Automated application privacy compliance checking in distributed Fog environments

Mozhdeh Farhadi

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Mozhdeh Farhadi »

« Automated application privacy compliance checking in distributed Fog environments »

Thèse présentée et soutenue à « Rennes, France », le « 5 novembre 2021 »
Unité de recherche : IRISA (UMR 6074)

Rapporteurs avant soutenance :

Pierre Sens	Professeur des Universités, LIP6 / Sorbonne Université, France
Kévin Huguenin	Professeur, Université de Lausanne, Switzerland

Composition du Jury :

Président :	Guillaume Doyen	Professeur, IMT Atlantique, France
Examinateurs :	Kévin Huguenin	Professeur, Université de Lausanne, Suisse
	Pierre Sens	Professeur des Universités, LIP6 / Sorbonne Université, France
	Claudia Ignat	Chargée de recherche HDR, LORIA–Inria Nancy-Grand Est, France
	Alessandra Toninelli	IT Director, Rejoint, Italy
Dir. de thèse :	Guillaume Pierre	Professeur des Universités, Université de Rennes 1, France
Co-dir. de thèse :	Daniele Miorandi	Chief Executive Officer, U-Hopper, Italy

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765452. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

ACKNOWLEDGEMENT [1]

I would like to thank my supervisors, Daniele Miorandi, and Guillaume Pierre for their inspiration, encouragement, and guidance throughout this research. I appreciate their valuable feedbacks and the time they dedicated to supervising me in this thesis. It was a great honor to work with them.

I would like to thank members of the jury: Guillaume Doyen, Kévin Huguenin, Claudia Ignat, Pierre Sens, and Alessandra Toninelli for accepting to review my thesis and raising inspiring questions during my defense session. Many thanks to the members of the CSID [2]: Laurent Réveillère, and Louis Rilling for their valuable feedbacks and suggestions on this research.

I thank all the ESRs [3] in the Fogguru project: Hamidreza Arkian, Davaadorj Battulga, Dimitrios Giouroukis, Felipe Gutierrez, Paulo Souza, Mulugeta Tamiru, and Li Wu. In particular, I appreciate the wonderful teamwork with Dava, Mulugeta, and Lilly on the LivingFog project in Valencia. Many thanks to Nena Markovic, Julie Montégu, and Gema Roig for their help and support in different phases of the Fogguru project.

I thank all my colleagues and friends at the U-Hopper company in Trento and Myriads team at INRIA Rennes-Bretagne Atlantique.

Special thanks to Jean-Louis Lanet for the great scientific collaboration experience that motivated me to pursue my dream of studying for a Ph.D.

Last but not least, my deepest gratitude to my parents and siblings for their love and support in my life. A special thanks to my husband for always being there for me and encouraging me to follow my dreams.

1. All the names are sorted in alphabetical order.
2. Comité de Suivi Individuel du doctorant.
3. Early stage researcher.

Résumé

Le Fog Computing est une réponse aux limites du modèle centralisé du cloud computing. Le paradigme classique du cloud computing reposait à l'origine sur l'hypothèse que les serveurs cloud sont la principale source de données qui seront consommées à la périphérie du réseau. Cependant, avec la prévalence croissante de l'IoT et des applications mobiles, le rôle des utilisateurs à la périphérie du réseau évolue de purs consommateurs de données à producteurs et consommateurs de données. Ce changement dans le rôle de la couche utilisateur crée de nouvelles exigences qui ne sont pas facilement satisfaites par les plateformes de cloud computing centralisées. Les applications gourmandes en bande passante, les applications sensibles à la latence, les applications avec des modèles de mobilité rapides et les applications qui doivent être déployées dans des zones géographiques avec une connexion peu fiable aux serveurs cloud sont des exemples où le cloud computing présente des limites.

Le volume de données généré par les appareils IoT dans le monde devrait atteindre 79,4 zettaoctets d'ici 2025 [4]. Les applications gourmandes en bande passante consomment de grandes quantités de bande passante réseau. Le volume de données que ces applications envoient aux serveurs cloud longue distance peut facilement encombrer la bande passante du réseau et consommer les ressources du réseau. Une consommation excessive de ressources réseau peut à son tour augmenter les latences et créer des difficultés pour les fournisseurs de services Internet à atteindre la qualité de service promise.

Les applications sensibles à la latence telles que la réalité augmentée et les applications de l'industrie 4.0 nécessitent une latence de bout en bout, y compris un délai de réseau et de traitement inférieur à 10-20 ms. Cependant, la latence entre un utilisateur final et le serveur cloud disponible le plus proche est généralement de l'ordre de 20 à 40 ms sur les réseaux câblés, et jusqu'à 150 ms sur les réseaux mobiles 4G [5]. Cette latence répond aux besoins de certaines applications, mais de nombreuses applications sensibles à la latence ne peuvent la tolérer.

4. IoT Data Volume, `https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-Taille/`.

5. CLAudit project, Planetary-scale cloud latency auditing platform , `http://claudit.feld.cvut.cz/index.php`.

Les applications IoT avec des modèles de mobilité rapide tels que les drones et les véhicules intelligents nécessitent du calcul et du stockage sur une large zone géographique distribuée avec une faible latence. Par exemple, les capteurs se déplaçant avec un véhicule peuvent traverser de vastes zones géographiques en quelques minutes, de sorte que les interactions de ces applications IoT avec les ressources cloud tout en maintenant une faible latence peuvent être difficiles.

Dans certaines zones géographiques, des communications fiables avec le cloud sont soit coûteuses, soit impossibles (par exemple, en raison d'une mauvaise connectivité). Cependant, dans de telles zones, il est souvent nécessaire d'installer des capteurs. Par exemple, les applications agricoles, les applications de détection de fuites de pétrole et de gaz et les applications de stations météorologiques sont des exemples de cas d'utilisation de l'IoT qui doivent généralement déployer des capteurs dans des endroits éloignés des villes et sans communication fiable avec le cloud. Étant donné que ces applications IoT peuvent avoir besoin d'actionner leur environnement sur la base d'une procédure de prise de décision effectuée sur des données agrégées provenant d'autres capteurs, une communication fiable avec la source externe de calcul et de stockage est cruciale. Dans ces cas d'utilisation, le modèle de cloud computing centralisé peut ne pas concrétiser la vision d'un monde IoT universellement connecté.

Le Fog Computing étend le cloud computing avec des ressources de calcul, de stockage et de réseau supplémentaires à proximité des sources de données pour éviter d'envoyer de grandes quantités de données brutes générées par les appareils IoT et les utilisateurs finaux vers le cloud. Les nœuds fog sont ainsi déployés entre la couche utilisateur final et la couche cloud et sont répartis géographiquement à proximité immédiate des utilisateurs et des appareils IoT. Selon la conception de l'application, certaines parties du calcul peuvent être déléguées à la couche fog, ce qui évite d'avoir à envoyer des données brutes vers le cloud. Comme les serveurs de cloud computing traitent les données locales localement, nous nous attendons également à une utilisation plus faible des ressources réseau longue distance ainsi qu'à des temps de réponse plus courts.

Comme toute nouvelle technologie, le fog computing suscite des inquiétudes chez les utilisateurs quant à ses vulnérabilités potentielles en matière de sécurité et de confidentialité. L'emplacement critique des nœuds fog leur fournit un accès privilégié aux données qui sont soit produites par l'utilisateur final et envoyées à la couche cloud, soit renvoyées par la couche cloud à l'utilisateur final. Les nœuds fog peuvent également être plus faciles à pirater que leurs homologues du cloud en raison de leur emplacement en dehors des cen-

tres de données. Par conséquent, une analyse systématique de la sécurité des plates-formes fog est nécessaire.

Dans cette thèse, nous analysons systématiquement la sécurité du fog computing en identifiant ses atouts et ses vulnérabilités possibles sous différents angles : au niveau de l'appareil, au niveau du système et au niveau du service. Notre analyse systématique des actifs et des vulnérabilités des systèmes fog fournit une compréhension claire de la sécurité du système fog pour les analystes de sécurité et les concepteurs de systèmes. Nous mettons également en évidence certaines contre-mesures possibles pour protéger les ressources fog contre les menaces potentielles.

Dans notre étude de la sécurité informatique du brouillard, un atout important identifié sont les données personnelles de l'utilisateur. Comme les nœuds fog sont des ressources proximales pour l'utilisateur final, les applications fog ont accès à des parties importantes des données de leurs utilisateurs. Ces données peuvent contenir des informations personnelles telles que la localisation, le nom et la voix, et sont donc soumises au Règlement Général Européen sur la Protection des Données (RGPD).

Le RGPD oblige les applications à publier une politique de confidentialité décrivant la manière dont elles traitent les données personnelles des utilisateurs. Cependant, la publication d'une politique de confidentialité ne garantit pas la conformité réelle de l'application aux revendications publiées. Par exemple, une étude sur les 20 applications de santé Android les plus populaires sur Google Play a montré que *chaque* application étudiée ne respectait pas au moins une partie des termes de sa politique de confidentialité. L'incident d'analyse vocale Alexa d'Amazon est un autre exemple. Amazon, dans son document de politique de confidentialité, n'indique pas explicitement que les humains écoutent les enregistrements de certaines conversations récupérées par Alexa. Cependant, les personnes qui ont travaillé sur le projet Alexa ont admis qu'en pratique, des humains sont impliqués dans le traitement des enregistrements audio d'Alexa pour l'apprentissage de la reconnaissance vocale et du traitement du langage naturel du système. Les employés d'Amazon avaient accès au prénom du client, au numéro de compte et au numéro de série de l'appareil intelligent pour chaque extrait vocal. Ces incidents mettent en évidence la nécessité de vérifier le comportement des applications concernant les réclamations dans leur politique de confidentialité.

