, global reverse trigger generation, outlier detection, and model repair. FederatedReverse also manages to restrict the influence of backdoor attacks via the model repair component.

A defence mechanism in FL settings, which mitigates poisoning attacks, but also provides *privacy preserving*, is a challenging issue and not yet well studied. MLGuard [68] is a privacy preserving distributed ML framework that simultaneously protects the learning procedure from the malicious nodes. The MLGuard allows the collaboration of two servers and multiple mobile users to train a common machine learning model, which provides privacy preservation as well as a defense mechanism against poisoning attacks. To accomplish this, the servers exploit a novel poisoning attack mitigation method to identify if node's updates are trustworthy or not. Specifically, the detection of malicious node is based on the similarity score of each node, so the servers reject those nodes with the lowest similarity score. Additionally, the strong privacy preserving character of MLGuard derives from an additive secret sharing mechanism, permitting the devices to secretly share their model updates to the servers.

Additionally, the research work in [69] presents a scheme for anomalous update detection in both IID and non-IID data, while the privacy of each FL client is protected without degrading the detection performance. In this method, the detection tasks are assigned to the clients, using the private data to evaluate the performance of the updates. Then, the aggregation of updates on the server is done by adjusting the weights based on the evaluation results. Finally, differential privacy is integrated to the detection tasks to protect clients' privacy.

# 3 Cybersecurity needs towards IoT-NGIN Living Labs

The Use Cases (UCs) which will be validated in the LLs have been analysed under the AI perspective in Deliverable document D3.1 "Enhancing deep learning / reinforcement learning" [70]. Leveraging this analysis, the following subsections provide cybersecurity-specific analysis of the UCs, for which ML model training via FL techniques has been specified.

## 3.1 Human-Centred Twin Smart Cities Living Lab

In the framework of the Human-Centred Twin Smart Cities Living Lab, the IoT-NGIN framework will be validated against three use cases, namely UC#1 "Traffic Flow Prediction & Parking Prediction", UC#2 "Crowd management" and UC#3 "Co-commuting solutions based on social networks". Based on D3.1, UC#1 will deploy Federated Learning for the training of the Traffic Flow Prediction and Parking Prediction AI services, while the rest will rely on other training techniques. The following subsections provide the background information for the UC needed for understanding the FL related cybersecurity needs of the UC, present the cybersecurity tools against on-device FL which will be validated, as well as the UC requirements from such tools.

### 3.1.1 Preconditions per application

The preconditions per UC application of the Human-Centred Twin Smart Cities Living Lab have been analyzed under Artificial Intelligence (AI) perspective in D3.1, section 3.1.1. The preconditions of UC#1 which are of interest to the cybersecurity tools against on-device FL attacks can be outlined as follows.

a) In order to model and **train distributed AI models on traffic flow and parking prediction**, weather data (and how that affects road & traffic conditions) and road data (number of cars, velocity, fluctuation) will be used. The predictive model will also consume historical data and public transportation information.
b) Traffic and parking prediction **ML models will be federated** at the edge cloud.
c) As IoT devices are installed in accessible public places and have network connectivity, their **safe network configuration** and operation must be ensured.

### 3.1.2 Mapping to IoT-NGIN FL-related cybersecurity tools

| GAN based dataset generation | Malicious Attack Detector (MAD) | IoT vulnerability crawler | MTD network of honeypots |
|---|---|---|---|
| - | - | Detection of vulnerabilities in IoT devices (cameras, radars) and edge/cloud nodes | - |

### 3.1.3 Use Case Requirements analysis

Table 1: Requirements analysis for Use case #1 "Traffic Flow Prediction & Parking Prediction".

| **Living Lab** | | **Human-Centred Twin Smart Cities Living Lab** | |
|---|---|---|---|
| **Use Case** | | **Traffic Flow Prediction & Parking Prediction** | |
| **FL service** | | **Traffic Flow Prediction** | **Parking Prediction** |
| **GAN based dataset generation** | **Dataset type** | N/A | |
| | **Attacks covered** | N/A | |
| | **Data size** | N/A | |
| **Malicious Attack Detector (MAD)** | **Data types** | N/A | |
| | **Data size** | N/A | |
| | **Data communication protocols** | N/A | |
| | **Data availability** | N/A | |
| **IoT vulnerability crawler** | **IoT devices** | Multispectral & Visual Cameras<br>Radars<br>Weather stations | |

| | | |
|---|---|---|
| | | Sensors on Robo-buses & City Streets |
| | **Deployment type** | Edge/cloud |
| **MTD network of honeypots** | **Service types mimicked** | N/A |
| | **Applications mimicked** | N/A |

## 3.2 Smart Agriculture IoT Living Lab

In the Smart Agriculture IoT Living Lab, the IoT-NGIN solution will be validated against UC#4 "Crop diseases prediction, Smart irrigation and precision aerial spraying" and UC#5 "Sensor aided crop harvesting". According to the AI-driven analysis of them, UC#4 will train its Crop diseases prediction AI service in a FL system. In this subsection, an analysis of UC#4 under the Smart Agriculture Living Lab is provided under the perspective of the cybersecurity tools against on-device FL; including the preconditions, then mapping of those preconditions to the cybersecurity tools and identifying relevant use case requirements.

### 3.2.1 Preconditions per application

The preconditions identified as relevant for cybersecurity against on-device FL for the "Crop diseases prediction, Smart irrigation and precision aerial spraying" use case are identified as follows:

a) Crop diseases prediction is based on images and real-time video analysis of the crop and the leaves captured from visual and multi-spectral cameras located on semi-autonomous drones flying over the orchard.
b) Crop diseases' prediction also considers measurements acquired via SYN SynField[4] precision agriculture IoT nodes, integrating a variety of sensor modules.
c) Real-time video analysis takes place either locally (on the drone), based on already trained ML models, or remotely (at the edge) based on **federated ML**.
d) Both SynField devices and drones will communicate with an edge server. So, the **network vulnerabilities** at device, edge or cloud level must be eliminated.
e) The training process of the ML prediction models must be protected against **cyberattacks**, which could highly impact the according predictions.
f) IoT-NGIN will leverage existing datasets available for Living Lab experimentation, to the extent possible. Alternatively, publicly available or **synthetic datasets** will be used.

[4] https://www.synfield.gr/about/.

## 3.2.2 Mapping to IoT-NGIN FL-related cybersecurity tools

| GAN based dataset generation | Malicious Attack Detector (MAD) | IoT vulnerability crawler | MTD network of honeypots |
|---|---|---|---|
| Generation of datasets including network-level attacks | Detection of network level attacks, as well as data/model poisoning attacks | Detection of vulnerabilities in IoT devices (SynField, drones) and edge/cloud nodes | Mimicking of vulnerable SynField nodes |

## 3.2.3 Use Case Requirements analysis

In this section, the requirements of UC#4 are drawn in Table 2 for each of the FL-related cybersecurity components of IoT-NGIN.

Table 2: Requirements analysis for Use case #4 "Crop diseases prediction, Smart irrigation and precision aerial spraying".

| **Living Lab** | | **Smart Agriculture IoT Living Lab** |
|---|---|---|
| **Use Case** | | **Crop diseases prediction. Smart irrigation and precision aerial spraying** |
| **FL service** | | **Crop diseases prediction** |
| **GAN based dataset generation** | **Dataset type** | CSV |
| | **Attacks covered** | Network-level attacks, Data & Model poisoning attacks |
| | **Data size** | To be defined |
| **Malicious Attack Detector (MAD)** | **Data types** | JSON |
| | **Data size** | To be defined |
| | **Data communication protocols** | HTTP<br>MQTT |
| | **Data availability** | Request only |
| | **IoT devices** | SynField devices, drones |

| | | |
|---|---|---|
| **IoT vulnerability crawler** | **Deployment type** | Edge/cloud |
| **MTD network of honeypots** | **Service types mimicked** | (indicative) Web server, Virtual Network Computer (VNC) |
| | **Applications mimicked** | (indicative) Smart Agriculture app dashboard, decoy database, Smart Agriculture app API |

## 3.3 Industry 4.0 Living Lab

The Industry 4.0 Living Lab will validate the IoT-NGIN framework in real-life applications via 3 use cases in both BOSCH facilities in Barcelona and ABB facilities in Pitäjänmäki, Helsinki, namely UC#6 "Human-centred safety in a self-aware indoor factory environment", UC#7 "Human-centred Augmented Reality assisted build-to-order assembly" and UC#8 "Digital powertrain and condition monitoring". All three use cases will use Federated Learning framework in training their AI models, according to D3.1.

### 3.3.1 Preconditions per application

The preconditions identified as relevant for cybersecurity against on-device FL for the "Human-centred safety in a self-aware indoor factory environment" use case are identified as follows:

a) **Edge computing** resources will be used to support a set of AI functions that will process the real-time location of the Automated Guided Vehicles (AGVs), based on the real-time stream coming from the safety cameras. The AI functions will determine a potential collision between AGVs, or between a worker and an AGV and will issue an early warning.
b) The AI models that will be used will be trained at the edge devices, in a **federated learning** set up among edge devices in the same factory.
c) The cameras installed in the factory for collecting images of the AGV are installed in accessible positions and communicate with the edge server via LAN or Internet connection, so any **network vulnerabilities** must be early identified.

The "Human-centred Augmented Reality assisted build-to-order assembly" Use Case underlines the following points, which are of interest to IoT-NGIN FL related cybersecurity tools.

a) This UC aims to assist human workers in the assembly line with the use of Augmented Reality (AR). Machine learning and computer vision techniques will be used to detect product defects and differentiate between different components and modules.
b) IoT-NGIN will be able to recognize the components and the stage of the assembly process using ML models, which will be trained locally at edge servers and **federated** to produce an aggregate model.

d) The cameras installed in the factory for collecting images during the assembly process are installed in accessible positions and communicate with the edge server via LAN or Internet connection, so any **network vulnerabilities** must be early identified.

Moreover, for UC#8 "Digital powertrain and condition monitoring", the following preconditions are of interest to FL-related IoT-NGIN tools.

a) Condition monitoring and predictive maintenance of powertrains and drive units will be based on ML models, which will be generated using **federated machine learning**.
b) Parameter tuning and optimization (e.g. in terms of energy consumption) of drive units will rely on ML models which will be generated via **federated learning**.
c) The IoT devices which will act as data sources, such as smart sensors and heat cameras will be installed in accessible positions and will communicate with edge devices to provide them their collected data. As such, **network vulnerabilities** should be early detected.

### 3.3.2 Mapping to IoT-NGIN FL-related cybersecurity tools

| GAN based dataset generation | Malicious Attack Detector (MAD) | IoT vulnerability crawler | MTD network of honeypots |
|---|---|---|---|
| - | - | Detection of vulnerabilities in IoT devices and edge/cloud nodes | - |

### 3.3.3 Use Case Requirements analysis

In this section, the requirements per use case are presented for each of the FL-related cybersecurity component of IoT-NGIN in Table 3 to Table 5, respectively.

Table 3: Requirements analysis for Use case #7 "Human-centred safety in a self-aware indoor factory environment".

| **Living Lab** | | **Industry 4.0 Living Lab** | |
|---|---|---|---|
| **Use Case** | | **Human-centred safety in a self-aware indoor factory environment** | |
| **FL service** | | **Collision Detection** | **AGV Route Planning** |
| | **Dataset type** | N/A | |

| | | |
|---|---|---|
| **GAN based dataset generation** | **Attacks covered** | N/A |
| | **Data size** | N/A |
| **Malicious Attack Detector (MAD)** | **Data types** | N/A |
| | **Data size** | N/A |
| | **Data communication protocols** | N/A |
| | **Data availability** | N/A |
| **IoT vulnerability crawler** | **IoT devices** | Cameras<br>Ultra-Wide Band (UWB) sensors<br>AGVs |
| | **Deployment type** | Edge/cloud |
| **MTD network of honeypots** | **Service types mimicked** | N/A |
| | **Applications mimicked** | N/A |

Table 4: Requirements analysis for Use case #8 "Human-centred Augmented Reality assisted build-to-order assembly "

| **Living Lab** | | **Industry 4.0 Living Lab** |
|---|---|---|
| **Use Case** | | **Human-centred Augmented Reality assisted build-to-order assembly** |
| **FL service** | | **(AR) Object detection and classification** |
| **GAN based dataset generation** | **Dataset type** | N/A |
| | **Attacks covered** | N/A |
| | **Data size** | N/A |
| **Malicious Attack Detector (MAD)** | **Data types** | N/A |
| | **Data size** | N/A |

| | | |
|---|---|---|
| | **Data communication protocols** | N/A |
| | **Data availability** | N/A |
| **IoT vulnerability crawler** | **IoT devices** | Cameras, Ultra-Wide Band (UWB) sensors, AGVs |
| | **Deployment type** | Edge/cloud |
| **MTD network of honeypots** | **Service types mimicked** | N/A |
| | **Applications mimicked** | N/A |

Table 5: Requirements analysis for Use case #9 "Digital powertrain and condition monitoring"

| **Living Lab** | | **Industry 4.0 Living Lab** | |
|---|---|---|---|
| **Use Case** | | **Digital powertrain and condition monitoring** | |
| **FL service** | | **Predictive maintenance of powertrains** | **Energy consumption optimization** |
| **GAN based dataset generation** | **Dataset type** | N/A | |
| | **Attacks covered** | N/A | |
| | **Data size** | N/A | |
| **Malicious Attack Detector (MAD)** | **Data types** | N/A | |
| | **Data size** | N/A | |
| | **Data communication protocols** | N/A | |
| | **Data availability** | N/A | |
| **IoT vulnerability crawler** | **IoT devices** | Variable speed drive, smart sensor, heat camera | |
| | **Deployment type** | Edge/cloud | |
| **MTD network of honeypots** | **Service types mimicked** | N/A | |
| | **Applications mimicked** | N/A | |

## 3.4 Energy Grid Active Monitoring/Control Living Lab

The Energy Grid Active Monitoring/Control Living Lab will validate the IoT-NGIN framework in two use cases, namely UC#9 "Move from Reacting to Acting in Smart Grid Monitoring & Control" and UC#10 "Driver-friendly dispatchable EV charging". UC#9 will exploit FL in training consumption and/or generation prediction AI services, while UC#10 will use FL for developing the "Forecasting of energy demand" AI service.

### 3.4.1 Preconditions per application

Under the IoT cybersecurity perspective, the following preconditions are taken into account for UC#9:

a) **Anomaly detection ML models** will be trained using **FL techniques** to identify grid health issues early, to the extent possible, allowing tracking the health of the grid and indicating that maintenance is required before obvious performance degradation or even failure.
b) The **cybersecurity of the model training** process is mandatory for the Critical Infrastructure of Energy, securing reliability and effectiveness of the ML models.
c) Given the integration of various platforms and numerous diverse devices, most of them exposed in public places (e.g. smart meters), **eliminating device vulnerabilities** is crucial for the UC.
d) **Network-level cybersecurity** is also of interest to the use case, to early detect and/or avoid hacking activity in the energy network, to the extent possible.

The preconditions identified as relevant for cybersecurity against on-device FL for the "Driver-friendly dispatchable EV charging" use case are identified as follows:

a) Data from geographically dispersed charging stations, electric vehicles and smart meters will be used to train ML models, able to forecast the energy demand for the EV charges.
b) The ML models will be trained via federated learning tools to exploit **knowledge from various edge servers**, which could belong to different vendors, so they would be unwilling to disclose any data.
c) As the charging stations, the electric vehicles and smart meters communicate with an edge or cloud server**, IoT vulnerabilities** should be early detected.
d) The **cybersecurity of ML model training** must be ensured.
e) **Network-level cybersecurity** will thus be enforced via anomaly detection ML models, providing inference over synthetic network datasets.