La vérification manuelle de la conformité des applications aux revendications formulées dans leur politique de confidentialité est à la fois sujette aux erreurs et chronophage. Par conséquent, nous étudions la faisabilité de vérifier automatiquement la conformité des ap-

plications à leur politique de confidentialité. Nous soutenons que la vérification automatique de la conformité à la confidentialité est faisable et que la plate-forme d'hébergement cloud ou cloud établit un endroit approprié pour effectuer ces vérifications. Premièrement, les plateformes constituent un tiers auquel font déjà confiance explicitement ou implicitement les développeurs d'applications et les utilisateurs finaux. Deuxièmement, ils ont accès à de nombreuses informations utiles sur les applications, telles que leur utilisation du processeur et le trafic réseau. Nous présentons donc la faisabilité d'une vérification automatisée de la conformité à la confidentialité en montrant une procédure pratique pour identifier un type spécifique de comportement axé sur la confidentialité.

Première contribution : Analyse de sécurité des systèmes de fog computing

Dans cette contribution, nous introduisons une méthodologie pour analyser systématiquement la sécurité des plateformes de fog computing. Actuellement, il n'existe pas d'outil ou de méthodologie spécifique pour analyser la sécurité des systèmes de fog computing de manière globale. Les procédures générales d'évaluation de la sécurité applicables à la plupart des produits informatiques sont chronophages, coûteuses et mal adaptées au contexte du brouillard. Nous appliquons notre méthodologie introduite à un système informatique fog générique, en présentant comment cette approche peut être utilisée à bon escient par les analystes de sécurité et les concepteurs de systèmes.

Sur la base de cette méthodologie, nous analysons un système générique de calcul fog. Notre analyse met en évidence les vulnérabilités possibles qu'un système fog peut présenter. Il indique où le concepteur du système devrait ajouter des contre-mesures pour lutter contre ces vulnérabilités lors de la conception de son système informatique fog.

Deuxième contribution : Vérification automatisée de la conformité à la confidentialité dans les environnements de brouillard

La deuxième contribution répond au besoin de systèmes et de méthodes de vérification automatique de la conformité des applications à la confidentialité. Nous soutenons que les plates-formes de cloud computing ou fog constituent un emplacement approprié pour vérifier la conformité des applications à la confidentialité et proposons un cadre de surveillance non intrusif et indépendant des applications. Dans ce modèle, la plate-forme surveille les signaux d'une application tels que ses caractéristiques de trafic réseau et en

déduit des comportements axés sur la confidentialité à l'aide des signaux de ces applications. Ces comportements inférés sont comparés aux principes extraits de la politique de confidentialité de l'application.

Nous appliquons ce modèle général pour exploiter les signaux de trafic réseau et vérifier la confidentialité du "partage de données avec un tiers" principe. Ce principe, qui définit quels types de données un l'application est autorisée ou non à partager avec un tiers spécifique parties, est au cœur de la réglementation RGPD européenne.

Nous présentons une procédure basée sur des techniques d'apprentissage automatique pour identifier le type de données partagées par les applications avec des tiers externes même si l'application utilise des communications cryptées. Nous démontrons que notre approche est capable de : (1) détecter correctement l'existence du comportement de partage de données dans une application ; et (2) identifier les types de données partagées en analysant le trafic réseau sortant des applications. Notre système repose uniquement sur les en-têtes des paquets réseau capturés, et fonctionne ainsi également en présence du trafic crypté. Dans nos expériences, le classificateur est capable de caractériser le type de trafic des applications déjà connues avec 86% précision en termes de score F1, et d'attribuer le trafic réseau de applications inconnues dans la classe correcte avec une précision de 84%.

Abstract

Fog computing is a response to the limitations of the centralized model of cloud computing. The classic cloud computing paradigm was originally based on the assumption that the cloud servers are the primary source of data that will be consumed at the edge of the network. However, with the increasing prevalence of IoT and mobile applications, the role of users at the edge of the network is changing from pure consumers of data to producers and consumers of data. This change in the role of the user layer creates new requirements that are not easily met by centralized cloud computing platforms. Bandwidth-intensive applications, latency-sensitive applications, applications with rapid mobility patterns, and applications that need to be deployed in geographic areas with unreliable connection to cloud servers are examples where cloud computing presents limitations.

The volume of data generated by IoT devices worldwide is forecast to reach 79.4 zettabytes by 2025 [6]. Bandwidth-intensive applications, consume large amounts of network bandwidth. The volume of data that these applications are sending to long-distance cloud servers may easily congest the network bandwidth and consume the network resources. An excessive consumption of network resources may in turn increase latencies and create difficulties for the Internet service providers to reach their promised quality of service.

Latency-sensitive applications such as augmented reality and industry 4.0 applications require end-to-end latency including network and processing delay under 10-20 ms. However, the latency between an end user and the closest available cloud server is typically in the range of 20-40 ms over wired networks, and up to 150 ms over 4G mobile networks [7]. This latency addresses the needs of some applications, but numerous latency-sensitive applications cannot tolerate it.

IoT applications with rapid mobility patterns such as drones and smart vehicles require compute and storage over a broad distributed geographical area with low latency. For example, sensors moving with a vehicle may traverse large geographical areas in a few

6. IoT Data Volume, `https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/`.

7. CLAudit project, Planetary-scale cloud latency auditing platform, `http://claudit.feld.cvut.cz/index.php`.

minutes, so the interactions of these IoT applications with cloud resources while keeping the latency low may be challenging.

In some geographic areas, reliable communications with the cloud are either expensive or impossible (e.g., due to poor connectivity). However, in such areas there is often a need to install sensors. For instance, agriculture applications, oil and gas leak detecting applications, and weather station applications are examples of IoT use cases which typically need to deploy sensors in locations far from the cities and without a reliable communication to the cloud. As these IoT applications may need to actuate their environment based on a decision-making procedure performed on aggregated data from other sensors, a reliable communication to the external source of compute and storage is crucial. In these use cases, the centralized cloud computing model may fail to realize the vision of a universally connected IoT world.

Fog computing extends cloud computing with additional computation, storage and network resources close to the sources of data to avoid sending large amounts of raw data generated by IoT devices and the end users toward the cloud. Fog nodes are thus deployed between the end-user layer and the cloud layer and are geographically distributed in close proximity of the users and the IoT devices. Depending on the design of the application, some parts of the computation may be delegated to the fog layer, which prevents one from having to send raw data to the cloud. As fog computing servers process local data locally, we also expect a lower usage of long-distance network resources as well as lower response times.

As any new technology, fog computing raises users' concerns about its potential security and privacy vulnerabilities. The critical location of fog nodes provides them with privileged access to data that are either produced by the end-user and sent to the cloud layer, or sent by the cloud layer back to the end user. Fog nodes may also be easier to hack than their cloud counterparts because of their location out of the data centers. Therefore, a systematic analysis of the security of fog platforms is necessary.

In this thesis, we systematically analyze the security of fog computing by identifying its assets and possible vulnerabilities from different perspectives: device level, system level, and service level. Our systematic analysis of assets and vulnerabilities of fog systems provides a clear understanding about the security of the fog system for security analysts and system designers. We also highlight some possible countermeasures to protect the fog assets from potential threats.

In our study of fog computing security, one important identified asset is the user's personal data. As fog nodes are proximal resources for the end-user, fog applications have access to significant parts of their users' data. These data may contain personal information such as location, name and voice, and therefore be subject to the European General Data Protection Regulation (GDPR).

The GDPR requires applications to publish a privacy policy describing how they handle users' personal data. However, publishing a privacy policy does not guarantee the actual compliance of the application to the published claims. For instance, a study on the top 20 most popular Android health applications on Google Play showed that *every* studied application failed to comply with at least part of its privacy policy terms. Amazon's Alexa voice analysis incident is another example. Amazon, in its privacy policy document, does not explicitly state humans are listening to recordings of some conversations fetched by Alexa. However, people who have worked on the Alexa project admitted that in practice humans are involved in processing the audio recordings of Alexa for the training of the speech recognition and natural language processing of the system. Amazon employees had access to the customer's first name, account number and the serial number of the smart device for each voice snippet. These incidents highlights the need to verify applications behavior regarding the claims in their privacy policy.

Checking the compliance of applications to the claims made in their privacy policy manually is both error-prone and time-consuming. Therefore, we study the feasibility of verifying the compliance of applications to their privacy policy automatically. We argue that automatic privacy compliance checking is feasible, and that the fog or cloud hosting platform establishes a suitable place to perform these checks. First, the platforms constitute a third party that is already trusted explicitly or implicitly by the application developers and the end-users. Second, they have access to many useful informations about the applications, such as their CPU usage and network traffic. We therefore present the feasibility of automated privacy compliance checking by showing a practical procedure to identify one specific type of privacy-oriented behavior.

First contribution: Security analysis of fog computing systems

In this contribution, we introduce a methodology for analyzing the security of fog computing platforms systematically. Currently, there is no specific tool or methodology for analyzing the security of fog computing systems in a comprehensive way. General

security evaluation procedures applicable for most information technology products are time-consuming, costly, and badly suited to the fog context. We apply our introduced methodology to a generic fog computing system, presenting how this approach can be purposefully used by security analysts and system designers.