### 3.4.2 Mapping to IoT-NGIN FL-related cybersecurity tools

UC#9 and UC#10 will exploit the IoT-NGIN tools towards addressing FL attacks as shown in the next table.

| GAN based dataset generation | Malicious Attack Detector (MAD) | IoT vulnerability crawler | MTD network of honeypots |
|---|---|---|---|
| Generation of datasets including network-level attacks | Detection of network level attacks in the smart grid and across the network of charging stations; Detection of data/model poisoning attacks in | Detection of vulnerabilities in IoT devices (e.g. smart meters, Power quality analysers (PQA), Phasor Measurement Units (PMUs), charging stations, electric | N/A |

| | | | |
|---|---|---|---|
| | anomaly detection ML models for smart grid status monitoring | vehicles) and edge/cloud nodes | |

## 3.4.3 Use Case Requirements analysis

The UC#9 requirements for each of the IoT cybersecurity components of IoT-NGIN are drawn in Table 6.

Table 6: Requirements analysis for Use Case #9 "Move from Reacting to Acting in Smart Grid Monitoring & Control ".

| **Living Lab** | | **Energy Grid Active Monitoring/Control Living Lab** |
|---|---|---|
| **Use Case** | | **Move from Reacting to Acting in Smart Grid Monitoring & Control** |
| **FL service** | | **Anomaly detection in smart grid operation** |
| **GAN based dataset generation** | **Dataset type** | CSV |
| | **Attacks covered** | Network-level attacks |
| | **Data size** | To be defined |
| **Malicious Attack Detector (MAD)** | **Data types** | JSON |
| | **Data size** | To be defined |
| | **Data communication protocols** | MQTT |
| | **Data availability** | Real-time |
| **IoT vulnerability crawler** | **IoT devices** | Power quality analysers |
| | **Deployment type** | Edge/cloud |
| **MTD network of honeypots** | **Service types mimicked** | N/A |
| | **Applications mimicked** | N/A |

The UC#10-specific requirements are drawn in Table 7 for each of the FL-related cybersecurity components of IoT-NGIN.

Table 7: Requirements analysis for Use Case #10 "Driver-friendly dispatchable EV charging".

| **Living Lab** | | **Energy Grid Active Monitoring/Control Living Lab** |
|---|---|---|
| **Use Case** | | **Driver-friendly dispatchable EV charging** |
| **FL service** | | **Forecasting of energy demand** |
| **GAN based dataset generation** | **Dataset type** | CSV |
| | **Attacks covered** | Network-level attacks |
| | **Data size** | To be defined |
| **Malicious Attack Detector (MAD)** | **Data types** | JSON |
| | **Data size** | To be defined |
| | **Data communication protocols** | HTTP<br>MQTT |
| | **Data availability** | Real-time |
| **IoT vulnerability crawler** | **IoT devices** | Charging stations, Electric vehicle OBD devices, Near real-time smart meters |
| | **Deployment type** | Edge/cloud |
| **MTD network of honeypots** | **Service types mimicked** | N/A |
| | **Applications mimicked** | N/A |

# 4 IoT-NGIN methodology to IoT-driven cybersecurity in FL

As presented in section 2, IoT systems may suffer from various cybersecurity attacks. In addition to device or network vulnerabilities, risks may be posed by advancing technologies, powered by the increasing use of AI, which expand the attack surface of IoT systems and feed even more sophisticated attackers. IoT-NGIN proposes a cyber-threat modelling approach, which mostly aligns to ENISA's threat modelling methodology for AI [71]:

1. Identification of security objectives: The security properties of the IoT system are identified. These could refer to protecting data injection, model training or inference processes, to ensuring network or communication security, etc.
2. Identification of the system functions: The IoT system and its components' processes should be identified, highlighting the interactions between them, as well as with third-party systems, in order to ensure proper IoT system function.
3. Asset identification: Assessment of hardware and software components, the security of which should be protected. These could include IoT devices, data, processes, ML models and artifacts, as well as tools necessary for system operation.
4. Threat identification: The cybersecurity threats are identified as states or setups which result in the system assets failing to meet the security objectives.
5. Vulnerability identification: Extensively monitor the IoT system properties, which may enable the realization of the identified threats, based on state-of-the-art cyberattacks.
6. Threat detection: Cybersecurity threats are detected, based on evidence by already realized attacks, as well as on more advanced techniques, which may identify new threats which may be attributed to zero-day vulnerabilities.
7. Threat monitoring and continuous feedback: Cybersecurity attack patterns are monitored and analyzed, in order to provide feedback to the previous threat modelling steps. This may include allowing attackers' access in controlled environments for tracing attackers' footprint.

The implementation of the cyberthreat modelling methodology in IoT-NGIN is depicted in Figure 3. First, the security objective has been identified as securing the operation of on-device FL systems. Then, the use cases per LL are analyzed from the cybersecurity perspective, in order to identify both the use cases lifecycle, including their processes and interactions. In addition, the IoT devices and services participating in each ML/FL process are identified as assets of each use case. Threat identification is based on state-of-the-art analysis of relevant attacks, including open repositories, such as MITRE ATT&CK[5]. Within IoT-NGIN scope, data and model poisoning attacks, as well as network-level attacks are identified as necessary to be tackled. Common network-level vulnerabilities are identified in literature and vulnerability scanning is performed to detect any such vulnerabilities in IoT-NGIN FL settings. IoT-NGIN proceeds with ML-based anomaly detection to identify potential threats that have been materialized in the FL systems. Last, but not least, IoT-NGIN applies threat monitoring, by deploying honeypots mimicking vulnerable services or applications in controlled environments. This allows better understanding of attacks, analyzing their network traffic, and

[5] https://attack.mitre.org

thus, providing valuable feedback to the threat and vulnerability identification processes, as well as towards improving the threat detection process.

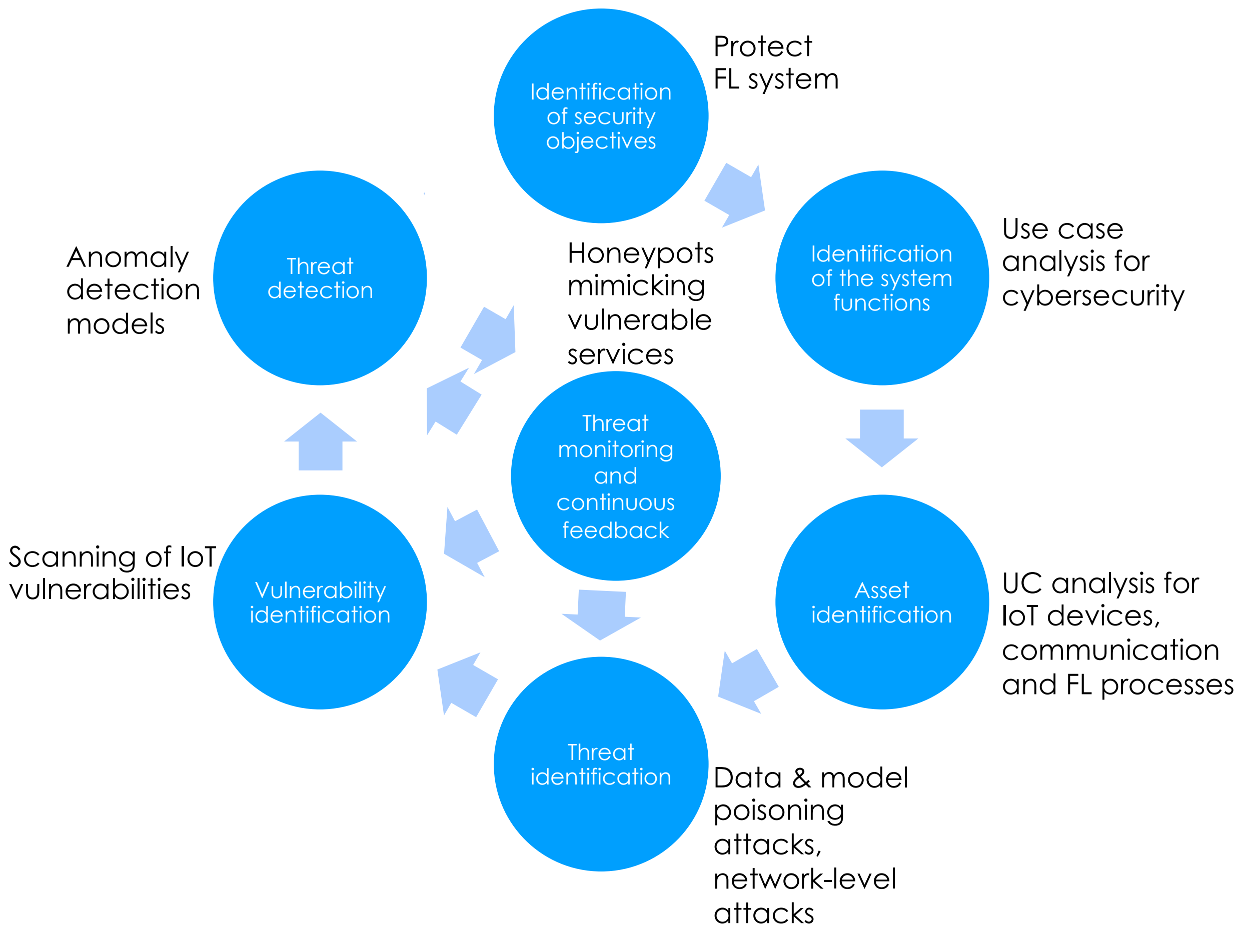

Identification of security objectives
Protect FL system
Identification of the system functions
Use case analysis for cybersecurity
Asset identification
UC analysis for IoT devices, communication and FL processes
Threat identification
Data & model poisoning attacks, network-level attacks
Vulnerability identification
Scanning of IoT vulnerabilities
Threat detection
Anomaly detection models
Threat monitoring and continuous feedback
Honeypots mimicking vulnerable services

Figure 3: Cyber-threat modelling approach; for each node, implementation hints in IoT-NGIN are provided.

In addition to the desk research and analysis required for accomplishing the identification steps, IoT-NGIN provides a set of tools which support the threat detection and monitoring processes for on-device FL settings.

First, IoT-NGIN provides the **Generative Adversarial Network (GAN) based IoT attack dataset generator** component, which generates high-value synthetic datasets of attacks, using a small portion of real data and preserving the utility and fidelity of real datasets. The GAN architecture consists of two components, namely the Generator and the Discriminator. The generator tries to generate new realistic data, as close to the input dataset as possible. Then, the discriminator tries to tell generated from real input data. As both components try to improve themselves, while the GAN is running, at the end the generator will be able to generate realistic datasets, while the discriminator will no longer be able to identify the synthetic against the real data. In IoT-NGIN, GANs are exploited in generating datasets implying data and model poisoning attacks, as well as network level attacks for IoT systems participating in FL configurations. Such synthetic datasets can be used to compensate for the unavailability of appropriate training datasets for ML anomaly detection models.

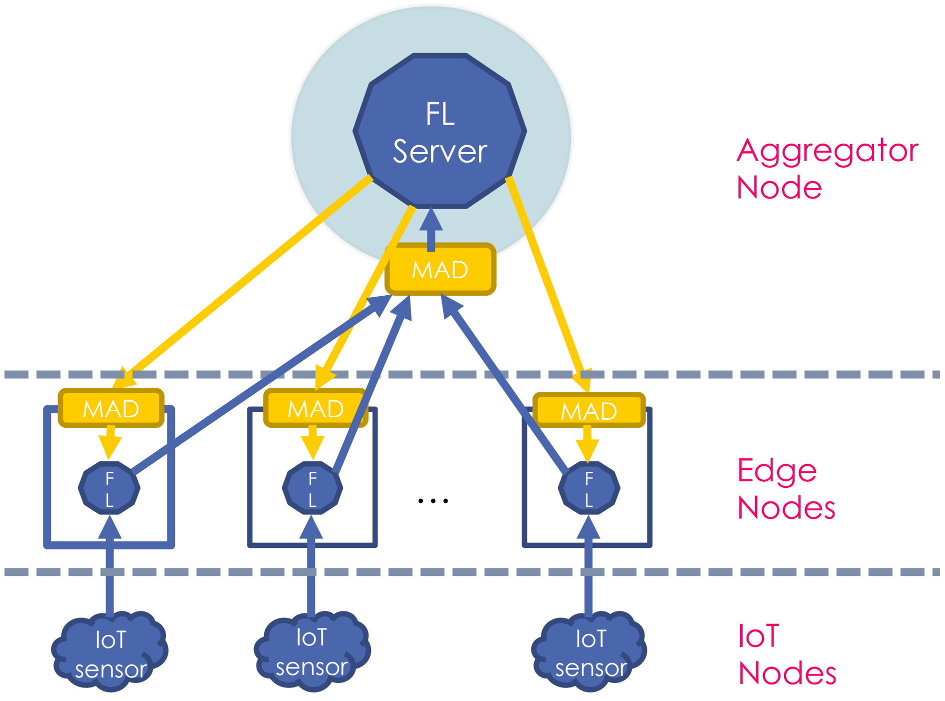


Figure 4: Malicious Attack Detector deployment in FL system.

Moreover, IoT-NGIN cybersecurity tools include the **Malicious Attack Detector (MAD)** component. It refers to ML-based anomaly detection module, able to identify attacks in on-device FL. The types of attacks or anomalies identified depend on the training dataset, which will rely on both real and synthetic data. Considering the FL system of Figure 4, a MAD instance can be 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.

The **IoT vulnerabilities crawler** is also part of the cybersecurity tools of IoT-NGIN for the distributed IoT systems, which may implement FL configurations. It is a distributed service which scans both IoT devices and services against a group of vulnerabilities organized in service oriented plugins. The crawler is useful to performing vulnerability assessment of IoT systems, as well as to providing useful feedback for improving their security, as well as to further analyzing cyber-threats or attacks related to the identified vulnerabilities in the ioT systems under investigation.

Last, but not least, the **Moving Target Defence (MTD) network of Honeypots** of IoT-NGIN allows exploration of attackers' behavior, exploiting IoT systems' vulnerabilities. Honeypots are widely used in network security. A honeypot is a decoy computer system that appears attractive to an attacker and can be used to collect information on threat behavior and vectors. MTD dynamically changes the attack surface to continuously increase complexity and confuse the attacker, thus preventing the system vulnerabilities from being exploited. The MTD network of Honeypots of IoT-NGIN can be used to mimick vulnerabilities identified by the IoT vulnerabilities crawler. Then, it can provide useful feedback to the vulnerability and threat modelling, as well as threat detection processes.

As a result, IoT-NGIN provides valuable tools for preserving cybersecurity in IoT systems. In the next sections, the initial versions of the GAN based IoT attack dataset generator and the IoT vulnerabilities crawler are presented.

# 5 State-of-the-art GAN-based approaches for synthetic dataset generation

Throughout this section, the GAN-based approaches for the generation of synthetic Intrusion Detection System (IDS) Dataset, developed over the IoT-NGIN project is going to be described. Firstly, an initial overview of GAN models is given, following by the analysis of the state-of-the-art GAN-based models for tabular data generation, which are examined over the project. Afterwards, the evaluation metrics that are applied on synthetic tabular data are detailed described.