Based on this methodology, we analyze a generic fog computing system. Our analysis brings to the fore the possible vulnerabilities that a fog system may have. It points out where the system designer should add countermeasures to tackle such vulnerabilities while designing his fog computing system.

Second contribution: Automated privacy compliance checking in fog environments

The second contribution addresses the need for systems and methods for automatic privacy compliance checking of applications. We argue that cloud or fog computing platforms constitute a proper location for verifying the privacy compliance of applications, and propose an unintrusive and application-agnostic monitoring framework. In this model, the platform monitors an application's signals such as its network traffic characteristics and infers privacy-oriented behaviors using these application's signals. These inferred behaviors are compared with the principles extracted from the application's privacy policy.

We apply this general model to exploit network traffic signals and verify the "data sharing with third party" privacy principle. This principle, which defines which data types an application is allowed or disallowed to share with specific third parties, is at the heart of the European GDPR regulation.

We present a procedure based on machine-learning techniques to identify the type of data being shared by applications with external third parties even if the application uses encrypted communications. We demonstrate that our approach is able to: (1) correctly detect the existence of the sharing data behavior in an application; and (2) identify shared data types by analyzing the applications' outgoing network traffic. Our system relies only on the headers of captured network packets, and thereby works also in the presence of encrypted traffic. In our experiments the classifier is able to characterize the traffic type of already-known applications with 86% accuracy in terms of F1 score, and to attribute network traffic of unknown applications to the correct class with 84% accuracy.

Table of Contents

Chapter 1

Introduction

Cloud computing has expanded the computer science horizon during the past decade by providing its users with on-demand, scalable, resilient and economic computing resources. Cloud users can easily scale up and down their resources according to their business needs without concerns about the maintenance of physical resources. They do not have to know or care about the exact physical location of these resources. Cloud providers allocate resources to customers from a large pool, allowing them to benefit from economies of scale. Moreover, cloud users pay for only what they use and, if they stop using resources, they simply stop paying.

Cloud computing platforms are usually referred to as a large pool of computing, storage and networking resources which can be accessed through an abstract interface [1]. The data centers that host the cloud resources are typically connected to each other with dedicated network connections to optimize latency and availability. For instance, Google, a popular cloud provider, uses private submarine cables to connect its data centers across the world; a submarine cable connects Chile to Los Angeles, a consortium submarine cable connecting U.S. to Denmark and Ireland [2]. However, the cloud users usually use commodity Wide-Area Network (WAN) connections to access cloud services. Figure 1.1 shows the 24 current Google's cloud regions in the world and its future planned regions [2].

It is forecast that the number of IoT devices will be 14.7 billions by 2023 [3]. In the most common organization, IoT devices generate data and send these data to the cloud servers for processing, aggregation and storage. This model places cloud servers as the external centralized resource for IoT devices. However, with the growth of IoT applications and emergence of new use cases, this centralized model is becoming insufficient in numerous use cases [4].

Latency-sensitive applications such as augmented reality and industry 4.0 applications require end-to-end latency including network and processing delay under 10-20 ms [5]. However, the latency between an end-user and the closest available cloud server is typically in the range of 20-40 ms over wired networks, and up to 150 ms over 4G mobile

Figure 1.1 – The Google cloud regions in the world. Blue dots show current regions. White dots show future regions [2].

networks [5]. This latency addresses the needs of some applications, but numerous latency-sensitive applications cannot tolerate it [6].

In some geographic areas, the communications with the cloud are expensive or unreliable (e.g., due to poor connectivity). However, there is often a need to install sensors in such areas. For instance, agriculture applications, oil and gas leak detecting applications, weather station applications are examples of IoT use cases which typically need to deploy sensors in locations far from the cities and without a reliable communication to the cloud. As these IoT applications may need to actuate their environment based on a decision-making procedure performed on aggregated data from other sensors, a reliable communication to the external source of compute and storage is indispensable. In these use cases, the centralized cloud computing model falls short of the vision of a universally connected IoT world.

IoT applications with rapid mobility patterns such as smart vehicles require compute and storage over a broad distributed geographical area with low latency [7]. For example, sensors moving with a vehicle may traverse large geographical areas in a few minutes, so the interactions of these IoT applications with cloud resources while keeping the latency low may be challenging [7].

The huge amount of data that devices are sending to long-distance cloud servers may easily congest the network bandwidth and consume the network resources. The volume of data that IoT devices generate is forecast to reach 79.4 zettabytes by 2025 [8]. An

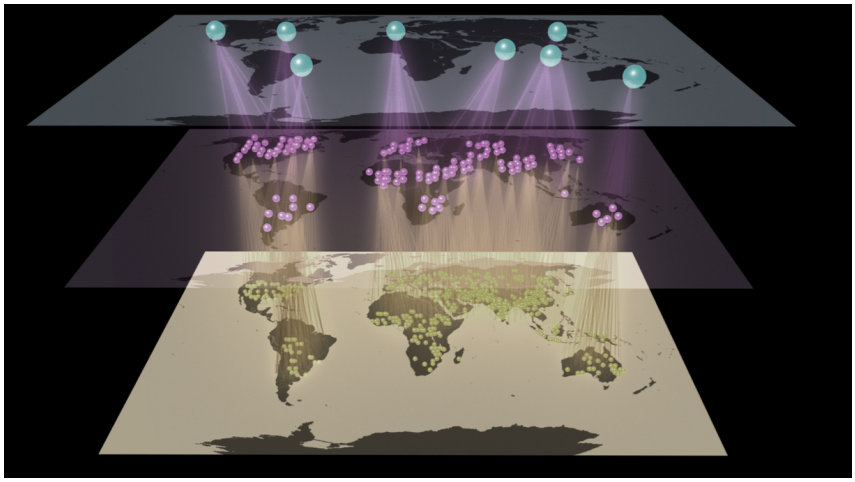

Figure 1.2 – Our vision for the fog computing platforms in the future. Blue dots illustrate sample locations of cloud data centers, purple dots represent fog nodes and yellow dots illustrate user devices that make use of the fog as a middle layer to the centralized server [4].

excessive consumption of network resources may in turn increase latencies and create difficulties for the Internet service providers to reach their promised quality of service.

An evident solution to the mentioned problems is to bring computing servers closer to the user and the IoT devices and avoid sending raw data to a far away cloud server. This is where fog computing appears. Fog computing is a technology to enrich cloud computing with additional compute, storage and networking resources in close proximity to the end-user devices [9]. Fog computing therefore acts as a middle layer, sitting between the IoT devices and the cloud. Depending on the application, some parts of the computation may be delegated to the fog layer, which prevents one from having to send raw data to the cloud. As fog computing servers process local data locally, we also expect a lower usage of long-distance network resources as well as lower response times.

Figure 1.2 illustrates a possible vision for fog platforms in the future. The fog layer, which is specified with purple dots, fills the gap of resources between end-users and cloud servers. The number of fog servers is envisioned to be high due to the necessity to cover a broad geographic area. We also expect fog servers to be smaller than cloud servers. For example, a set of ten connected Raspberry Pis constitutes a powerful fog cluster with an affordable price and an easier form factor to install in random locations [10].

One major obstacle to the prevalence of any new technology, including fog computing, is the users' concern about potential security and privacy issues. According to McKinsey,

security and data privacy represent the biggest threat for the further growth of IoT systems and the adoption of IoT-enabled applications [11]. Fog computing, due to its critical location between the IoT and the cloud layer, has access to the application data, whether in the form of sensor data being sent to the cloud or control data sent back to the IoT devices. These data often carry personal information about the user's activities, location, and so on. It is forecast that by 2025 about 75% of all enterprise-generated data will need analysis and action in the fog layer [12]. Therefore, a precise and systematic analysis of the security of fog platforms is necessary.

Analyzing the security of fog system in a systematic way brings to the fore the vulnerabilities a fog system may have. There are several researches that explore security of fog platforms, but they do not follow a systematic approach. Fog computing lies at the cross-roads of some well-established research topics (distributed systems, IoT and cloud computing) [13]. Although fog computing naturally inherits many of the security issues and possible solutions from cloud computing, it also has a number of specificities due to its decentralized architecture and its expected usage. A systematic analysis of a fog system would arguably help the designers of a fog architecture to identify the vulnerabilities and tackle the potential threats by placing appropriate countermeasures.

The first contribution of this thesis is a new methodology for security analysis of fog computing systems in a systematic way. We study the security of the fog platform from multiple perspectives: device level, system level and service level. For each perspective, we discuss the possible vulnerabilities that the system may have and highlight some possible solutions.

One of the important identified assets in our study of fog platform's security is the user's personal data. Because of fog nodes' proximate location to the user, fog applications have access to significant parts of their users' private data. A fog user's data may contain personal information and therefore be subject to the European General Data Protection Regulation (GDPR) [14]. According to the GDPR, personal data is defined as any information that relates to an identified or identifiable living individual.

The GDPR enforces all companies that handle personal data located in the EU to publish a clear and concise Privacy Policy that informs the users how their data will be collected, processed and controlled [15]. Similar requirements about exposing a privacy policy are enforced in other places of the world. For example UK GDPR [16] and The California Consumer Privacy Act (CCPA) [17] enforce companies that hold personal data to publish privacy notices.