## 5.1 GAN models overview

Generative Adversarial Networks (GANs) are a type of Neural Network architecture for generative modeling. GANs were first introduced by Goodfellow et al. [72] in 2014. GAN models are used for unsupervised learning, based on a two-player game theoretical scenario to learn the distribution and the patterns of the training data, in such a way that the model can generate new data that preserve the characteristics of the training data. Later developments in GANs improved speed and training performance. Arjovsky et al. introduced the Wasserstein model [73], which is an improved GAN model that leverages the Wasserstein-1 metric to define a more sophisticated loss function with Gulrajani et al. introducing gradient penalty on Wasserstein GAN [74] to address the side effects of weight clipping during training. In various applications, mainly focusing on generating images [75], [76], [77], [78] GANs have shown remarkable results. Regarding the generation of adversarial malicious examples, GANs have been tried out in some methods. In [79] and [80], the authors generate malicious traffic records using GANs, while those type of models are used in [81] to synthesize malware examples.

### 5.1.1 Vanilla GAN

A Vanilla GAN model consists of two independent sub-models, the Generator G and its adversary, the Discriminator D. The generative model G understands the data distribution p(g) of the real data space x. Then, considering an input noise variable, the Generator G generates new adversarial examples G(z) that have the same distribution of x. The Generator G is trained to maximize the probability that the Discriminator D could correctly predict generated as real samples, while the Discriminator D is trained to distinguish if the given sample is real or generated by the Generator G.

The mathematical expression of the Vanilla GAN derives from the cross-entropy between the real and generated distributions, and is the following:

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

In equation (1), discriminator D(x) tries to maximize the quantity $V(G,D)$ for any given generator G(z), while G(z) is the generator's output when given z. $\mathbb{E}_{x\sim p_{data}(x)}$ and $\mathbb{E}_{z\sim p_z(z)}$ correspond to expected values over all real data instances and over all generated fake instances, respectively. The global optimum for this task is when $p_{data} = p_g$ and this corresponds to the global minimum of the training criterion. To avoid overfitting when training finite datasets, the Discriminator D must be optimized simultaneously with the Generator G. Practically, it is possible that the equation (1) could not provide adequate gradients for G to be trained sufficient. During the start of the learning process, the Discriminator D may reject high confidence samples created by a poor Generator G, because they are different from the training data. This can lead to saturation of $log(1 - D(G(z))$. To address that, Generator G can be trained to maximize $log\, D(G(z))$ instead of minimizing $log(1 - D(G(z))$.

## 5.1.2 Wasserstein GAN (WGAN)

A modification of Vanilla GAN is the Wasserstein GAN (WGAN) [73], that aims to train the Generator and the Discriminator to better approximate the distribution of the real data and improve the training process. Firstly, a meaningful loss function, namely Wasserstein-1 distance, is applied, which is correlated with the quality of the generated samples. The Wasserstein-1 distance measures the distance between probability distributions. Additionally, the Discriminator of WGAN does not contain a sigmoid activation at the last layer, resulting to logits. The Wasserstein-1 distance, as it is described in equation (2), is applied directly to logits, forcing the logit distributions to be similar. The output of the discriminator loss is a score, which indicates the realness or fakeness of the generated sample. The Lipschitz function is used to constrain the optimization problem, by clipping the weights of the discriminator function. Lastly, the RMSProp optimizer is used. The Wasserstein-1 metric, also called Earth Mover's distance is defined as follows:

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||], \tag{2}$$

where $||x - y||$ is the cost function, $p_r$ , $p_g$ are the probability distributions and $\Pi(p_r, p_g)$ denotes the set of all joint distributions $\gamma(x, y)$. The infinite number of joint distributions in $\Pi(p_r, p_g)$ makes the Wasserstein-1 distance intractable. Thus, the authors in [73] apply Kantorovich-Rubinstein duality. Hence, Wasserstein-1 distance takes the following form:

$$W(p_r, p_g) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_\theta}[f(x)], \tag{3}$$

where the supremum corresponds to all the 1-Lipschitz functions $f: X \to \mathbb{R}$. Merging this function with a GAN, the result is as follows:

$$\min_G \max_{w \in W} \mathbb{E}_{x\sim P_{data}}[f(x; w)] - \mathbb{E}_{z\sim p_z}[f(\mathrm{G}(z; \theta_g); w)], \tag{4}$$

where the functions $\{f_w\}_{w\in W}$ are *K-Lipschitz*.

The WGAN model performs a more stable training process, that is less sensitive to the architecture of the model and the selection of hyperparameters. Moreover, the mode collapse phenomenon, a typical GAN issue in which the Generator is able to produce limited varieties of the trained samples, is reduced. The most significant benefit of WGAN is the continuously estimation of Wasserstein-1 distances, by training the Discriminator till optimality. In contrast, weight clipping is not a good way to enforce Lipschitz constraint. The role of this constraint is to prevent $f$ from arbitrarily enhancing small differences and to assure that the $f$ result of two similar samples will be similar as well. If the clipping parameter is large, then the time needed for the weights to reach their limit increase. If the clipping is small, this may result to vanishing gradients problem, in which a deep NN cannot propagate important gradient information from the output of the model back to starting layers. That is the reason why authors in [74] proposed the use of gradient penalty instead of weight clipping.

### 5.1.3 WGAN-GP

Wasserstein GAN with Gradient Penalty (WGAN-GP) [74] was introduced shortly after the WGAN algorithm. The improvement of this work lies on the gradient penalty that is used to enforce Lipschitz constraint, instead the weight clipping of the WGAN. Particularly, the WGAN-GP penalizes the model when the gradient norm moves away from the target norm value of 1. The application of gradient penalty requires one more modification in the architecture. Specifically, the batch normalization is not used in the Discriminator, since the batch normalization introduces correlation between the samples of the same batch. However, the gradient penalty is calculated for each individual sample and not for the entire batch, making the batch normalization not a suitable technique. Other normalization techniques, which does not correlate the samples can be used, such as layer normalization. The WGAN-GP demonstrates strong performance as well as stability in different applications.

The loss function with Wasserstein distance and gradient penalty applied, is defined by the equation:

$$\max_{w\sim W} \mathbb{E}_{x\sim P_{data}}[f(x;w)] - \mathbb{E}_{z\sim p_z}\left[f\left(\mathrm{G}\left(z;\theta_g\right);w\right)\right] + \lambda\mathbb{E}_{Z\sim p_z}[(||\nabla_w f(G\left(z;\theta_g\right);w)||_2 - 1)^2], \tag{5}$$

where λ is the penalty coefficient.

## 5.2 GAN models for IDS

The ability of GANs to learn data distribution and then synthesize data based on this distribution can bring valuable effects in many fields. GAN models have shown great possibilities in the generation of synthetic images [75] and text [82]. Recently, extensive research has been performed in the application of GAN models in cybersecurity field. Specifically, GAN-based models can be used in different tasks in in this field, such as malware detection [83], generation of adversarial malware attacks [84] as well as in IDS. In this chapter, an analysis of GAN-based models for generation of adversarial attacks trained and validated at real-world network datasets is provided.

### 5.2.1 DoS-WGAN

DoS-WGAN [84] is a GAN-based architecture that utilizes the Wasserstein generative adversarial networks (WGAN) with gradient penalty technology with aim to generate DoS attacks that evade Network Intrusion Detection System (NIDS). The algorithm synthesizes a set of eigenvalues of traffic, preserving the characteristics of DoS attacks. Output of this model can reduce the detection rate of a network traffic classifier.

The DoS-GAN model is trained and evaluated at DoS attacks of KDD'99 [85] dataset. The architecture is composed by a Generator, Convertor and Discriminator. The Generator is trained to make Discriminator incapable to distinguish the generated data from the real. The Generator products a set of 41-dimensional values. The Convertor integrates the property values of DoS records with the created ones, creating a forged set of values for network traffic. Then, the discriminator receives the forged and the normal network traffics and attempts to distinguish them. The generated data are fed to a NIDS, in which the detection rate drops to 47.6 % from 97.3 %.

### 5.2.2 IDSGAN

IDSGAN [79] is a GAN-based framework, having as goal to generate adversarial malicious traffic examples, that are able to deceive and evade detection process of the defense systems. The architecture of IDSGAN methods consists of a Generator, a black-box IDS and a Discriminator. The training and evaluation of IDSGAN models is performed at NSL-KDD dataset, which their characteristics were considered during the preprocessing of the dataset.

For the training of IDSGAN, the NSL-KDD dataset is split to normal and malicious traffic. The adversarial malicious attacks are generated by the Generator, based on the original malicious traffic records. Then, those adversarial attacks and the original normal traffic are fed to the black-box IDS, which aims to detect attacks. Machine learning algorithms are used for the implementation of black-box IDS component of IDSGAN method. Subsequently, the Discriminator is trained to simulates the black-box IDS from the predicted labels and the original labels. Feedback from the Discriminator is given to the training procedure of the Generator. The Generator and the Discriminator are designed and developed on basis of Wasserstein GAN [73].

During the generation of adversarial examples, the attack function of the traffic is not invalidated since the functional and nonfunctional features are considered. The IDSGAN shows good performance in synthesizing adversarial malicious attacks of different attacks. Additionally, the authors of IDSGAN performed experiments, that demonstrate the wide feasibility and flexibility of IDSGAN framework.

### 5.2.3 SynGAN

SynGAN [86] is another framework that generates adversarial network attacks based on GAN models. This model synthesizes malicious packet flow mutations from real attack traffic, that can be used to improve NIDS attack detection rates. SynGAN model contains the Generator, the Discriminator and the Evaluator and it applies GP-WGAN algorithm on order to generate network Distributed Denial of Service (DDoS) attacks.

The Generator tries to generate adversarial examples similar to real attacks. Afterwards, the Discriminator attempts to distinguish the generated by the real samples and provides feedback to the Generator to improve the quality of generated samples. When the training is finished, only the Generator is used to generate adversarial DDoS attacks. In the end, the

Evaluator tries to discern the generated from the real attack record using appropriate quality benchmark.

The developed framework is evaluated at two IDS datasets, which are NSL-KDD and CICIDS2017 [87]. The quality of generation procedure is measured by benchmark based on root mean square error. The SynGAN model is able to generate similar adversarial attacks to the original attacks.

## 5.3 GAN Models for Tabular Data Generation

Over the past years, potential use of GANs have been explored for tabular data generation since they offer great flexibility to model data distributions in contrast with traditional statistical techniques. Specifically, several algorithms such as TableGAN [88], CTGAN [89] and CopulaGAN [90] proved that GANs outperform classical methods for tabular data synthetic generation. Additionally, suitable quantitative and qualitative methods to evaluate those trainable GAN models are needed.

Several GAN models have been used to handle tabular data. Kunar et al. introduced CTAB-GAN [91], a conditional table GAN that can model diverse data types with complex distributions. In [92], Passenger Name Records (PNRs) data are synthesized utilizing a Cramer GAN, categorical feature embedding and a Cross-Net architecture. The authors in [93], use GANs to generate continuous time series on Electronic Health Records (EHR) data while in [94] MedGAN, which combines an autoencoder with a GAN, is proposed to generate high-dimensional discrete variables from EHR data. TableGAN [88], consists of a convolutional Discriminator and a de-convolutional Generator and a Classifier to increase the semantic integrity of the synthetic data. In [89], CTGAN is proposed which uses a conditional generator to synthesize tabular data. CopulaGAN [90], a variation of the CTGAN model which utilizes Cumulative Distribution Function (CDF) based trans-formation to facilitate the CTGAN model training.

Giving the table Treal with real data, the task of synthetic data generation results to a synthetic table Tsyn. The T is partitioned into training set Ttrain and test set Ttest. A GAN model is trained at Ttrain. The data generator G learns the data distribution of each column in a table T and then it is used to generate a synthetic data of the table Tsyn. A successful data generator G for tabular data should be able to address the challenges associated with the nature of real-world tabular data.

It is common, that the T contains mixed data types of numerical and categorical columns. The numerical columns of table can have either discrete or continues values. Thus, the Generator G should be trained to simultaneously learn and generate a mix of data types. Additionally, the shape distribution of each column may differ, following usually non-Gaussian and multimodal distributions, where the min-max transformation causes vanishing gradient problems. In categorical columns of real-world tabular data, the imbalance problem often occurs since some classes have more instances than others. Imbalanced data leads to mode collapse as well as to inadequate training of the minor classes. Furthermore, sparse one-hot-encoded vectors can cause issues at the training procedure of the Discriminator D since it learns to distinguish real from fake data from the distribution's rareness rather to the realness of the value. Different GAN models have been created in order to solve some or all of the above aforementioned issues. TGAN [95] is constructed to work on any tabular dataset, while MedGAN [94] tries to generate simultaneously discrete and continuous samples.

### 5.3.1 TableGAN

TableGAN [88] is a GAN-based architecture which synthesizes fake tabular data with similar statistical properties to the original table. Privacy concerns motivate the authors to develop this model to prevent information leakage. The discriminator D and the generator G of the TableGAN are convolutional neural networks (CNN). The architecture of TableGAN is based on the deep convolutional GAN (DCGAN) [75], which is one of the most popular models for image synthesis.

Another architectural component of TableGAN is the classifier C, which is involved in the training process and aims to increase the semantic integrity of generated records. The classifier C has the same architecture as the discriminator D, while it is trained based on the ground-truth labels of the real table. C learns the correlation between true labels and the features of the table and predicts the labels of the synthetic data. Thus, C educates the generator G if the generated record is semantically correct.

### 5.3.2 CTGAN

Conditional Tabular GAN (CTGAN) [89] is a GAN-based architecture, designed to synthesize tabular data. The key improvements of CTGAN try to overcome the challenges of modelling tabular data using GAN architecture. In particular, the architecture of CTGAN deals with non-Gaussian and multimodal distribution by exploiting a mode-specific normalization, which converts continuous values of arbitrary distribution into a bounded vector, a representation suitable for neural networks. Previous models, such as TableGAN [88], normalize continuous values to [-1, 1] using min-max normalization techniques. In CTGAN, the variational Gaussian mixture model (VGM) [96] is used for each continuous column independently.

Additionally, a conditional generator and training-by-sampling is implemented to overcome the data imbalance challenge of discrete columns. The data is sampled in a way that all the categories of discrete columns are evenly occurred during the training procedure. A *cond* vector allows the conditioning on a value of a specific column via one-hot-encoding. The conditional generator G takes as inputs random noise as well as the *cond* vector, while it is forced to mimic the desired condition. The training of the model is done using the WGAN loss with gradient penalty [74]. The output of conditional generator is evaluated by the critic, computing the distance between the learned and real conditional distribution.

### 5.3.3 CopulaGAN

CopulaGAN model [90] is a variation of the CTGAN, which is introduced in SDV opensource library. It exploits the Cumulative Distribution Function (CDF) based transformation, that is applied via the GaussianCopula. Particularly, the CopulaGAN uses those alternatives of CTGAN in order to learn easier the data. Based on probability theory, copulas are used to describe the intercorrelation between random variables.

During the training procedure, CopulaGAN tries to learn the data types and the format of the training data. The non-numerical and null data are transformed using a Reversible Data Transformation (RDT). Due to this transformation, a fully numerical representation occurs from which the model can learn the probability distributions of each table column. Additionally, the CopulaGAN attempts to learn the correlation between the columns of the table.

## 5.4 Evaluation Metrics for Tabular Data Generation

Recent advances of generative modelling identified the need for suitable quantitative and qualitative methods to evaluate trainable models. Reliable evaluation metrics are important not only to rate GAN models but also to identify possible errors in the generated data. Specifically in cases where people face difficulties distinguishing the quality of synthetic data, such as medical images, the requirement for trusted metrics is essential [97].

Evaluating a GAN model is not a straightforward procedure since various metrics can lead to different outcomes. Specifically, a good performance in one evaluation metric cannot guarantee good performance in another metric [98]. Additionally, the metrics should be chosen with respect to the application that are going to be used for. Inception Score [99], Fréchet Inception Distance [100] and Perceptual Path Length [101] are some metrics that are introduced for the evaluation of general GAN models.

The synthetic data would be evaluated against a sufficient number of metrics that are suitable for the task of tabular data generation. A combination of those methods can express a complete picture about the performance of the generator G of different GAN models. The evaluation is performed on the table of real data Treal as well as on the table of synthetic data Tsyn, which are generated from the trained generator G. Metrics can be categorized into two subcategories: Visual, and Machine Learning based.

### 5.4.1 Visual evaluation

Visual representation of the generated data is a powerful method to evaluate the performance of the generator G, by analyzing if G is able to maintain the properties of the real data. Based on this, humans can easily verify results and recognize similar patterns between real and synthetic data. Additionally, the visual analysis of results provides information that cannot be covered from the quantitative metrics. The visual evaluation can be based on Distribution, Cumulative Sums, and Column Correlation.

The *Distribution* plot of each column for real and synthetic data can be a quick sanity check, although it does not reveal any hidden relation. This representation can point out if the statistical properties of the generated and real data are similar to each other.

The *Cumulative Sum* of each column for real and generated data can be visualized to indicate the similarity between the distributions per column. This visualization can present a useful understanding for both categorical and continuous columns. However, this representation cannot provide any insight about the relations between columns.

Another evaluation method can be based on the *Correlation* table, which shows the association between each column of the table. Comparing the correlation matrix of the real and synthetic data can indicate if the generator manages to appropriately model the relationship between the columns of the table.

### 5.4.2 Machine Learning-Based Metrics

This family of metrics exploits Machine Learning algorithms to evaluate the quality of the generated data. They are able to provide insight knowledge about the relations that Treal and Tsyn have. Particularly, GANs for tabular data generation task are evaluated by the detection metrics as well as Machine Learning efficacy metrics as they are described in [102].

The *Detection Metrics* evaluate how difficult is to differentiate the generated from the real data. Specifically, those metrics are based on Machine Learning models, which predict if the input data is synthetic or real. For this reason, a flag is associated to each data record, indicating if it is real or generated. Afterwards, the data with the flags are shuffled and the Machine Learning models are cross-validated, attempting to predict the flag. Finally, the result of those metrics equal to 1 minus the average ROC AUC score of all cross-validation splits. The Machine Learning models that can be used in those metrics are Logistic Regression or SVD classifier.

The *Machine Learning Efficacy Metrics* indicate if it is possible to replace the real with generated data to solve problems using Machine Learning models. In particular, a model is trained on Tsyn, and then, it is tested on Treal. In case of classification problems, Decision Tree, AdaBoost, or MLP classifier can be used, while the performance of those models is evaluated based on accuracy and F1 score. For regression tasks, Linear Regression or MLP regression may be utilized as machine learning models, and the evaluation is performed by R2. The average performance of different models can be used as metric for the evaluation of G.

As it is mentioned above, those metrics are occurred by solving Machine Learning problems. Therefore, they can only be applied on datasets that contain a target column, which should be predicted based on the rest of the data. The target column could contain true labels or ground truth values for the classification and regression task, respectively.

## 5.5 Intrusion Detection System (IDS) Datasets

In this chapter, an analysis of the most well-known available IDS datasets is provided, including the KDD'99 [85] , the NSL-KDD [103] and the UNSW-NB15 [104].

### 5.5.1 KDD'99

KDD'99 [85] is a very common IDS dataset, which have been extensively used in research, especially for intrusion detection in network traffic. The dataset contains 41 features, and it has 5 classes, which are Normal, Denial of Service (DoS), Probe, Remote to Local (R2L) and User to Root (U2R). The KDD'99 includes 4,898,430 and 311,029 records in training and testing set respectively, making it the largest IDS dataset. Training and testing sets are highly imbalanced since the DoS class has many more records than the second most frequent class, that is Normal. Another characteristic of KDD'99 is that the testing set includes a higher amount of R2L records than the training set. However, the most critical issue of this dataset is the large number of duplications records, specifically for DoS class. This can lead to unstable and inconsistent training procedure of machine learning algorithms.

## 5.5.2 NSL-KDD

The NSL-KDD [103] is an Intrusion Detection System dataset, which is an improved version of its predecessor, the KDD'99 [85] dataset, which suffers from a large number of duplicate records. Specifically, the NSL-KDD dataset does not contain redundant records in the train and test sets. Therefore, classifiers that would be trained and tested at those datasets would not be biased based on the most frequent records. Another advantage of NSL-KDD over the KDD'99 is the reasonable amount of data in the train and test sets, providing the opportunity to perform experiments on the complete sets. The NSL-KDD is one of the most well-known publicly available datasets since it is used by many researchers to develop efficient and accurate Intrusion Detection Systems. The labeled attacks of the NSL-KDD dataset can be grouped in four main types, namely Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L).

## 5.5.3 UNSW-NB15

The UNSW-NB15 [104] is a relatively new dataset, published in 2015. The dataset has 49 features and ten classes, included some modern attacks. The features of UNSW-NB15 are grouped in five parts, which are Basic, Flow, Time, Content and Additional features. The classes are Normal, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. The classes of UNSW-NB15 are also imbalanced, with Normal class to have most records, following by records of Generic and Exploits class.

# 6 IoT-NGIN GAN-based dataset generator

Throughout this section, the procedures that takes place over the IoT-NGIN project to design, develop and evaluate the GAN-based data generation models, are going to be described. Firstly, an analysis of GAN-based dataset generators regarding the technical design aspects is provided, including feature engineering, data preprocessing, and the generation of adversarial DoS attacks. Then, a detailed overview of hands-on experiments for GAN-based model on IDS data generation are presented, containing visual evaluation as well as the Machine Learning Metrics of CTGAN, CopulaGAN and TableGAN. Finally, installation guidelines provide all the needed information on how to properly install the IoT-NGIN GAN-based dataset generation component, combined with user guidelines on how to successfully run and evaluate the models.

## 6.1 Technical design

This section describes the processes, which are been considered during the design and implementation of robust GAN-based models to generate high quality DoS attacks. Specifically, feature engineering and data preprocessing procedures are analyzed as well as the reasons that certain decisions are made. Lastly, information about the data generation procedure is given.

### 6.1.1 Feature engineering

The generation of meaningful synthetic IDS data requires the exploitation of appropriate features. The processes of using domain knowledge to extract those needed features from the NSL-KDD dataset are presented in this section. Specifically, this decision making requires an analysis of the dataset as well as a deep understanding of the limitations and requirements for adversarial attacks generation procedure.

Therefore, the types of NSL-KDD attacks are presented, following by a statistical analysis of the dataset. Then, the properties of NSL-KDD features are described. Based on those properties and the attacks' principles, the features are divided to functional and non-functional, making the last ones appropriate inputs to the generation models.

#### 6.1.1.1 Types of attacks

The attacks of the NDL-KDD dataset can be categorized in four main types, which are Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L), which have been described in Section 2.2.2.3. Table 1 illustrates the type and the labeled attacks of the NSL-KDD dataset.

Table 8: Types of attacks in the NSL-KDD dataset.

| Type | Labeled Attack |
|---|---|
| **DoS** | neptune, back, land, pod, smurf, teardrop, mailbomb, apache2, processtable, udpstorm, worm |
| **Probe** | ipsweep, nmap, portsweep, satan, mscan, saint |
| **R2L** | ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster, sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, httptunnel |
| **U2L** | buffer_overflow, loadmodule, perl, rootkit, ps, sqlattack, xterm |

#### 6.1.1.2 Statistical analysis of dataset

The NSL-KDD dataset [103] contains 148,514 traffic records of normal activities and attacks. The dataset includes the train set KDDTrain+ and the test set KDDTest+. The training and the testing datasets do not have the same distribution, while the testing dataset contains some attacks that are not included in the train dataset.

From analysis of the NSL-KDD dataset, it is observed that there are 77,052 normal traffics, which is more than half of the total records. Regarding the attack records, DoS type is the most frequent with 53,386 instances, Probe type has 14,077 instances, while there are a few traffic records for R2L and U2R attacks, 3880 and 119 instances, respectively. Figure 5 illustrates the distribution of records, normal and attack types, in the NSL-KDD dataset.

DoS is one of the most common types of attack, and it frequently occurs in everyday life. This characteristic of DoS attacks is reflected in the NSL-KDD dataset, in which this category contains the majority of attack records. Additionally, the detection of DoS attacks is a crucial challenge that needs thorough investigation.

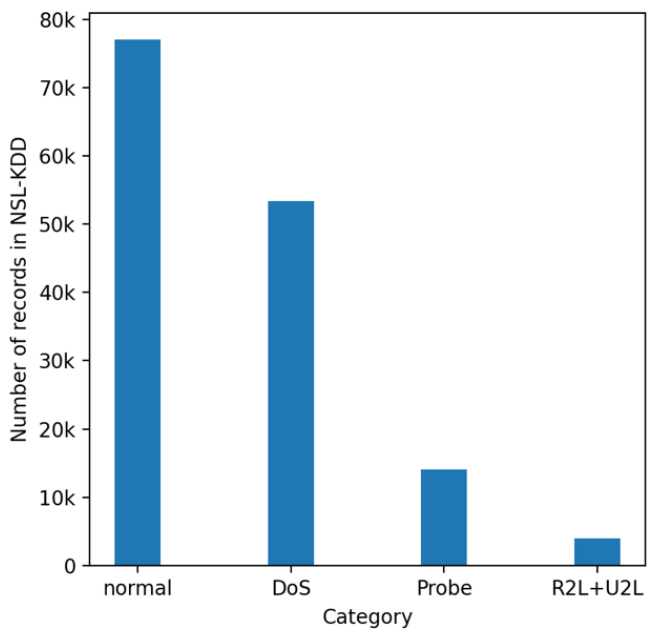


Figure 5: Number of each category for NSL-KDD dataset.

### 6.1.1.3 Features of NSL-KDD dataset

Each traffic record of the NSL-KDD dataset has 41 features, nine of them have categorical values, while the rest of the features are in discrete or in continuous values. The categorical features of NSL-KDD are mentioned in Table 9.

Table 9: Categorical features of NSL-KDD dataset.

| | |
|---|---|
| **Categorical Features** | protocol_type, Service, Flag, land, logged_in, root_shell, is_host_login, is_guest_login, su_attempted |

The NSL-KDD features can be broken down into four categories, according to the type and property of the features. A short description of the categories is given below, while the Table 11 contains the features of each category.

- *Intrinsic features* (9): includes necessary information of the record, such as protocol, service, and duration.
- *Content features* (13): comprise information about the content, such as the login activities. Those features demonstrate if there are behaviors related to attacks.
- *Time-based features* (9): contains the number of connections to the same destination host or service as the current connection in the past two seconds.
- *Host-based features* (10): checks the 100 past connections, which have the same destination host or service with the current connection.

Table 10: The four groups of features in the NSL-KDD dataset.

| Feature category | NSL-KDD features |
|---|---|
| **Intrinsic** | duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent |
| **Content** | hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login |
| **Time-based** | count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate |
| **Host-based** | dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate |

#### 6.1.1.4 Constraints for the Generating Adversarial Examples

The generated tabular data should represent attacks that have evading IDS as their purpose; however, in order to achieve this, the generation process should take account and maintain the functional characteristic of each attack category [79]. Based on attack principles, each category of attacks has its functional and nonfunctional features. The functional features describe the basic function of the attack, while the nonfunctional represent the secondary characteristics of the attack. The attack properties remain undistributed when the functional features do not change, and only the nonfunctional features are modified. Thus, to achieve reliable and valid generated attack records, the functional features should be unchanged, while the nonfunctional features can be modified. Therefore, the GAN models should be trained and then generate only the non-functional features, taking into account the examined attack category. Table 3 illustrates the functional features of each attack category of the NSL-KDD dataset, as it is discussed in [105].

Table 11: The functional features of each attack category.

| Attack Category | Intrinsic | Content | Time-Based Traffic | Host-Based Traffic |
|---|---|---|---|---|
| **DoS** | x | | x | |
| **Probe** | x | | x | x |
| **U2R** | x | x | | |
| **R2L** | x | x | | |

### 6.1.2 Data preprocessing

An essential step in the ML cycle is data preprocessing, which refers to the techniques of preparing the raw data to be used for building and training ML models. Specifically, data preprocessing may contain cleaning, filtering, organization or transforming procedures. To build robust IoT-NGIN GAN-based generator, data preprocessing techniques are performed at NSL-KDD dataset.

Firstly, data cleaning is performed, mainly regarding the process of fixing incorrect values in the features of the dataset. The columns names of the features are converted to lower case and spaces are removed. Furthermore, the "su_attempted" feature of NSL-KDD should be binary, however after data exploration, it is observed that this feature has 3 values, namely 0.0, 1.0 and 2.0. Therefore, the last value is replaced to '0.0' for both train and test datasets.

The next step is related to the organization of the data. The presented GAN models are focused on the application of different GAN models for the generation of synthetic DoS attacks. Thus, the traffic records of the NSL-KDD dataset are divided into normal and attack classes based on the 'class' column, while the attacks are separated to four categories: DoS, Probe, R2L, and U2R. The records of DoS attack are selected for the training and evaluation GANs. Considering the constraints for the generation of adversarial attacks, the features of each DoS record of NSL-KDD are split into functional and nonfunctional based on the Table 11. The nonfunctional are used as the training set of GAN models, while the functional ones remain unmodified at the generated samples.

To improve the quality of the GAN-based generators, further data organization techniques are performed. Specifically, the nonfunctional features with categorical values are identified since the training of GANs requires this knowledge to model the data appropriately. Additionally, the distribution of each feature is determined since the training of CopulaGAN is improved by providing the distribution of features that is not Gaussian.

### 6.1.3 Generation of tabular data

Experiments are designed to investigate the general properties and performance of the different GAN models for the task of generation of synthetic IDS data. In particular, the CTGAN [89], CopulaGAN [90], and TableGAN [88] are trained at the NSL-KDD dataset, taking into account the analysis and the preprocessing steps, which are described in subsections 6.1.1 and 6.1.2. For our experiments, we use the GAN models, which are provided by the open-source synthetic data generation ecosystem SDV–The Synthetic Data Vault [106] and Synthetic Data Gym (SDGym) [107]. Each model is trained with a batch size of 500 and for 100 epochs. The learning rates for the generator and discriminator of all models are both 0.0002. Additionally, the discriminator steps of CTGAN and CopulaGAN are set to 5. The training inputs to GAN models are the nonfunctional features of DoS attacks. Consequently, the trained models are able to generate the nonfunctional features, which afterwards are coupled with the functional ones to create meaningful DoS records.

Then, the synthetic datasets generated by the trained GAN models are evaluated using the metrics, which are described in Section 5.4 of this deliverable. The Distribution and Cumulative Sum plots are created using the Table Evaluator [108] library, while the Machine Learning-based metrics are calculated based on the Single Table Metrics of the SDV library [102].

## 6.2 Hands-on experiments

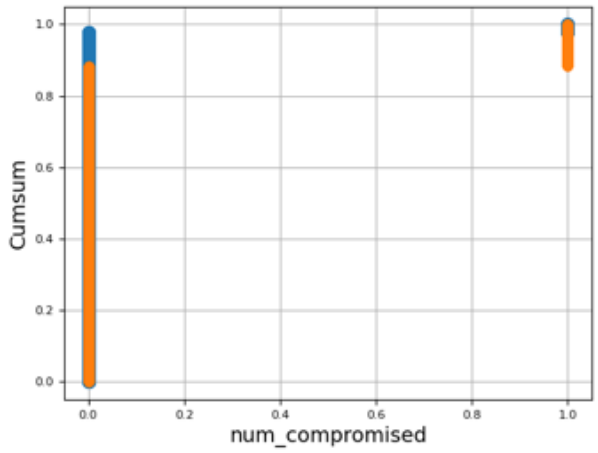
In this subsection, visual evaluation, and machine learning-based metrics, related to the performance of generated DoS attacks, are provided.