In the privacy policy documents, applications usually claim they respect users' privacy. However, some reports have been published recently, which show that applications do not necessarily act as they claim in their privacy policy. An example is Amazon's Alexa voice analysis incident [18]. Amazon, in its privacy policy document, doesn't explicitly say humans are listening to recordings of some conversations picked up by Alexa. However, seven people who have worked on Alexa project admit that in practice humans are involved in processing the audio recordings of Alexa for the training of the speech recognition and natural language processing of the system. According to Bloomberg [19], Amazon employees had access to the customer's first name, account number and the serial number of the smart device for each voice clip [20]. As another example, a recent study of the top 20 most popular Android health applications on Google Play showed that *every studied application* failed to comply with at least part of its own privacy policy terms [21]. As a consequence, the compliance of applications to their privacy policy terms should not be taken for granted but rather be the subject of "Trust, But Verify" strategies [22].

The said incidents bring to the fore the need for a system to verify applications behavior regarding their claims in their privacy policy. However, manually checking whether applications actually respect the claims made in their privacy policy is both error-prone and time-consuming.

We argue that fog platforms constitute a strategic location where the actual behavior of applications may be automatically monitored and verified against their published privacy policies. First, the hosting platform constitutes a third party which is already trusted explicitly or implicitly by the application developers and the end-users. Second, it has access to many useful informations about the applications, such as their network traffic and CPU usage. Last but not least, in the case of Fog computing platforms, their location in close proximity to the end-users gives them an interesting opportunity to reduce potential data leaks between the end-user devices and the application running in the Fog.

Our second contribution is therefore a system model for privacy compliance checking of applications using fog environments. We emphasize the need for unintrusive and application-agnostic privacy compliance checking systems and outline a research roadmap towards the development of such systems.

1.1 Contributions

The main contributions of this thesis are as follows:

1. **A systematic approach in the security of fog computing systems**
   This work presents an analytical approach to the security analysis of fog computing systems. We utilize the Common Criteria methodology [23], and tailor it for analyzing the security of fog computing systems. In our methodology, we identify assets of fog computing, and cluster them into device level, system level and service level. For each perspective, we point out possible vulnerabilities that the system may have and discuss possible solutions. In order to shape our analysis, we utilize attack-defense tree [24] and represent vulnerabilities with possible countermeasures.

   We do believe that this approach can provide a valuable tool for both researchers and practitioners. We analyze a generic fog computing system based on our introduced methodology. Our analysis brings to the fore the possible vulnerabilities that a fog system may have. It points out where the system designer should add countermeasures to tackle such vulnerabilities while designing his fog computing system.

2. **Automated application privacy compliance checking in fog environment**
   This work is a first step towards automatic privacy compliance checking of fog applications. An automated application privacy compliance checking system essentially needs to derive a set of privacy principles from the application's privacy policy, and to watch the application as a set of privacy-oriented behaviors. Later, these two sets may be compared to report possible differences between the claims made in the privacy policy and the observed application behavior. This comparison may be performed off-line or on-line. In both cases, the system needs to monitor application signals (e.g., applications' CPU usage profile, network traffic) and based on the application signals deduce the application behavior. If a non-compliant behavior is detected, the application signals can be saved to be referred to as an evidence of the application's non privacy-compliant behavior. Figure 1.3 represents a high level design of our model for automatic privacy compliance checking.

   As a proof of concept to our proposed model, we present a system to automatically identify the behavior of sharing data with third parties which is an important privacy-oriented behavior in most of privacy policies. From the technical perspective, our system builds upon the usage of machine learning methods and tools, suitably applied to features extracted from the network traffic captured in the fog node. We demonstrate that our approach is able to: (1) correctly detect the ex-

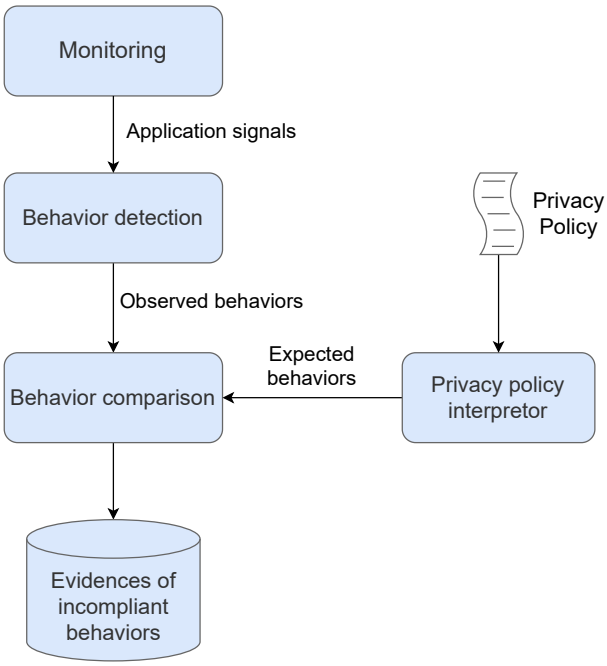

Figure 1.3 – The high level design of automatic privacy compliance checking system.

istence of the sharing data behavior in an application; and (2) to identify shared data types by analyzing the applications' outgoing network traffic. Our system relies only on the headers of captured network packets, and thereby works also in the presence of encrypted traffic. In our experiments the classifier is able to characterize the traffic type of already-known applications with 86% accuracy in terms of F1 score, and to attribute network traffic of unknown applications to the correct class with 84% accuracy.

1.2 Published papers

Journal article(s)

— "A systematic approach toward security in fog computing: Assets, vulnerabilities, possible countermeasures", M Farhadi, JL Lanet, G Pierre, D Miorandi Wiley, Software: Practice and Experience, Special Issue, 50 (6): New Generation cloud Computing.

Conference paper(s)

— "Towards automated privacy compliance checking of applications in Cloud and Fog environments", M Farhadi, G Pierre, D Miorandi, in Proc. IEEE Conference on Future Internet of Things and Cloud, 2021.
— "Blockchain enabled fog structure to provide data security in IoT applications", M Farhadi, D Miorandi, G Pierre, in Doctoral symposium of the ACM/IFIP Middleware conference, 2018 [1].
— "An experimental evaluation of the scalability of permissioned blockchains", S Tavonatti, D Battulga, M Farhadi, C Caprini, D Miorandi, in Proc. IEEE Conference on Future Internet of Things and Cloud, 2021[1].

1.3 Organization of the thesis

This thesis is organized in seven chapters.

Chapter 2 presents the conceptual and technical background of the thesis. First, we present an overview of cloud computing and some of its limitations. We then define fog computing as an emerging technology to addresses the limitations of cloud computing technology. We detail Kubernetes [25], a vendor independent orchestrator which is the basis of our prototypical implementation and further discuss its networking subsystems. We present Common Criteria and Attack-tree as security analysis techniques. Finally, we present a brief overview of machine learning and especially classification techniques in supervised learning.

Chapter 3 describes the state-of-the-art on security and data privacy in fog computing. We start the discussion about current security challenges in the fog domain and then we narrow down our discussion to personal data and methods to protect personal data in fog platforms. We present current challenges in providing automatic privacy compliance checking for the fog applications.

Chapter 4 presents our first contribution which is a methodology for systematic analysis of the security of fog platforms. Although fog computing naturally inherits most of the security issues and possible solution from cloud computing, it also has a number of specificities due to its decentralized architecture or its expected usage. On the other hand, it also creates potential opportunities to address specific types of privacy and/or security issues. In this chapter we discuss the specific properties of fog computing security and privacy and then introduce our methodology to cover the analysis of these properties.

1. This publication will not be discussed in the thesis

Additionally, we apply our methodology on a generic fog platform to showcase how our methodology can be exploited.

Chapter 5 discusses the automatic privacy compliance checking in fog platforms. We present the system model for privacy compliance checking. We also discuss about unintrusive monitoring of applications in the fog platforms and discuss about its challenges. We present implementation of an important privacy principle which is sharing data with third parties. We describe how we used the monitored properties of our fog application to understand its privacy-oriented behavior. We demonstrate that machine learning models can solve the mentioned privacy principles in the privacy compliance checking. We describe our created dataset, the experiments to find the best model and finally the accuracy of our model to decide about an specific behavior of an application.

Finally, Chapter 7 presents the conclusions of the thesis and discusses directions for future work.

Chapter 2

Background

This chapter provides the technical background for the thesis. First, we present cloud computing and its limitations which caused the advent of fog computing. Second, we describe fog computing, its architecture and example use cases. We then provide an introduction to Kubernetes, a vendor-neutral container orchestration platform for fog/cloud environments in which our solution is prototyped. Finally, we provide a general overview of machine learning and its usage in classification of network traffics.

2.1 Cloud computing

The idea of providing centralized computing services accessible over a network dates backs to the 1960's and the mainframe timesharing technology. In those times, computers were extremely expensive and the idea of timesharing dramatically decreased the cost of accessing computing resources for individuals and organizations. In the 1980's personal computers arrived, promising to liberate programs and data from mainframe timesharing centers [26]. However, enterprises continued to use centralized computing services in the form of client-server architecture.

The centralized architecture was cost-effective for enterprises because they did not have to pay for multiple software licenses and the centralized architecture provided easy access to aggregated data. In the 1990's, a typical enterprise computing infrastructure consisted of powerful and very expensive servers which could host up to 20-30 enterprise applications [27]. This market was dominated by only a few hardware vendors, such as IBM [28], Sun [29], HP [30], and Dec [31], whose servers were expensive to purchase and maintain, and required considerable time for installation and maintenance. Most of the servers hosted multiple applications within the same operating system without providing physical or virtual isolation. Additionally, as it was difficult to move and rebalance applications across servers quickly, server resources were not utilized effectively. Moreover, the

provided services were not resilient to server outages. Any problem in the server could last several hours until a vendor representative delivered proprietary replacement parts [27].