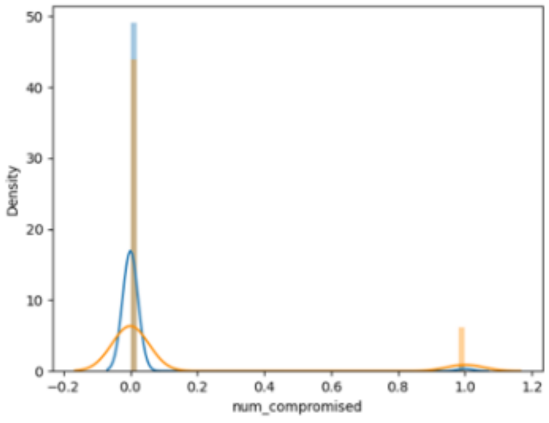
### 6.2.1 Visual evaluation

The performance of GAN models for the generation of DoS attack records can be demonstrated visually by cumulative sum and distribution plots. Indicatively, the evaluation plots of two discrete features and two continuous features are displayed in order to summarize the quality of synthetic data. Figure 6 and Figure 7 show the plots of discrete features, namely "num_compromised" and "hot". It is observed that both CTGAN and CopulaGAN outperform TableGAN in the case of discrete values. Cumulative sum and distribution plots of CTGAN and CopulaGAN indicate that the generated discrete features of data present similar behavior with the real one. On the other hand, the corresponding plots of TableGAN show that this method cannot model the behavior of discrete features.

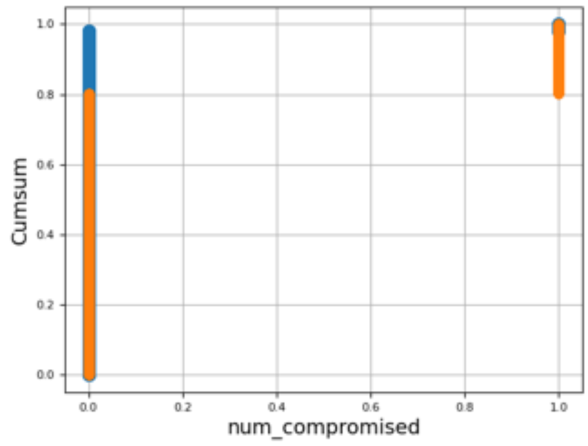
***Discrete features***

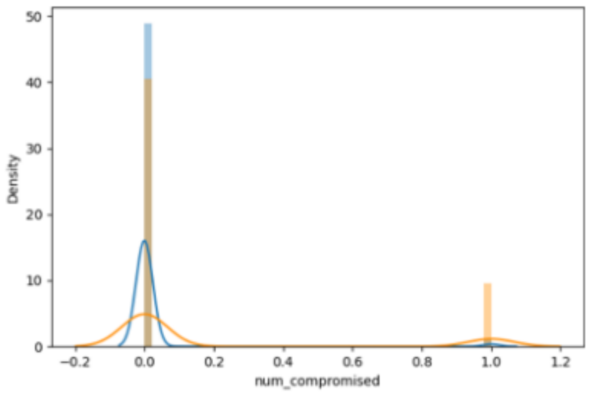
CTGAN





CopulaGAN





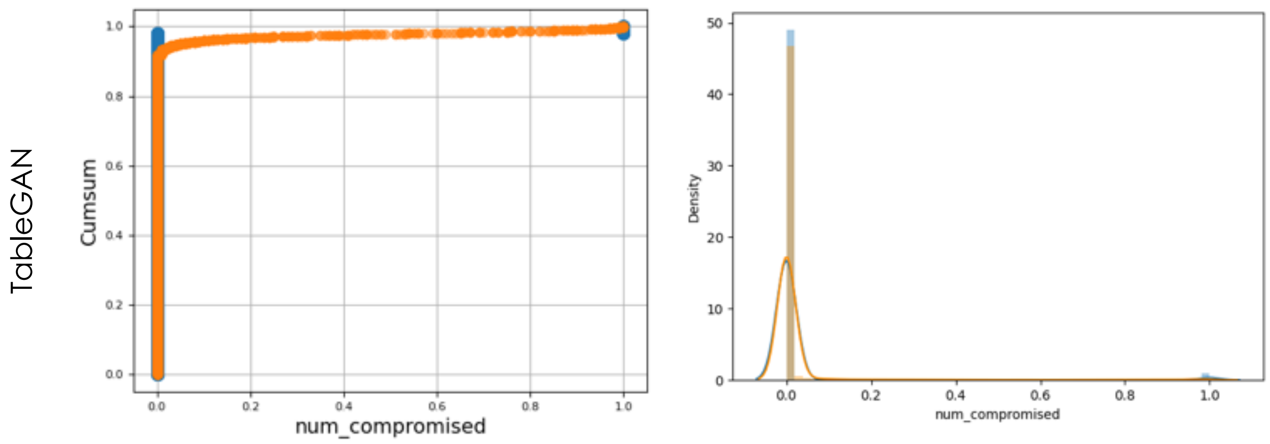
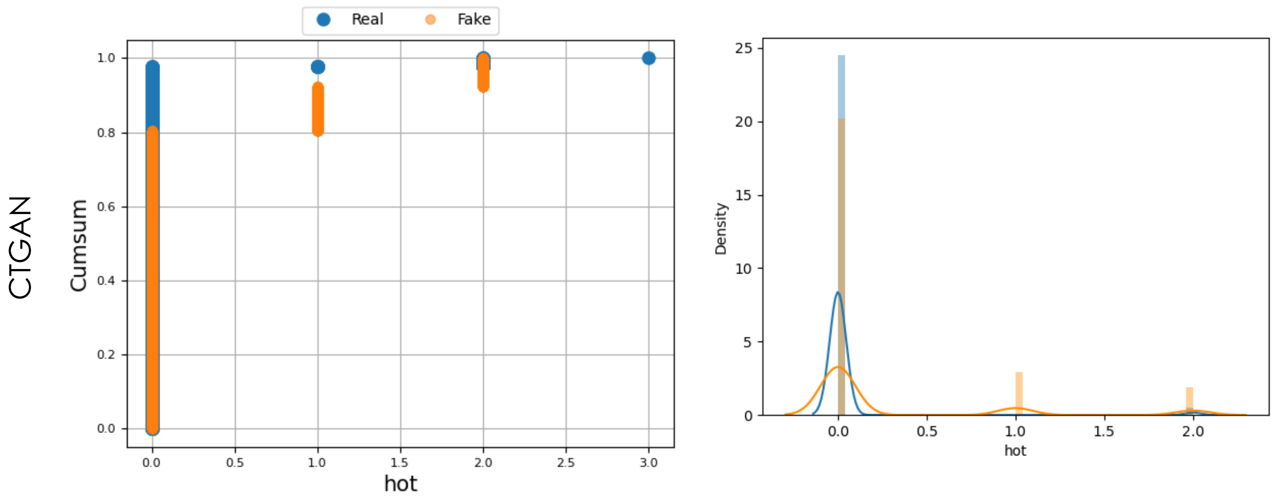
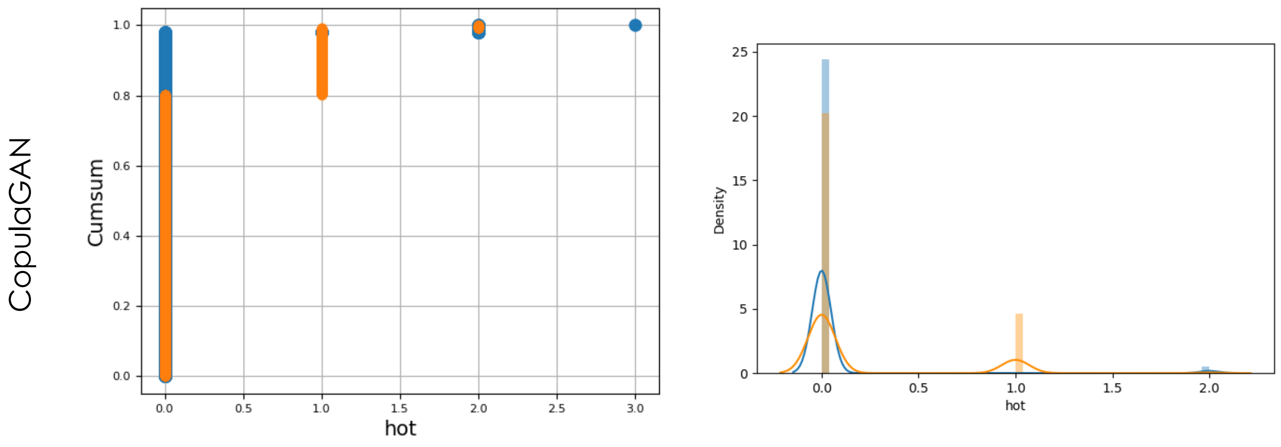


Figure 6: Cumulative Sums and Distributions for discrete feature "num_compromised". Real data are illustrated in blue, while synthetic are illustrated in orange color.





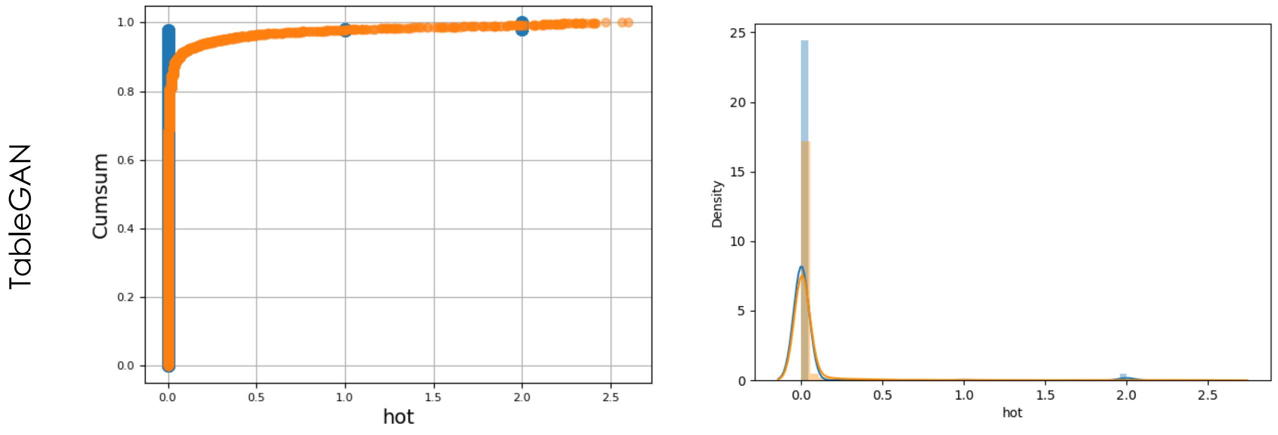
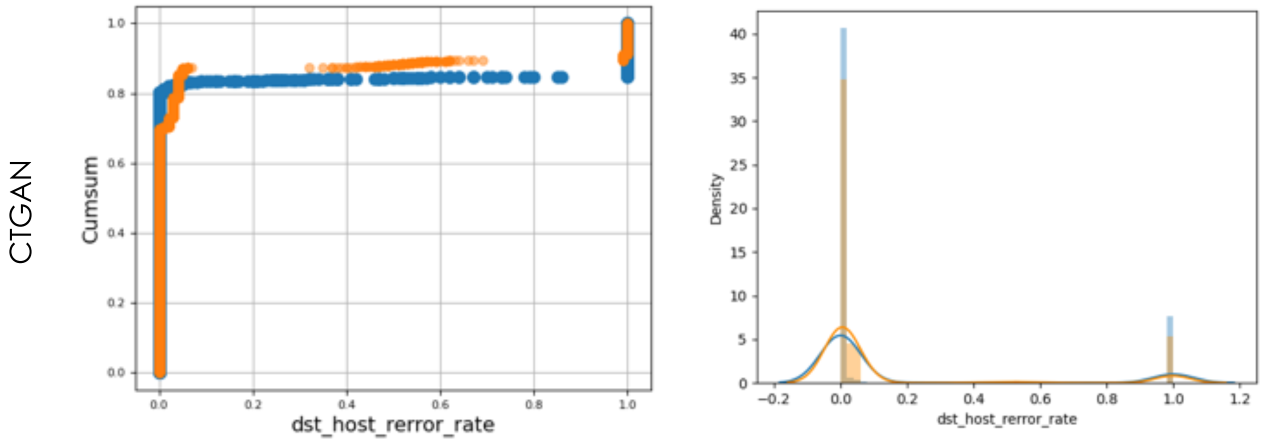


Figure 7: Cumulative Sums and Distributions for discrete feature "hot". Real data are illustrated in blue, while synthetic are illustrated in orange color.

***Continuous features***

Figure 8 and Figure 9 illustrate the corresponding diagrams for continuous features. In those cases, the TableGAN achieves slightly better performance than CTAGN and CopulaGAN. However, as it is depicted in Figure 9, all GAN methods suffer when modeling continuous features with sparse data.



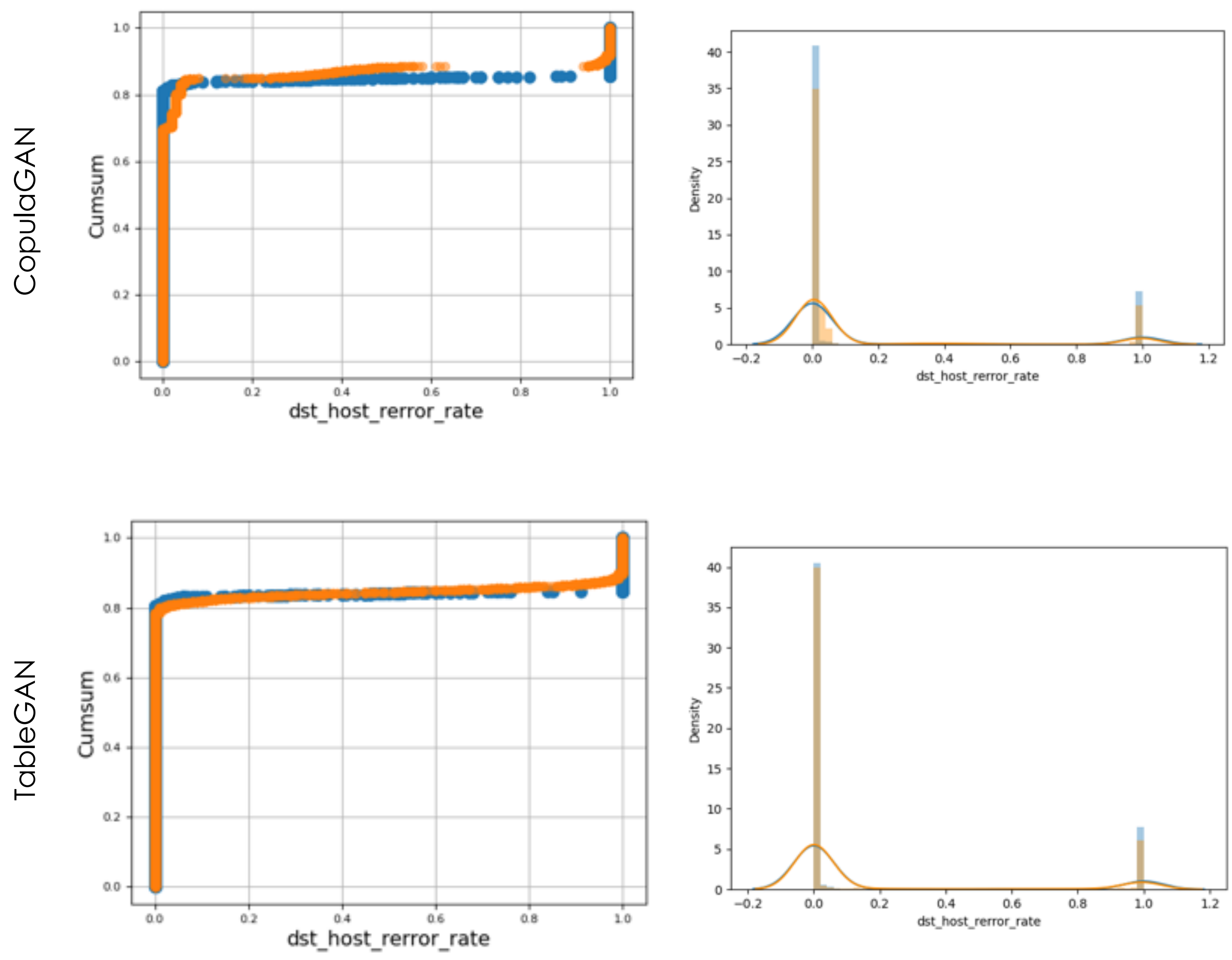
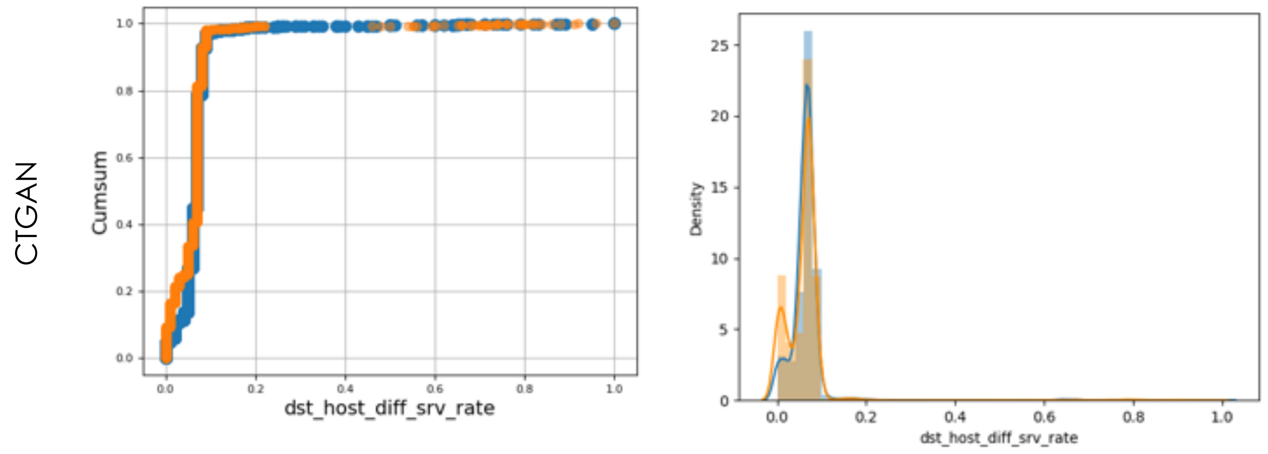


Figure 8: Cumulative Sums and Distributions for continuous feature “dst_host_rerror_rate”. Real data are illustrated in blue, while synthetic are illustrated in orange color.



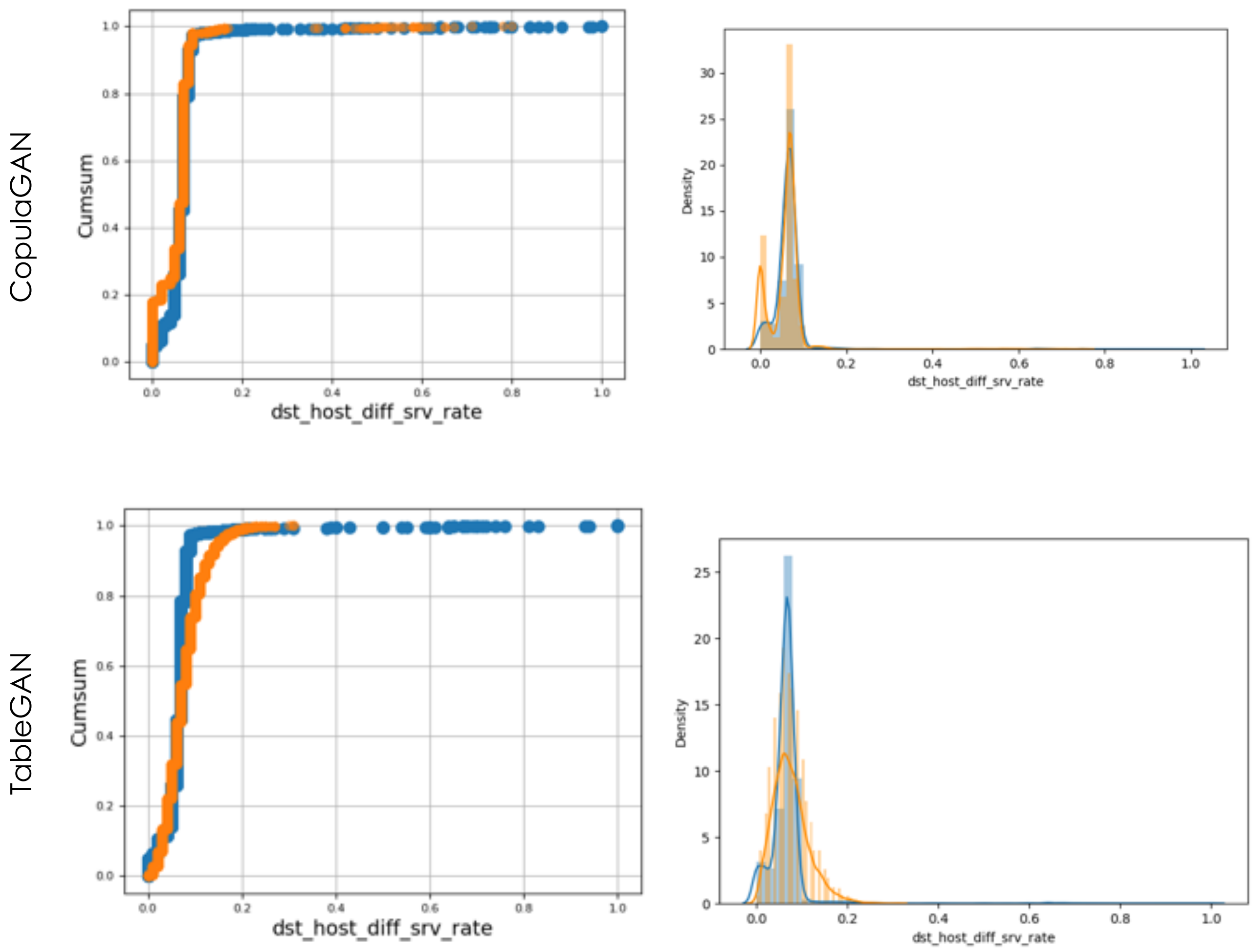


Figure 9: Cumulative Sums and Distributions for continuous feature "dst_host_srv_diff_host_rate". Real data are illustrated in blue, while synthetic are illustrated in orange color.

***Correlation matrices***

Figure 10 illustrates the column wise correlation of real data as well as the generated one from CTGAN, CopulaGAN, and TableGAN. The columns with features that contain zero values are eliminated from this representation. Ideally, the correlation matrix of generated data should be as similar as possible to the real data correlation table. Therefore, the correlation matrix of generated data is compared to the correlations of real data and the methods, that succeed to maintain the real data correlation, are considered as successful. Generally, all the models are able to adequately capture the correlations between features. Although, as it is observed, TableGAN faces difficulties in capturing some of the correlations.

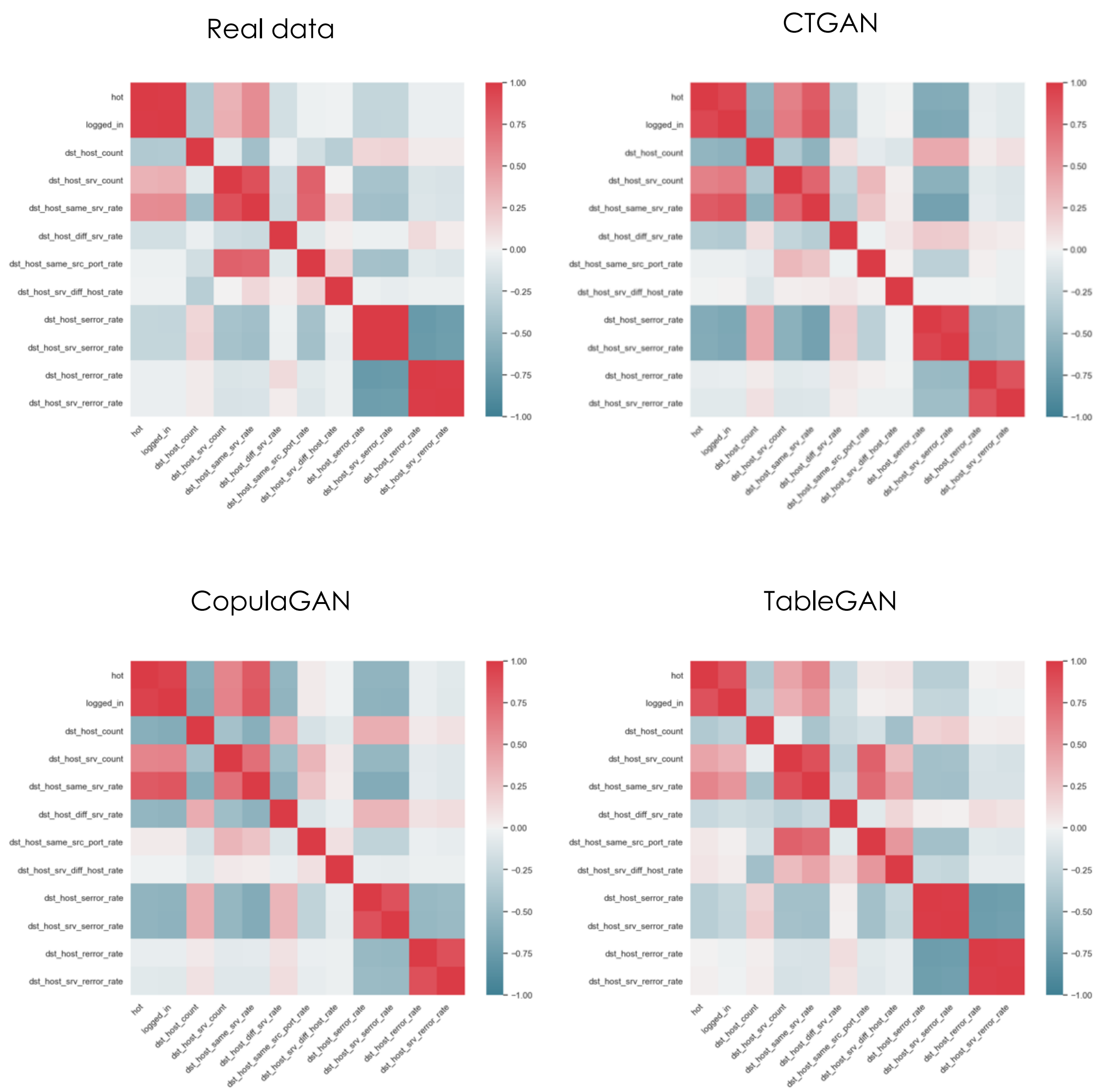


Figure 10: Correlation matrices, indicating the associations per column of the real dataset and each of the synthesizers.

## 6.2.2 Machine Learning-Based Metrics

The Machine Learning-Based metrics of the three models are demonstrated in Table 12. It is observed that for all the models, the Detection Metric indicates that the Logistic Regression Classifier finds it moderately difficult to distinguish the real from the generated data. Therefore, the real and the generated data are distinguishable to one degree. Finally, regarding the Machine Learning Efficacy Metrics, it seems that all the models indicate similar performance, showing that it is possible to replace the real with synthetic data to solve Machine Learning problems.

Table 12: Detection and Machine Learning *Efficacy* Metrics for CTGAN, CopulaGAN and TableGAN.

| | Detection Metric | Machine Learning Efficacy Metrics | | | | |
|---|---|---|---|---|---|---|
| | Logistic Regression | Decision Tree | AdaBoost | Logistic Regression classifier | MLP classifier | Average |
| CTGAN | 0.74 | 0.97 | 0.96 | 0.84 | 0.96 | 0.93 |
| CopulaGAN | 0.75 | 0.97 | 0.95 | 0.92 | 0.96 | 0.95 |
| TableGAN | 0.76 | 0.95 | 0.94 | 0.92 | 0.97 | 0.94 |

## 6.2.3 Comparative performance analysis of GAN-based models

Based on a thorough examination of models' performance with respect to various aspects, the Table 13 is generated. Regarding the ability of methods to model discrete features, it is observed that CTGAN and CopulaGAN show much better performance than TableGAN, which is incapable to perform this task. On the other hand, TableGAN indicates the strongest ability to capture the statistical properties and then generate realistic continuous features among the other models. CopulaGAN seems that is capable, to one extent, to model continuous features, while CTGAN presents slightly poor behavior. Both of CTGAN and CopulaGAN display a limited ability to model sparse continuous values.

Regarding the performance of GAN models on ML-based metrics, all of them indicate similar behavior. The Logistic Regression classifier has almost the same moderately difficulty to differentiate the generated from the real data. Additionally, the generated IDS data can replace the real one to solve Machine Learning problems. To conclude, considering the aforementioned, the CopulaGAN seems to have the best in-total performance among the others, mainly based on its ability to satisfactorily model both discrete and continuous values.

Table 13: Comparison table of models' performance.

| | CTGAN | CopulaGAN | TableGAN |
|---|---|---|---|
| Discrete features | ✓ | ✓ | x |
| Continuous features | - | ✓ | ✓ |
| Detection metric | - | - | - |
| ML Efficacy metrics | ✓ | ✓ | ✓ |
| In total | | ✓ | |

## 6.3 Installation guidelines

This subsection describes the processes to install the developed GAN-based components. It should be mentioned, that to build and train CTGAN and CopulaGAN models, the open-source Synthetic Data Vault (SDV) library [106] is used, which is a synthetic data generation ecosystem of libraries, providing the means to learn tabular data and then to generate synthetic data with the same statistical properties as the original one. For the development of TableGAN, the Synthetic Data Gym (SDGym) [107] is exploited, which is a framework to benchmark the performance of data generators based on SDV ecosystem.

The visual evaluation of trained models is performed based on Table Evaluator [53] library, while the Machine Learning-based metrics are calculated based on the Single Table Metrics of the SDV library [102]. The code and the installation procedures are available on the project Gitlab repository, accessible at https://gitlab.com/h2020-iot-ngin, and a brief description is given here.

The developed methods can be installed by the following command:

```
git clone https://gitlab.com/h2020-iot-ngin/enhancing_iot_cybersecurity_and_data_privacy/gan-based-data-generators/ids-data-generator.git
```

A folder ***"ids-data-generator"*** is created, containing the following subfolders:

- CTGAN_copulaGAN: the code to train CTGAN and CopulaGAN
- data: the NSL-KDD data
- evaluation: the code to evaluate CTGAN, CopulaGAN and TableGAN
- sdgym: a modified version of SDGym repo [107], including model save function and some other minor changes
- tableGAN: the code to train tableGAN
- utils: developed functions and the modified Table Evaluator code, which initially is cloned by the repo[6] and adjusted to the evaluation needs of models

After the execution of each model training and evaluation script, the following files will also be created:

- models: the trained models
- results: plots of cumulative sums, distributions, and correlation matrices for the three models
- synthetic_dataset: one .csv file of synthetic DoS attacks for each model

To avoid having conflicts with other software installed in the system, it is recommended to create two different conda environment[7] for each project. The creation of the two different conda environments as well as the running guidelines for each component are provided below.