In 2006, cloud computing introduced major improvements in resource utilization, isolation between applications, and service resilience [32]. Cloud computing is the delivery of virtualized computing services over the Internet to offer faster innovation, flexible resources, and economies of scale [33]. Currently, 94% of enterprises are using at least one cloud service [34].

The main enabling technology for cloud computing is virtualization which refers to the abstraction of computer resources [35]. In virtualization, a single physical resource may serve as multiple virtual resources or vice versa (multiple physical resources may function as a single virtual resource) [36]. The virtualization technology improves resource utilization and energy efficiency, reduces server maintenance overhead and provides fast disaster recovery and high availability [37]. Cloud computing isolates software from hardware and so provides a mechanism to reallocate resources across servers based on computational demands.

Currently, the most two popular virtualization technologies are hypervisor-based and container-based technologies [38]. The main difference between these two types of virtualization is that Hypervisor-based virtualization provides strong isolation of a complete operating system, whereas container-based virtualization uses the same Operating System (OS) for all the virtualized resources and it isolates processes from each other at lower resource costs [39], [40]. The use of Docker as a container-based virtualization tool is nowadays very popular due to its efficient use of resources and ease of use [39]. We will come back to Docker later in this thesis.

In cloud computing, users can quickly scale up and down their provisioned resources. Cloud users pay as they use the services and, by this model, the capital expenses (CapEx) of buying hardware and software, setting up and running data centers is replaced by operational expenses (OpEx). Cloud computing improves business reliability by simplifying data backup and disaster recovery, and hence it facilitates business continuity [33].

2.1.1 Cloud computing architecture

Cloud computing platforms are usually organized as a large pool of computing, storage and networking resources which can be accessed through an abstract interface [1]. A cloud provider's virtualized resources are exposed through an abstraction layer which hides the complexity of managing resources in the cloud. For example, AWS [41] from Amazon or

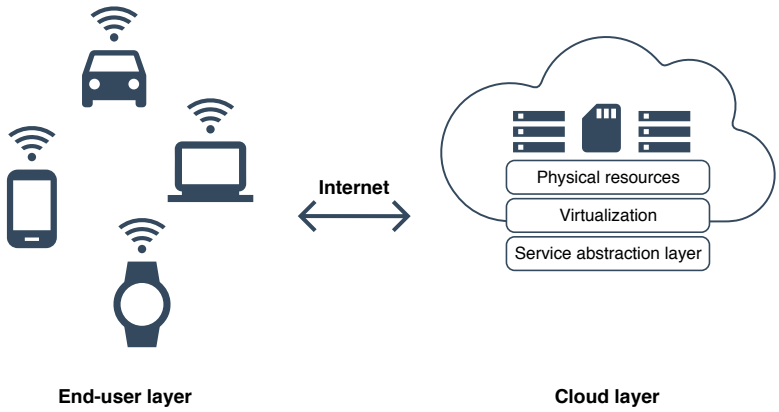

Figure 2.1 – The high-level architecture of cloud computing systems.

GCP [42] from Google provide a set of services for their cloud users to make configuration and management tasks more convenient.

As illustrated in Figure 2.1, the cloud computing architecture has two main parts: (1) the end-user layer, which is the layer containing the client devices that use the cloud services and resources; and (2) the cloud layer which comprises the resources hosted in the data centers [43]. The end-user layer and cloud layer are connected to each other through the Internet. The data centers that host the cloud resources are typically connected to each other with dedicated network connections to optimize latency and availability.

2.1.2 Cloud computing limitations

Cloud computing, a revolutionary technology in our era, unfortunately also suffers from a number of limitations. Cloud data centers are centralized in a small number of data centers and, thus, typically located far from the end-users, resulting in high latencies [44]. Latency-sensitive applications such as augmented reality and industry 4.0 applications require end-to-end latency including network and processing delays under 10-20 ms [5]. However, the network latency alone between an end-user and the closest available cloud server is typically in the range of 20-40 ms over wired networks, and up to 150 ms over 4G mobile networks [5]. This latency addresses the needs of some applications, but numerous latency-sensitive applications cannot tolerate it [6].

In some geographic areas, the communications with the cloud may be expensive or unreliable (e.g., due to poor connectivity). However, there is often a need to install sensors

in such areas. For instance, agriculture applications, oil and gas leak detecting applications, weather station applications are examples of IoT use cases which typically need to deploy sensors in locations far from the cities and without a reliable communication to the cloud [7]. As these IoT applications may need to actuate their environment based on a decision-making procedure performed on aggregated data from other sensors, a reliable communication to the external source of compute and storage is indispensable. In these use cases, the centralized cloud computing model falls short of the vision of a universally connected IoT world.

IoT applications with rapid mobility patterns such as smart vehicles require compute and storage over a broad distributed geographical area with low latency [7]. For example, sensors moving with a vehicle may traverse large geographical areas in a few minutes, so ensuring the interactions of these IoT applications with cloud resources while keeping the latency low may be challenging [7].

The huge amount of data that devices are sending to far-away cloud servers may easily congest the network bandwidth and consume the network resources. The volume of data that IoT devices generate is forecast to reach 79.4 zettabytes by 2025 [8]. An excessive consumption of network resources may in turn increase latencies and create difficulties for the Internet service providers to reach their promised quality of service.

As a response to the above limitations of cloud computing, the fog computing appeared.

2.2 Fog computing

The classic centralized computing paradigm was based on the assumption that the cloud servers are the primary source of data that will be consumed at the edge of the network. However, with the increasing prevalence of IoT and mobile applications, the role of users at the edge of the network is changing from pure consumers of data to producers and consumers of data [44]. This change in the role of the edge layer creates new requirements which are not easily met by centralized cloud computing platforms.

Fog computing technology extends cloud computing services and resources at the edge of the network [9]. It provides computation, storage and network resources in close proximity of data sources using distributed, heterogeneous and resource-constrained devices called fog nodes [45].

The IEEE defines fog computing as "a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the cloud-to-things continuum" [46]. "Fog computing" is largely interchangeable with other similar computing paradigms such as edge computing and mist computing. In this thesis, we utilize the IEEE definition as our reference and we consider any technology which conform to this definition [46] as fog computing. A complete survey of all similar technologies to fog computing can be found in [47].

Fog computing performs data processing tasks at the edge of the network and hence decreases the need of sending large amounts of raw data to the cloud. This reduces latency, network congestion, and the usage of network resources.

Fog nodes are geographically distributed in close proximity of the users and IoT devices in some areas with unreliable connectivity to the cloud. Hence, the fog locally provides the resources that these devices need. Additionally, for IoT applications which use devices with rapid mobility patterns, such as drones, the distributed resources of the fog in broad areas satisfies their need of nearly computing and storage resources [7].

Note that processing of the data near the data sources typically provides the fog nodes with privileged access to the users' personal data. The data that a fog node deals with is often more security- and privacy-critical than the data cloud servers receive [13]. Moreover, the fog nodes have access to the data source location, which is another type of personal data. These considerations highlight the needs of scrutinizing the security and privacy of fog computing platforms, which is the main topic of this thesis.

2.2.1 Fog computing architecture

As illustrated in Figure 2.2, a fog computing architecture typically consists of three layers: (1) End-user layer (also called IoT layer); (2) Fog layer; (3) Cloud layer.

End-user layer: This layer consists of IoT devices, computers, mobile phone, etc. that are in direct contact with the user or the target environment (in the case of IoT applications). The end-user layer gathers data from the user or the environment and sends it to the other layers for further processing. This layer may also receive data from the other layers in the architecture in the form of data or commands to actuate their environment. The IoT devices usually use commodity access network such as cellular network, WiFi and LoRa to connect to the fog and cloud layers [43]. This layer is referred to with different terms such as end-user layer, edge layer, and IoT layer. However in this thesis we use the end-user layer.

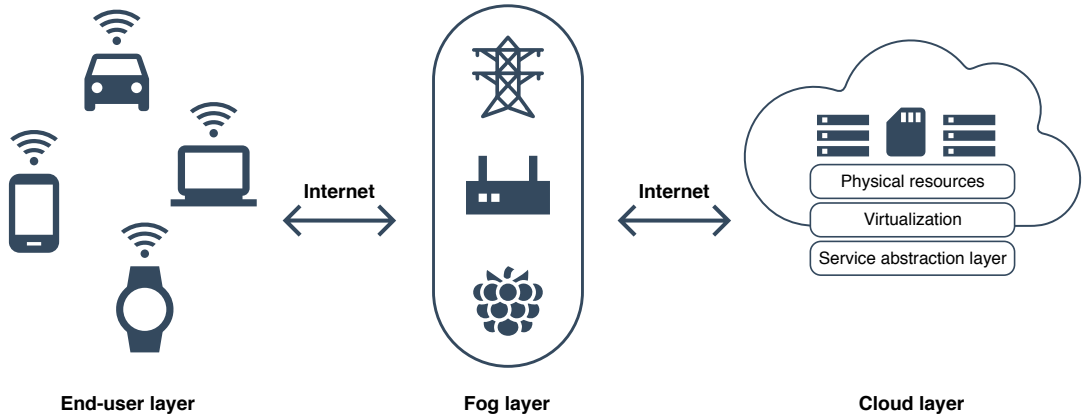

Figure 2.2 – The high-level architecture of a fog computing system.