***CTGAN_copulaGAN component***

[6] https://github.com/Baukebrenninkmeijer/table-evaluator

[7] https://docs.conda.io/projects/conda/en/latest/index.html

Create a conda environment

```
cd CTGAN_copulaGAN
conda create -n sdv python=3.7.9
conda activate sdv
pip install -r requirements.txt
```

The training of the model is done by executing the following commands:

```
python CTGAN_train.py   # CTGAN model training
python copulaGAN_train.py # CopulaGAN model training
```

Evaluation of models:

```
cd ../evaluation
python CTGAN_evaluation.py # CTAGAN model evaluation
python copulaGAN_evaluation.py  # copulaGAN model evaluation
```

***TableGAN components***

Create a different conda environment, based on the requirements of SDGym framework:

```
conda deactivate
cd ../tableGAN
conda create -n sdgym python=3.7.9
conda activate sdgym
pip install -r requirements.txt
```

Training of TableGAN model:

```
python tableGAN_train.py
```

To evaluate the TableGAN model, the sdv conda environment should be activated:

```
conda deactivate
conda activate sdv
python tableGAN_evaluation.py
```

Access the outcomes of the training and evaluations procedures by:

```
cd ../models  # the trained models
cd ../evaluation  # visual evaluation plots
cd ../synthetic_dataset  # synthetic DoS attacks
```

# 7 IoT-NGIN IoT vulnerability crawler

According to ENISA, a security vulnerability is "*a weakness an adversary could take advantage of to compromise the confidentiality, availability, or integrity of a resource*" [40]. Tightly coupled with the notion of security threats that are defined as "*any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, or the Nation through an information system via unauthorized access, destruction, disclosure, or modification of information, and/or denial of service*" [109] and attack defined as "*any kind of malicious activity that attempts to collect, disrupt, deny, degrade, or destroy information system resources or the information itself*" [110], the management of vulnerabilities is a proactive rather than a reactive task, since there are no explicit hints of their existence; instead, active and targeted vulnerability scanning should be pursued in order to identify and classify the existing vulnerabilities of a system. Indeed, from a process perspective, vulnerability management should be performed in a step-wise manner, ENISA identifying 5 relevant steps, as per Table 14.

Table 14: Vulnerability management process [40].

| Step | Descripition |
|---|---|
| **Preparation** | Define the scope of the vulnerability management process. |
| **Vulnerability Scanning** | Vulnerability scanners are automated tools that scan a system for known security vulnerabilities providing a report with all the identified vulnerabilities sorted based on their severity. Known vulnerability scanners are Nexpose, Nessus and OpenVAS. |
| **Identification, Classification and Evaluation of the Vulnerabilities** | The vulnerability scanner provides a report of the identified vulnerabilities. |
| **Remediating Actions** | The asset owner determines which of the vulnerabilities will be mitigated. |
| **Rescan** | Once the remediating actions are completed, a rescan is performed to verify their effectiveness. |

The vulnerability management steps of ENISA imply thoughtful and targeted vulnerability management. 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 tackle these challenges by designing and implementing a dynamic vulnerability crawler able to scan for vulnerabilities the IoT-NGIN-supported IoT devices and services under three different modalities:

- On demand,
- On first appearance,

- On a time-scheduled basis.

In the following paragraphs, the technical specification and high-level design of this vulnerability crawler are presented.

## 7.1 High-level architecture and specification

The aim of the vulnerability crawler is to scan devices and services in order to identify possibly vulnerable services executed in either. To tackle with 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 depicted in Figure 11.

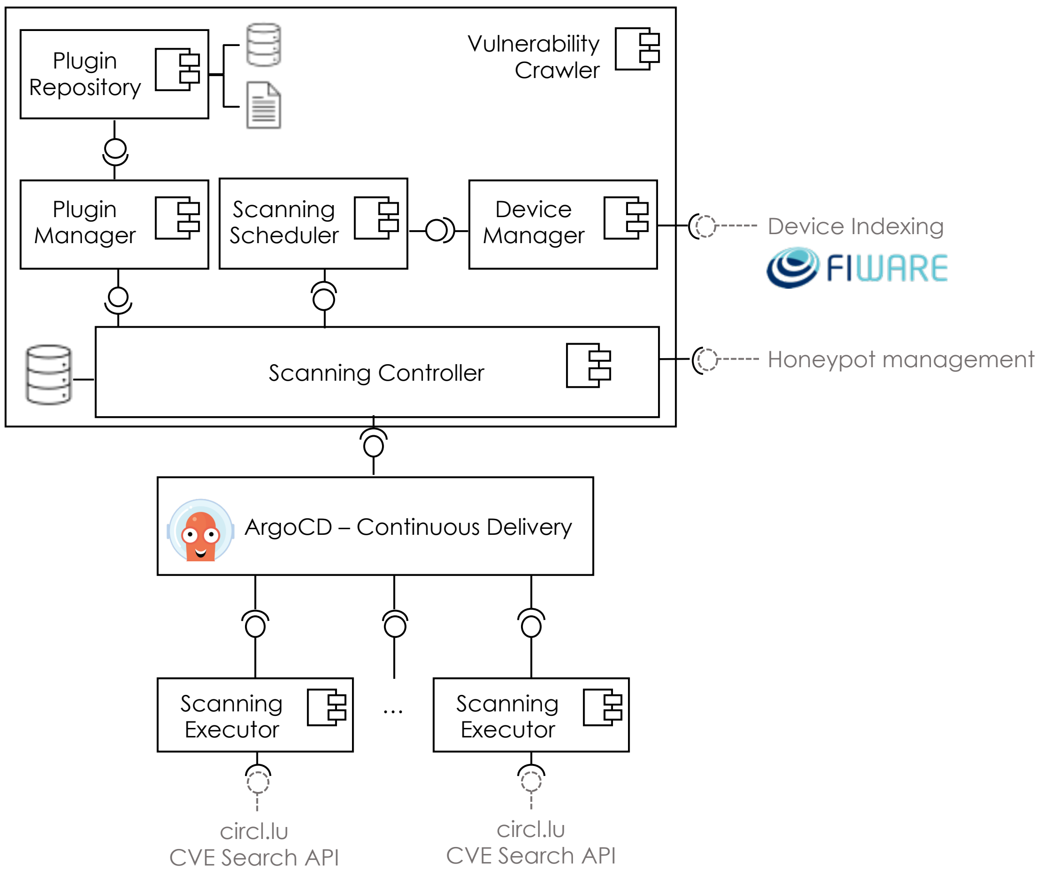


Figure 11: High-level architecture of the vulnerability crawler.

Five core subcomponents have been determined, collectively interworking to achieve the functionality of the vulnerability crawler as a whole, as per Table 15. Note that each subcomponent has been designed as a stateless microservice, the necessary state management being operated by a dedicated DB. The adopted microservices design is compliant to the state-of-the-art microservices-oriented architectures definition [111], which matches the cloud-native principles and requirements [112], effectively granting the vulnerability crawler with resilience, manageability, observability, adaptability, predictability and scalability characteristics [113].

Table 15: Subcomponents of the vulnerability crawler.

| Sub component | Description | Input | Output |
|---|---|---|---|
| Device Manager | Gathers and maintains information about devices and services known to the vulnerability crawler framework | Device information from the Device Indexing component (WP4) | Device information to the Scanning Scheduler |
| Scanning Scheduler | Schedules vulnerability scans, either on demand, or following a schedule, effectively enabling time-driven IoT services | Device information from the Device Manager. Can also be self-initiated, when dealing with scheduled scans | Scanning jobs to the Scanning Controller |
| Scanning Controller | Orchestrates the spawning of vulnerability scanning jobs against the devices | Scanning jobs from the Scanning scheduler. List of supported plugins from the Plugin manager | Plugin-specific scan directives to the job-spawning framework (ArgoCD [114]). Vulnerability scan results to the network of honeypots (WP5). |
| Plugin Manager | Maintains a list of scanning plugins, each one scanning a device or service for a particular grouped set of vulnerabilities | List of plugins from the plugin repository. The plugin repository may be mounted as a volume containing plugin descriptors or a DB containing plugin descriptors | List of available plugins to the scanning controller |
| Scanning Executor | Performs a scan against a device or service, against a particular grouped set of vulnerabilities, as managed by a scan plugin | Plugin-specific scan directives to the job-spawning framework. Information from external vulnerability DBs and associated services | Vulnerability scan results to the scanning controller |

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 [115]), it publishes the device details to a publish subscribe broker (FIWARE Orion Context Broker, see D4.2 [115] and [116] 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 [117] including its IP address. This information gets stored into the internal DB of the Device Manager and gets transferred to the Scanning Scheduler, so that a new scan pipeline may be scheduled.

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** in order to get the list of available and active vulnerability scanning plugins. Each plugin should be considered as a descriptor document presenting the plugin details, accompanied with the plugin source code. Figure 12, below presents a tentative, indicative description of a plugin descriptor document.

```
apiVersion: iot-ngin.eu/v1
kind: VulnerabityScannerPlugin
metadata:
  name: {plugin name}
  description: {plugin description}
  version: {plugin version}
  url: {URL of the plugin homepage, if any.}
  target-device-class: {The device class for which the plugin is targeted to}
plugin:
  service: {Service relevant to vulnerability}
  ports: [{List of ports to scan. May also be a range of ports}]
  timeout: {Time in ms before a connection gets characterized as timed out}
  vulnerability-services: [{List of vuln. search API endpoints to search}]
  vulnerability-services-keys: [{List of API keys to use with the external
                                 vulnerability databases}]
  workspace-type: {Location type of the plugin, may be “volume”, or “remote”}
  workspace-chroot: {The home directory of the plugin}
  flavour: {Programming langugage of the plugin}
  command: {The command to run to execute the scan}
```

Figure 12: Tentative vulnerability scanning plugin descriptor document.

Once having access to the list of available vulnerability scanning plugins, the scanning controller will communicate with **ArgoCD**, instructing it to launch a set of **Scanning Executors**, one for each plugin that is available. Each one of the spawned executors will be also fed with the device details to scan for vulnerabilities and with the corresponding scanning plugin descriptor (possibly also code, depending on the *workspace-type* of the plugin; if the workspace-type of the plugin corresponds to a git repository, then the code will automatically be downloaded from that repo, into *workspace-chroot*). Upon spawning, each scanning executor will start scanning the target device for vulnerabilities considering the descriptor-defined *service*, against the ports defined in the *ports* descriptor field. Finally, depending on the flavour of the plugin (runtime required for the plugin to be executed), a *command* gets executed to start the vulnerability scanning, at service level, i.e. identify whether there are some expected services running behind the designated ports, which service version is running etc. Upon scanning completion, the relevant results 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 which, in turn, interfaces with the network of honeypots, so that a relevant honeypot may be spawned and configured, if such one exists.

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.

More concrete technical details on the Vulnerability Crawler framework will be provided in the continuation of this deliverable, D5.2, due in Q3 2022.

## 7.2 Installation guidelines

The vulnerability crawler will be available for installation using two different modalities for use in Kubernetes environments, namely i) using standard Kubernetes manifests and ii) using the well-known Kubernetes applications package manager Helm [118]. More details will be provided on the component's gitlab repository available at https://gitlab.com/h2020-iot-ngin/enhancing_iot_cybersecurity_and_data_privacy/vulnerability-crawler and in the continuation of this deliverable, D5.2, due in Q3 2022.

# 8 Conclusions

The present report has 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. To this end, the results of the desk analysis of IoT and FL related vulnerabilities and attacks have been briefly presented and considered in identifying cybersecurity needs for the use cases against which the IoT-NGIN framework will be validated in the LLs.

Moreover, IoT-NGIN has addressed the need for strategic management of cyberthreats for corporations adopting or developing smart IoT solutions. To this end, the report presents the cyberthreat modelling methodology proposed by IoT-NGIN and explains how it is implemented in IoT-NGIN. Specifically, IoT-NGIN contributes with four cybersecurity tools to threat modelling and management, namely the GAN based IoT attack dataset generator, MAD, the IoT vulnerabilities crawler and the MTD Network of Honeypots.

The report includes the initial version of the GAN based IoT attack dataset generator, featuring technical design and specifications, as well as initial implementation. Extensive comparative analysis of GAN methods and tools for synthetic dataset generation has presented and considered as the basis for the IoT-NGIN generator. The first version of the tool supports generation of synthetic tabular IDS datasets. Future extension will include generation of datasets incorporating data and model poisoning attacks in FL systems.

Moreover, the initial version of the IoT vulnerabilities crawler has been presented. This includes technical design and specifications, while the initial implementation is part of our future work.

In addition, the next steps of IoT-NGIN activities towards IoT cybersecurity include finalization of the technical design and specifications of all four cybersecurity tools, as well as their implementation. The outcomes of this work will be reported in deliverable document D5.2 "Enhancing IoT Cybersecurity (Update)", which is due in the third quarter of 2022. The stable releases will be made available in the project GitLab page as open source, allowing further development and experimentation by third parties.

# 9 References

[1] IDC, "IoT Growth Demands Rethink of Long-Term Storage Strategies, says IDC," 2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prAP46737220. [Accessed 2021].

[2] Gartner, "Gartner Survey Reveals 47% of Organizations Will Increase Investments in IoT Despite the Impact of COVID-19," 2020. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2020-10-29-gartner-survey-reveals-47-percent-of-organizations-will-increase-investments-in-iot-despite-the-impact-of-covid-19-. [Accessed 2021].

[3] Deloitte, "Intelligent IoT - Bringing the power of AI to the Internet of Things," 2017. [Online]. Available: https://www2.deloitte.com/us/en/insights/focus/signals-for-strategists/intelligent-iot-internet-of-things-artificial-intelligence.html. [Accessed 2021].

[4] EC, "2021 State of the Union Address by President von der Leyen," 2021. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/SPEECH_21_4701. [Accessed 2021].

[5] European Commission, "State of the Union Address 2020," 2020. [Online]. Available: https://ec.europa.eu/info/sites/info/files/soteu_2020_en.pdf.

[6] European Commission, "JOINT COMMUNICATION TO THE EUROPEAN PARLIAMENT AND THE COUNCIL - The EU's Cybersecurity Strategy for the Digital Decade," 2020. [Online]. Available: https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=72164.

[7] European Parliament and the Council of the European Union, "DIRECTIVE (EU) 2016/1148 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 6 July 2016 concerning measures for a high common level of security of network and information systems across the Union," Official Journal of the European Union, 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.194.01.0001.01.ENG&toc=OJ:L:2016:194:TOC.

[8] EC, "Proposal for a Directive of the European Parliament and of the Council on measures for a high common level of cybersecurity across the Union, repealing Directive (EU) 2016/1148," 2020. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52020PC0823&from=EN. [Accessed 2021].

[9] European Parliament and the Council of the European Union, "REGULATION (EU) 2019/881 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 17 April 2019 on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) No," 2019. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32019R0881&from=EN.

[10] EU, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Da," 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EL. [Accessed 2021].

[11] H. Li, K. Ota and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network,* vol. 32, no. 1, pp. 96-101, 2018.

[12] M. Mohammadi, A. Al-Fuqaha, S. Sorour and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys Tutorials,* vol. 20, no. 4, pp. 2923-2960, 2018.

[13] Q. Yang, Y. Liu, T. Chen and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST),* vol. 10, no. 2, 2019.

[14] E. M. H. B. McMahan, D. Ramage, S. Hampson and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017.

[15] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang and P. Warden, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," *CoRR,* 2020.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani and Ch, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *CoRR,* 2019.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard and R. Jozefowicz, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org/.

[18] T. Ryffel, r. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017,* 2018.

[19] S. Caldas, P. Wu, T. Li, J. Konecny´, H. B. McMahan, V. Smith and A. Talwalkar, "LEAF: A benchmark for federated settings," *CoRR,* 2018.

[20] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang and Annavaram, "FedML: A Research Library and Benchmark for Federated Machine Learning," *CoRR,* 2020.

[21] A. K. S. M. S. M. Z. A. T. V. S. T. Li, "Federated Optimization in Heterogeneous Networks," in *Conference on Machine Learning and Systems (MLSys)*, 2020.

[22] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," *CoRR,* vol. abs/1610.05492, 2016.

[23] Z. Chai, H. Fayyaz, Z. Fayyaz, A. Anwar, Y. Zhou, N. Baracaldo, H. Ludwig and Y. Cheng, "Towards Taming the Resource and Data Heterogeneity in Federated Learning," *2019 {USENIX} Conference on Operational Machine Learning (OpML 19),* pp. 19-21, 2019.

[24] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," in *CC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, 2019.

[25] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang and D. I. Kim, "Incentive design for efficient federated learning in mobile networks: A contract theory approach," *arXiv preprint ,* 2019.

[26] M. Hao, H. Li, X. Luo, G. Xu, H. Yang and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence.," *EEE Transactions on Industrial Informatics,* 2019.

[27] R. Ito, M. Tsukad and H. Matsutani, "An On-Device Federated Learning Approach for Cooperative Model Update Between Edge Devices," *IEEE Access,* vol. 9, pp. 92986-92998, 2021.

[28] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, Kudlur, L. M., J., R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and Zhen, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[29] A. Mathur, D. J. Beutel, P. P. B. d. Gusmão, J. Fernandez-Marques, T. Topal, X. Qiu, T. Parcollet, Y. Gao and N. D. Lane, "On-device Federated Learning with Flower," in *On-device Intelligence Workshop at the Fourth Conference on Machine Learning and Systems (MLSys)*, 2021.

[30] V. Smith, C.-K. C. M. Sanjabi and A. S. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[31] M. Khodak, M.-F. F. Balcan and A. S. Talwalkar, "Adaptive Gradient-Based Meta-Learning Methods," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019.

[32] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng and D. Liu, "LoAdaBoost: Loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data," *PLOS ONE,* vol. 15, no. 4, pp. 1-16, 04 2020.

[33] M. Mohri, G. Sivek and A. T. Suresh, "Agnostic Federated Learning," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[34] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang, "Deep Learning with Differential Privacy," in *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[35] F. McSherry and K. Talwar, "Mechanism Design via Differential Privacy," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, 2007.

[36] R. Rivest, L. Adleman and M. Dertouzos, "On Data Banks and Privacy Homomorphism," *Foundations of Secure Computation,* pp. 169-179, 1978.

[37] C. Zhao, S. Zhao, M. Zhao, Z. Chen, C.-z. Gao, H. L and Y.-a. Tan, "Secure Multi-Party Computation: Theory, practice and applications," *Information Sciences,* 2019.

[38] Q. Yang, Y. Liu, T. Chen and Y. Tong, "Federated Machine Learning: Concept and Applications," *ACM Transactions on Intelligent Systems and Technology,* vol. 10, no. 2, 2019.

[39] O. Goldreich and Y. Oren, "Definitions and Properties of Zero-Knowledge Proof Systems," *Journal of Cryptology,* vol. 7, no. 1, 1994.

[40] European Union Agency for Cybersecurity - ENISA, "Vulnerabilities and Exploits - What is a Security Vulnerability?," [Online]. Available: https://www.enisa.europa.eu/topics/csirts-in-europe/glossary/vulnerabilities-and-exploits. [Accessed Oct. 2021].

[41] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor and R. Rogers, "Protection Against Reconstruction and Its Applications in Private Federated Learning," *arXiv preprint,* 2018.

[42] L. Melis, C. Song, E. D. Cristofaro and V. Shmatikov, "Inference Attacks Against Collaborative Learning," *C0RR,* 2018.

[43] L. T. Phong, Y. Aono, T. Hayashi, L. Wang and S. Moriai, "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption," *IEEE Transactions on Information Forensics and Security,* vol. 13, no. 5, pp. 1333-1345, 2018.

[44] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee and D. Papailiopoulos, "Attack of the Tails: Yes, You Really Can Backdoor Federated Learning," *CoRR,* 2020.

[45] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras and T. Goldstein, "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks," in *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 2018.

[46] T. Gu, B. Dolan-Gavitt and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain".

[47] B. Biggio, B. Nelson and P. Laskov, "Poisoning Attacks against Support Vector Machines," 2013.

[48] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin and V. Shmatikov, "How To Backdoor Federated Learning," in *Twenty Third International Conference on Artificial Intelligence and Statistics*, 2020.

[49] A. N. Bhagoji, S. Chakraborty, P. Mittal and S. Calo, "Analyzing Federated Learning through an Adversarial Lens," in *36th International Conference on Machine Learning*, 2019.

[50] A. Bhagoji, S. Chakraborty, P. Mittal and S. Calo, "Model poisoning attacks in federated learning," in *Workshop on Security in Machine Learning (SecML), collocated with the 32nd Conference on Neural Information Processing Systems (NeurIPS'18)*, 2018.

[51] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui and J. Stainer, "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[52] J. Zhang, J. Chen, D. Wu, B. Chen and S. Yu, "Poisoning attack in federated learning using generative adversarial nets," in *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.

[53] R. Shokri, M. Stronati, C. Song and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 3-18.

[54] M. Nasr, R. Shokri and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*, IEEE, 2019, pp. 739-753.

[55] L. Melis, C. Song, E. De Cristofaro and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 691-706.

[56] J. Geiping, H. Bauermeister, H. Dröge and M. Moeller, "Inverting Gradients--How easy is it to break privacy in federated learning?," *arXiv preprint arXiv:2003.14053,* 2020.

[57] S. Chen, M. Kahla, R. Jia and G.-J. Qi, "Knowledge-Enriched Distributional Model Inversion Attacks," *arXiv preprint arXiv:2010.04092,* 2020.

[58] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li and D. Song, "The secret revealer: Generative model-inversion attacks against deep neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 253-261.

[59] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS),* vol. 24, pp. 115-139, 2006.

[60] N. Khamphakdee, N. Benjamas and S. Saiyod, "Improving intrusion detection system based on snort rules for network probe attack detection," in *In 2014 2nd International Conference on Information and Communication Technolog*, Bandung, Indonesia, May 2014.

[61] A. Alharbi, S. Alhaidari and M. Zohdy, "Denial-of-service, probing, user to root (U2R) & remote to user (R2L) attack detection using hidden Markov models," *International Journal of Computer and Information Technology,* 2018.

[62] S. Paliwal and R. Gupta, "Denial-of-service, probing & remote to user (R2L) attack detection using genetic algorithm," *International Journal of Computer Applications,* vol. 60, no. 19, pp. 57-62, 2012.

[63] S. Li, Y. Cheng, Y. Liu, W. Wang and T. Chen, "Abnormal Client Behavior Detection in Federated Learning," *CoRR,* 2019.

[64] S. Li, Y. Cheng, W. Wang, Y. Liu and T. Chen, "Learning to Detect Malicious Clients for Robust Federated Learning," *CoRR,* 2020.

[65] M. Fang, X. Cao, J. Jia and N. Z. Gong, "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning," *CoRR,* 2019.

[66] C. Fung, C. J. M. Yoon and I. Beschastnikh, "Mitigating Sybils in Federated Learning Poisoning," *CoRR,* 2018.

[67] C. Zhao, Y. Wen, S. Li and F. a. M. D. Liu, "FederatedReverse: A Detection and Defense Method Against Backdoor Attacks in Federated Learning," in *2021 ACM Workshop on Information Hiding and Multimedia Security*, 2021.

[68] Y. Khazbak and T. a. C. G. Tan, "MLGuard: Mitigating Poisoning Attacks in Privacy Preserving Distributed Collaborative Learning," in *29th International Conference on Computer Communications and Networks (ICCCN)*, 2020.

[69] L. Zhao, S. Hu, Q. Wang, J. Jiang, S. Chao and X. a. H. P. Luo, "Shielding collaborative learning: Mitigating poisoning attacks through client-side detection," *IEEE Transactions on Dependable and Secure Computing,* 2020.

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

[71] ENISA, "AI cybersecurity challenges," 2020.

[72] I. . Goodfellow, J. . Pouget-Abadie, M. . Mirza, B. . Xu, D. . Warde-Farley, S. . Ozair, A. . Courville and Y. . Bengio, "Generative Adversarial Nets," , 2014. [Online]. Available: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf. [Accessed 27 9 2021].

[73] M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, PMLR, 2017, pp. 214-223.

[74] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. Courville, "Improved training of wasserstein GANs," 2017. [Online]. Available: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf. [Accessed 10 9 2021].

[75] A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434,* 2015.

[76] E. Denton, S. Chintala, A. Szlam and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," *arXiv preprint arXiv:1506.05751,* 2015.

[77] T. Karras, T. Aila, S. Laine and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196,* 2017.

[78] S. Liu, T. Wang, D. Bau, J.-Y. Zhu and A. Torralba, "Diverse image generation via self-conditioned gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14286-14295.

[79] Z. Lin, Y. Shi and Z. Xue, "Idsgan: Generative adversarial networks for attack generation against intrusion detection," *arXiv preprint arXiv:1809.02077,* 2018.

[80] J. Charlier, A. Singh, G. Ormazabal, R. State and H. Schulzrinne, "SynGAN: Towards generating synthetic network attacks using GANs," *arXiv preprint arXiv:1908.09899,* 2019.

[81] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," *arXiv preprint arXiv:1702.05983,* 2017.

[82] L. Yu, W. Zhang, J. Wang and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," *Proceedings of the AAAI conference on artificial intelligence,* vol. 31, no. 1, 2017.

[83] J.-Y. Kim, S.-J. Bu and S.-B. Cho, "Malware detection using deep transferred generative adversarial networks," in *International Conference on Neural Information Processing*, Springer, 2017, pp. 556-564.

[84] Q. Yan, M. Wang, W. Huang, X. Luo and F. R. Yu, "Automatically synthesizing DoS attack traces using generative adversarial networks," *International Journal of Machine Learning and Cybernetics,* vol. 10, no. 12, pp. 3387-3396, 2019.

[85] "KDD Cup 1999 Data," [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[86] J. Charlier, A. Singh, G. Ormazabal, R. State and H. Schulzrinne, "SynGAN: Towards generating synthetic network attacks using GANs," *arXiv preprint arXiv:1908.09899,* 2019.

[87] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSp,* vol. 1, pp. 108-116, 2018.

[88] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park and Y. Kim, "Data synthesis based on generative adversarial networks," *arXiv preprint arXiv:1806.03384,* 2018.

[89] L. Xu, M. Skoularidou, A. C. Infante and K. Veeramachaneni, "Modeling Tabular data using Conditional GAN," 2019. [Online]. Available: https://nips.cc/conferences/2019/acceptedpapersinitial. [Accessed 10 9 2021].

[90] "CopulaGAN Model," [Online]. Available: https://sdv.dev/SDV/user_guides/single_table/copulagan.html.

[91] Z. Zhao, A. Kunar, H. Van der Scheer, R. Birke and L. Y. Chen, "CTAB-GAN: Effective Table Data Synthesizing," *arXiv preprint arXiv:2102.08369,* 2021.

[92] A. Mottini, A. Lheritier and R. Acuna-Agost, "Airline passenger name record generation using generative adversarial networks," *arXiv preprint arXiv:1807.06657,* 2018.

[93] A. Yahi, R. Vanguri, N. Elhadad and N. P. Tatonetti, "Generative adversarial networks for electronic health records: A framework for exploring and evaluating methods for predicting drug-induced laboratory test trajectories," *arXiv preprint arXiv:1712.00164,* 2017.

[94] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart and J. Sun, "Generating multi-label discrete patient records using generative adversarial networks," in *Machine learning for healthcare conference*, PMLR, 2017, pp. 286-305.

[95] L. Xu and K. Veeramachaneni, "Synthesizing Tabular Data using Generative Adversarial Networks.," *arXiv: Learning,* 2018.

[96] M. Svensén and C. M. Bishop, "Pattern recognition and machine learning," Springer, 2007.

[97] A. Borji, "Pros and Cons of GAN Evaluation Measures: New Developments," *arXiv preprint arXiv:2103.09396,* 2021.

[98] L. Theis, A. v. d. Oord and M. Bethge, "A note on the evaluation of generative models," *arXiv preprint arXiv:1511.01844,* 2015.

[99] T. Salimans, G. Ian, W. Zaremba, V. Cheung, A. Radford and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems,* vol. 29, pp. 2234-2242, 2016.

[100] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems,* vol. 30, 2017.

[101] T. Karras, S. Laine and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401-4410.

[102] "Synthetic Data Evaluation-Single Table Metrics," [Online]. Available: https://sdv.dev/SDV/user_guides/evaluation/single_table_metrics.html.

[103] "NSL-KDD Dataset'," [Online]. Available: https://www.unb.ca/cic/datasets/index.html.

[104] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 military communications and information systems conference (MilCIS)*, IEEE, 2015, pp. 1-6.

[105] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM transactions on Information and system security (TiSSEC)*, vol. 3, no. 4, pp. 227-261, 2000.

[106] "SDV - The Synthetic Data Vault," [Online]. Available: https://sdv.dev/SDV/user_guides/benchmarking/synthesizers.html.

[107] "SDGym: Benchmarking framework for Synthetic Data Generators," [Online]. Available: https://github.com/sdv-dev/SDGym.

[108] "Table Evaluator," [Online]. Available: https://baukebrenninkmeijer.github.io/table-evaluator/.

[109] National Institute of Standards and Technology, "Definition of threat," [Online]. Available: https://csrc.nist.gov/glossary/term/threat. [Accessed Oct. 2021].

[110] National Institute of Standards and Technology, "Definition of attack," [Online]. Available: https://csrc.nist.gov/glossary/term/attack. [Accessed Oct. 2021].

[111] C. Richardson, "Microservices Architecture," 2021. [Online]. Available: https://microservices.io/. [Accessed Nov. 2021].

[112] The Linux Foundation, "CNCF Cloud Native Definition v1.0," 11 June 2018. [Online]. Available: https://github.com/cncf/toc/blob/main/DEFINITION.md. [Accessed Nov. 2021].

[113] DataStax, "What is Cloud Native?," [Online]. Available: https://www.datastax.com/cloud-native. [Accessed Nov. 2021].

[114] "ArgoCD home page," The Linux Foundation, 2021. [Online]. Available: https://argoproj.github.io/cd/. [Accessed Nov. 2021].

[115] H2020 IoT-NGIN consortium, "Deliverable D4.2: Enhancing IoT Ambient Intelligence," 2021.

[116] "Welcome to Orion Context Broker," 2021. [Online]. Available: https://fiware-orion.readthedocs.io/en/master/. [Accessed Nov. 2021].

[117] 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].

[118] The Linux Foundation, "Helm - The package manager for Kubernetes," 2021. [Online]. Available: https://helm.sh/. [Accessed Nov. 2021].

[119] S. Caldas, J. Konečny, H. B. McMahan and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," *arXiv preprint arXiv:1812.07210,* 2018.