Fog layer: The fog computing layer consists of heterogeneous computation, networking and storage resources that are deployed in close proximity of the IoT layer. This layer comprises a set of fog nodes distributed over a geographic area. The fog layer may sometimes be organized along a hierarchical layout, with also possible communication between the nodes in a distributed manner. The network that these fog nodes use to communicate with each other may be different than the network used by cloud servers for their internal communication. For example, in some geographic locations, the only available network for communication between fog nodes may be wireless.

Different types of fog nodes have been proposed by researchers. A medium sized server, a router in the network, a cellular base station or a simple Raspberry Pi can be used as a fog node [48].

As the fog nodes are geographically distributed elements, they also may belong to different organizations. Fog infrastructure providers may be Internet Service Providers (ISP), Mobile Network Operators (MNO), cloud infrastructure providers, smart cities or any organization that owns a set of Internet-connected gateways with computing capabilities. Fog service providers can be one of the aforementioned actors, or even end-users who own a private cloud and are interested in exploiting their spare resources [49]. Thus, the ownership of the fog nodes can be different than in the cloud model.

Cloud layer: The uppermost layer is the cloud layer. This layer is composed of a small number of powerful data centers that are connected to each other and to the rest of the Internet with ultra high-speed networks [1]. It contains powerful computing, storage and networking capacity to support applications which need extensive computational analysis and back-end storage.

2.2.2 Fog computing applications

Thanks to different fog computing use cases surveyed in [6], [46], [50], we argue that the main purposes of using fog computing technology stems from the applications with at least one of the following characteristics:

— Latency-critical applications: Latency-sensitive applications such as augmented reality and industry 4.0 applications require end-to-end latency including network and processing delay under 10-20 ms [5]. However, the network latency alone between an end-user and the closest available cloud server is typically in the range of 20-40 ms over wired networks, and up to 150 ms over 4G mobile networks [5]. This latency addresses the needs of some applications, but numerous latency-sensitive applications cannot tolerate it [6].

— Data-intensive applications: Applications which produce massive volumes of data, such as live video broadcasting and surveillance applications, may find it impractical to send their data to faraway cloud servers for analysis. These applications can benefit from fog computing, which processes the generated data in the device-to-cloud continuum and therefore optimizes the network bandwidth usage.

— Geographically distributed applications: Applications which are used in geographic areas with the lack of sufficient and reliable connection to the cloud, such as gas/oil leak detection and agriculture applications may strongly benefit from fog computing. In [50], the authors provide a review of real-world fog applications which are deployed in geographic areas which lack sufficient and reliable bandwidth to the cloud.

Similar to cloud technology, fog can benefit from containerization of applications and services [39]. Containerization is nowadays a very effective, industrially-mature technology that facilitates the deployment of the application regardless of the target environment [39]. It encapsulates an application with all its dependencies in a virtual instance that can be easily replicated and scaled, with low boot-up time, and low virtualization overhead [45].

Fog applications mainly use one of the following two types of organizations: 1) Replication, where multiple instances of the same application are deployed in different fog nodes [51]. 2) Multi-component, where each component is developed as a microservice located in one or more fog nodes [6]. In the microservices approach, each business capability of the application is implemented as a small, self-contained service with a well-defined application programming interface (API). Unlike the monolithic architecture, where all the functional logic for handling user requests runs within a single process, in microservices the functional logic is separated in services and each service is responsible for processing a single task [52]. Microservices are easily containerizable because the small services are loosely coupled, independent and self-sustained instances [45]. Due to the wide use of the microservices, in this thesis, we mainly consider containerized fog computing applications following the microservice-based organization.

2.3 Kubernetes

Kubernetes (K8s) is an open-source and vendor-neutral container orchestration platforms which automates the deployment, scaling, and management of containerized applications and resources in distributed environments such as cloud data centers [53]. According to a survey conducted in 2019 by the Cloud Native Computing Foundation, 78% of the respondents were using Kubernetes in production [54]. Today, half of organizations running containers use Kubernetes, whether in self-managed clusters, or through a cloud provider service [55]. K8s was initially designed for cloud scenarios, but it is now being extended to make it suitable for fog computing scenarios [48], [56]–[58]. As presented in Figure 2.3, since 2017 Kubernetes adoption has more than doubled and it continues to grow steadily, without any signs of slowing down [55]. As a result we decided to base our system implementation on Kubernetes. However, the same research contributions may arguably be implemented in any other fog orchestration platform.

K8s is installed on a collection of worker nodes which form a cluster, and which are managed by a master node. However, in production environments more than one master node may be created to increase the resiliency and availability of the system [59]. The master is the central controller of the system. It is composed of different components to control and manage the worker nodes and the containers in the system [60].

Kubernetes manages the available resources to maintain applications in the desired state as declared in the manifest files written by the K8s users. Kubernetes automatically

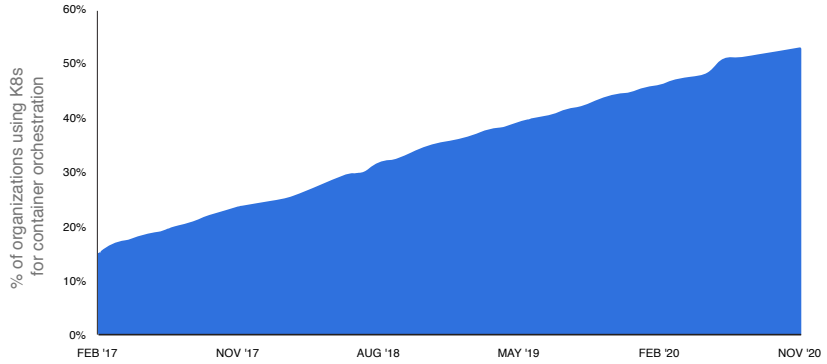

Figure 2.3 – Kubernetes share among container orchestrators since 2017 [55].

scales up and down containerized applications based on available resources in the cluster and the desired declared state of the application in the manifest.

In K8s, the smallest deployable software element is a *pod*, defined as a group of closely-related containers which perform a single function or service and which must be deployed on a single node. The containers inside a pod share the same resources and the same private IP address. Multiple pods of the same application may be deployed and exposed to the external world using a *service*. Note that Kubernetes does not provide a definition of an application. In this thesis, we define an application as a set of K8s services which share the same namespace and application label. Figure 2.4 shows the relation between container, pod, service and application.

2.3.1 Container runtimes in Kubernetes

A container runtime is the software that allows the execution of containers within a cluster [61]. It is responsible for pulling and running the container images in K8s [62]. K8s supports multiple container runtimes such as Docker, CRI-O [63] and Containerd [64]. Currently, the most common runtime is Docker. However, recently, Kubernets announced they are deprecating Docker as its default container runtime, although Docker-produced images will continue to work in the cluster with all K8s' container runtimes [65]. In this thesis, we use Docker as our runtime. Note that as we are using the properties of the container network interface, Docker's deprecation has no effect on our work and our solution should work correctly with the other runtimes.

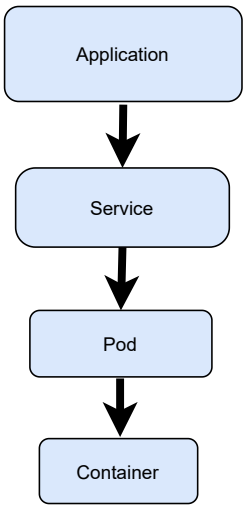

Figure 2.4 – The relation between application, service, pod and container in K8s.

Docker, as a popular containerization tool, has been widely used in the fog environments [39], [66]. In [66], the use of docker in fog environments has been evaluated along four fundamental criteria: (1) deployment and termination; (2) resource & service management; (3) fault tolerance; and (4) caching. The authors conclude that Docker provides fast deployment, small footprint and good performance, which makes it suitable to be used in resource-constrained environments such as fog platforms.

2.3.2 Networking in Kubernetes

The applications, services, pods and containers inside a Kubernetes cluster communicate with each other using internal networking. Kubernetes uses the Container Network Interface (CNI) which is a set of standards that define how a software should be developed to implement networking in a containerized environment [61]. Some of the most common CNI plugins for K8s are Flannel [67], Calico [68] and Weave Net [69]. Also, some cloud providers have developed their own CNI plugins for running in their cloud infrastructures like Azure CNI for Kubernetes [70]. Kubernetes imposes the following fundamental requirements for such network plugins [61]:

1. Pods on a node can communicate with all pods on all nodes without NAT;
2. Kubernetes system daemons on a node can communicate with all pods on the same node;

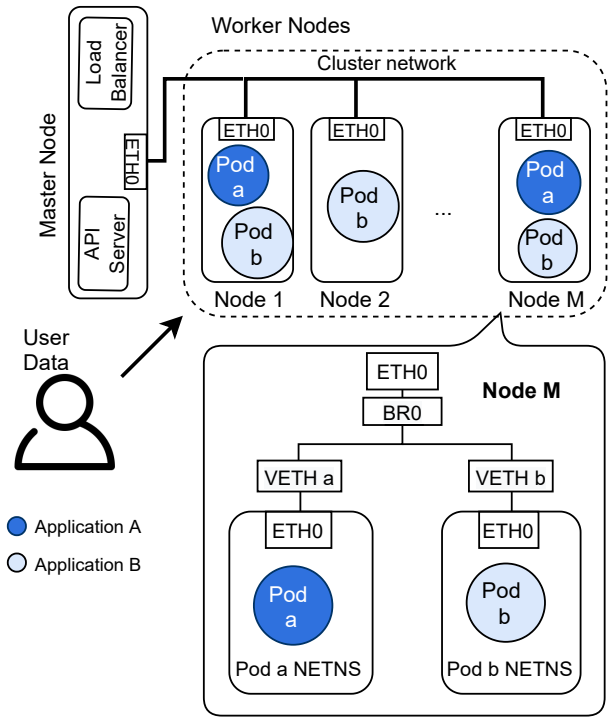

Figure 2.5 – A simplified version of Kubernetes networking.

3. Pods in the host network of a node can communicate with all pods on all nodes without NAT;

In Kubernetes, all containers within a single pod share their network namespaces, including their private IP address and virtual MAC address [71]. This means that containers within a pod can all reach each other's ports on localhost. Moreover, containers within a pod must coordinate their port usage.

In K8s, each pod and each service is assigned a unique private IP address. These private addresses are then bridged inside the node to communicate with the rest of the world. Each container inside the pod is assigned a specific virtual network interface (VETH) that is seen by the host. Figure 2.5 illustrates the K8s network organization.

In this thesis, we use Docker as our container runtime for Kubernetes. Docker uses bridged mode as its default networking setup. In this mode, a bridge is setup on the host, commonly named docker0, and a pair of virtual interfaces (veth) is created for each container. One side of the veth pair remains on the host and attaches to the bridge while the other side is placed inside the container's namespaces. An IP address is allocated to containers on the bridge's network and traffic is routed though this bridge to the container [72]. To access the network traffic that goes in/out from a container, one first

needs to find the veth interface of the container and then capture the traffic from that interface using standard tools such as tcp dump.

2.4 Security analysis methodologies

2.4.1 Common Criteria for computer security certification

Common Criteria (CC) is an ISO/IEC standard (ISO/IEC 15408) for computer security certification [73]. It provides a methodology to evaluate the security of IT products and to rate products according to Evaluation Assurance Levels (EAL). The EAL (EAL1 through EAL7) of an IT product or system is a numerical grade assigned following the completion of a CC security evaluation. The increasing assurance levels reflect added assurance requirements that must be met to achieve Common Criteria certification [73].

To attain a CC certificate, the product should be sent to a CC laboratory for evaluation, which has a cost. Moreover, a group of technical people involved in the design and development of the product should explain the product to the evaluation team, which may require months of labor. Furthermore, for evaluation of a product according to the CC methodology, a document called Protection Profile (PP) is needed. The CC laboratory evaluates the product according to this document, which is written by an association that cares about users' security. In most cases, it is a government sector. A PP is a representation of users' security requirements, and it includes the security requirements for the product. Through the evaluation process, the product is investigated against these specified security requirements.

Currently there is no PP document for evaluating a fog computing product. Moreover, even with the existence of such document, the CC evaluation process would not be affordable for all perspective actors, because of the associated time and cost. However, as the CC evaluation process is complete and systematic, we use it as an inspiring methodology for our fog security contribution as detailed in Chapter 4 of this thesis. We tailor the CC procedure and make use of a modeling tool which is called attack trees to create our methodology.

2.4.2 Attack trees

Security has different meanings in different contexts and environments. We need to know "secure from what?" and "secure from whom?" [24]. To secure a system, we need

to know all the possible ways that the system can be attacked to be able to design countermeasures to tackle them [24]. Knowing the abilities of the attacker and the operation environment of the system gives us another vector to understand and defend the system. The first question, which is to secure from what, can be modeled using attack trees. The latter question, which is from whom, can be investigated by the threat analysis of the system.

In every system, assets are the entities that the attackers aim to access, and the system owner/user wants to keep safe. For example, if we define our system as a mobile application, the user data stored in the application is the asset of it and needs to be kept safe from attackers.

Attack trees were introduced in [24]. They form a convenient approach to analyze the different ways in which a system can be attacked. It is an analytical technique (top-down) where an undesirable event is defined and the system is then analyzed to find the combinations of basic events that could lead to that undesirable event. The undesirable event represents either the objective of the attacker or a property of the asset to protect by the defender. This seminal work has been extended to Defense Trees [74], Attack Countermeasure Trees [75], Boolean logic Driven Markov Processes [76] and others.

An attack tree follows a hierarchical structure, where the top-most element represents the main objective of the attacker. The other nodes are either referred to as sub-goals or leaf nodes. In Figure 2.6, an attack tree for unauthorized entrance to a locked room is depicted.

At the bottom are leaf nodes, which are the potential attacks of a systems which cannot be detailed further. In the middle, there are sub nodes made of a combination of leaf nodes. These leaf nodes and intermediate nodes are combined using a logical relationship of either OR-gate or AND-gate. An OR-gate is used to represent an alternative attack relationship whereas, an AND-gate is used to represent a set of attacks that if occurred together would sequel a certain goal or intermediate node.

If the attack tree also includes the defense mechanisms, it is called an attack-defense tree. In this case, the security analyst is able to check if all the attacks presented in the tree are mitigated using at least one countermeasure. In Figure 2.7, the corresponding attack-defense tree for unauthorized entrance into the room is depicted.

As can be seen in Figure 2.7, the countermeasures are represented with NOT gates. If a countermeasure is located as near to the root level, it can tackle more threats. For example, the use of “Biometric key” can cover all the branches under the “using key”

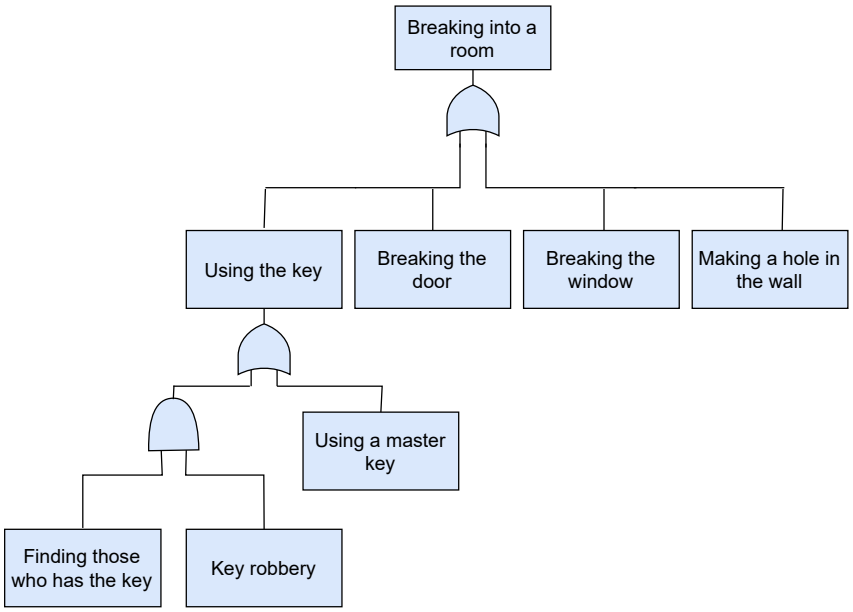

Figure 2.6 – Attack tree for unauthorized entrance into a room.

attack. As a general rule in attack-defense trees, the higher the countermeasure is in the tree, the better is its coverage.

Our first contribution in this thesis is the introduction of a methodology for systematic analysis of security of fog computing systems. We use Common Criteria approach and the attack tree model as a basis for our methodology which will be detailed in Chapter 4 of this thesis.

2.5 Machine learning techniques for network traffic classification

In this thesis, we use Machine Learning (ML) techniques to understand the behavior of the applications running in the fog. We collect application signals such as the network traffic, and utilize ML to classify traffic samples and derive information about the application's behavior. We detail this procedure in Chapter 5.

Machine learning has recently received considerable attention for its ability to solve problems that are not easy to solve using the traditional rule-based programming techniques [77]. Many complex systems such as recommendation systems, search engines, social-media feeds and voice assistants are powered by machine learning nowadays [78].

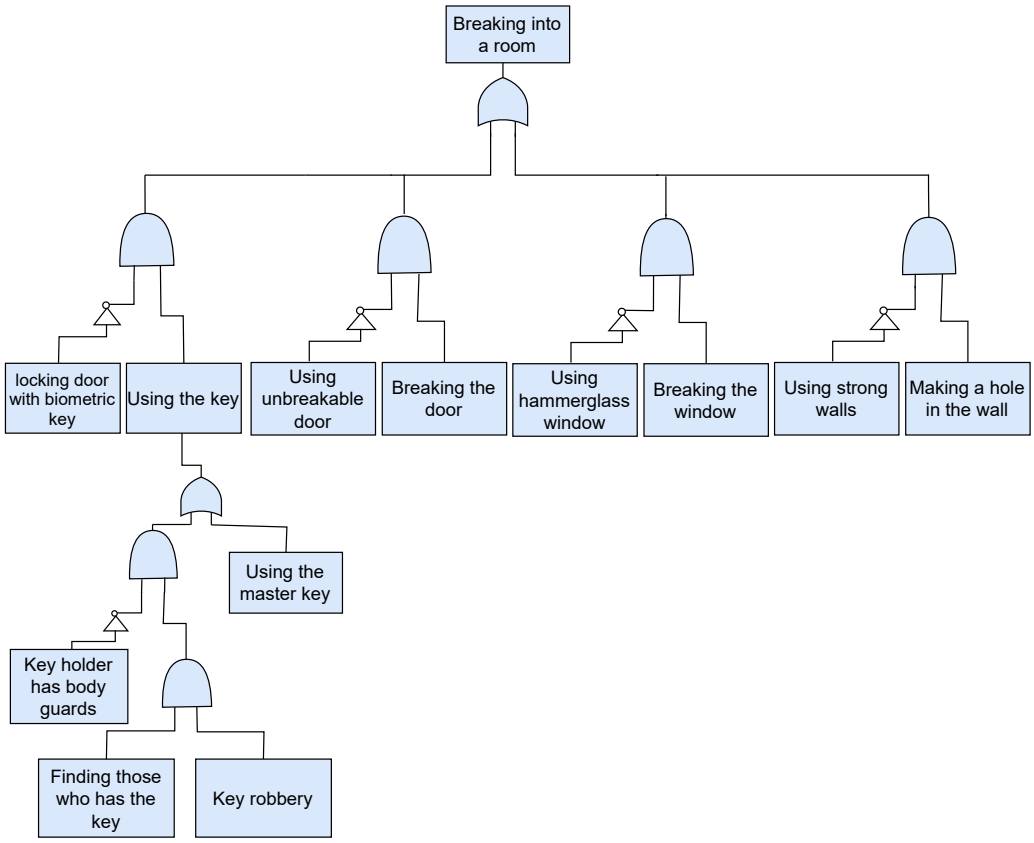

Figure 2.7 – Attack-defense tree for unauthorized entrance into a room.

Machine learning can be opposed to rule-based programming where a human programs explicit algorithms telling the machine how to execute all the required steps to solve a specific problem. However, for complex tasks, it may be challenging for a human to imagine all the necessary steps and create the needed algorithms. Machine learning is the study of computer algorithms that improve automatically through experience [79]. Machine learning algorithms build a model based on sample data, known as "training data", in order to issue predictions or take decisions without being explicitly programmed to do so [80].

There are two main approaches to address a problem using machine learning: (1) supervised learning; and (2) unsupervised learning. Supervised machine learning is the practice of finding algorithms that reason from training data to produce general hypotheses, which then make predictions about future input data [81]. The goal of supervised learning is the prediction of an output associated to a given input. One therefore needs to have a dataset which comprises of desired output (labeled outputs) for a set of given inputs. On the other hand, in unsupervised learning the purpose is spotting patterns in

the inputs. Unsupervised learning does not require labeled outputs, as the goal is to infer the structure present within the dataset [82].

In this thesis, we use supervised machine learning techniques for classification of the applications' network traffic. Network traffic classification, as the name suggests, essentially aims to classify a set of network traffics (inputs) in some number of predefined classes [83]. In the remainder of this section we describe supervised machine learning techniques in more details.

2.5.1 Supervised machine learning

The process of applying supervised ML to a classification problem is described in Figure 2.8 [81]. The first step is collecting the relevant data. We may need some insight about the problem to be able to capture the relevant data. Otherwise, the simplest method is "brute-forcing", which means measuring everything available in the hope that the right information can be isolated. As the collected dataset may have some noise or missing information, a pre-processing step is often required.

Every sample that is used in machine learning is also referred to as instance. An instance is a single object of the world from which a model will be learned, or on which a model will be used (e.g., for prediction) [84]. Instances usually require a pre-processing step to extract one or more numerical metrics called feature that are then used as input of the ML system. The instances are used in the form of feature sets, which means that one should extract features from an instance and train or test a model using the features.

After data collection and data pre-processing, the next step is to select the instances for the training phase. The simplest method randomly splits the training and test sets.

The next step, algorithm selection, is an important step in building the classifier. One needs to try different algorithms with different parameters to find a satisfying model. There are several popular supervised learning algorithms such as Logistic Regression, Decision Trees, Support Vector Machine and Random forest, while each of them have different configurable hyper parameters [85], [86]. The hyper parameters are parameters whose values are set by hand prior to the commencement of the learning process [87]. In order to know if a model is satisfying, we need to perform evaluations on the model.

After selecting the algorithm, we build a model using the training dataset. In this step, we use part of the training dataset for evaluating the performance of the algorithm and its associated parameters before moving to the next step which is testing with the

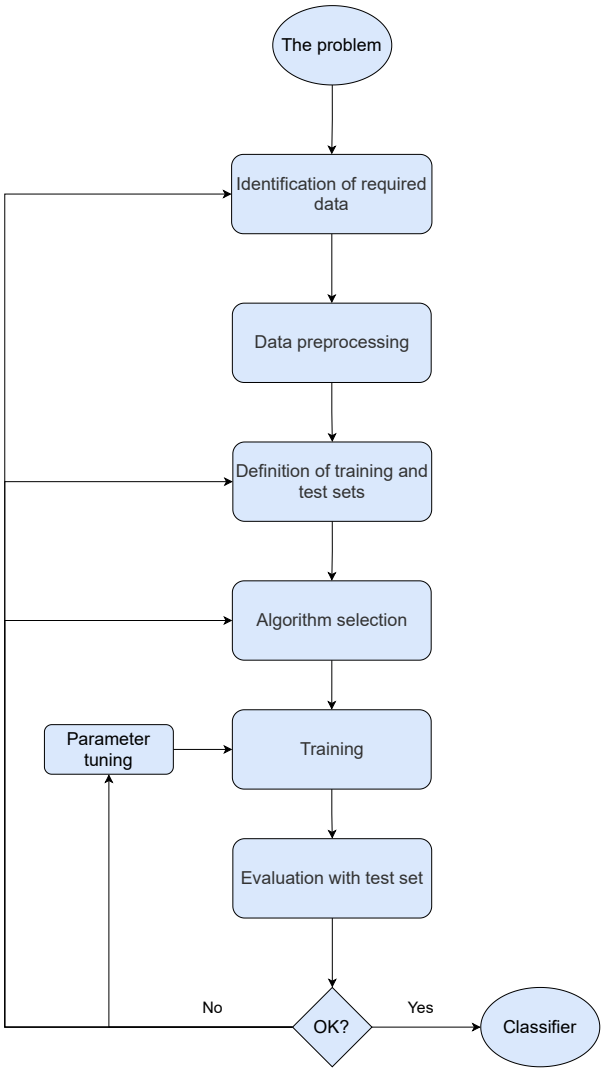

Figure 2.8 – The process of supervised machine learning [81].

test set. If the performance of the model is satisfactory then we can use it as our final classifier. Otherwise we go back to the algorithm selection step and redo the training.

There are two main techniques to use the training dataset. One technique is to split the training dataset by using two-thirds for training and the other third for estimating the performance of the model in training phase [81]. The other technique, called cross-validation, splits the training dataset in several parts and repeats training and validating using one part of the data for validating and the remaining parts of data for training [88].

After the training step, we can test the model with the test dataset. If the error rate of the model with the test dataset is unsatisfactory, a variety of factors in the previous

Table 2.1 – The confusion matrix for a classifier with three classes.

	True class		
Predicted class	Class 1	Class 2	Class 3
Class 1	c_{11}	c_{12}	c_{13}
Class 2	c_{21}	c_{22}	c_{23}
Class 3	c_{31}	c_{32}	c_{33}

steps of the learning need to be examined. This procedure continues until we obtain a model with a tolerable error rate.

2.5.2 Performance indicators for a classifier

As discussed in previous section, it is necessary to have reliable performance indicators to be able to select a model over another one. The performance of a ML model is typically evaluated using three standard metrics: precision, recall and F-score.

In order to describe these performance indicators, it is convenient to first introduce the Confusion Matrix (CM). The Confusion Matrix is a specific table layout that allows one to visualize the performance of a classification algorithm. The structure of a CM for $k = 3$ classes is presented in Table 2.1.

In the Confusion Matrix, c_{ij} represents the number of instances that are classified in class j, whereas these instances actually belong to class i. When i is equal to j, the instances are correctly classified.

We define the *Precision* $prec_j$ for a given class j as follows 2.1:

$$prec_j = \frac{c_{ij}}{m_j} \tag{2.1}$$

Where m_j is the number of examples predicted as belonging to class j. The overall precision $Prec$ of the classifier is defined as the weighed mean of precisions on the individual classes:

$$Prec = \sum_{j=1}^{k} \frac{m_j}{N} prec_j = \frac{1}{N} \sum_{j=1}^{k} c_{jj} \tag{2.2}$$

Where N is the total number of instances.

Recall (rec_j) for a given class j is equal to Formula 2.3:

$$rec_j = \frac{c_{jj}}{n_j} \tag{2.3}$$

Where n_j is the number of examples actually belonging to class j.

In a classification problem, a precision score of 1.0 for a $class_j$ means that every item labeled as belonging to $class_j$ does indeed belong to $class_j$. On the other hand, it says nothing about the number of items from $class_j$ that were not labeled correctly. A recall of 1.0 means that every item from $class_j$ was labeled as belonging to $class_j$ (but it says nothing about how many items from other classes were incorrectly also labeled as belonging to $class_j$) [89].

To combine the Precision and Recall metrics into a single "goodness" evaluation metric, a new performance indicator, *F-score*, is defined. The *F-score* (F_j) for a given class j is a performance indicator which tries to balance between the precision and recall. The *F-score* is defined as Formula 2.4:

$$F_j = 2 \times \frac{prec_j \times rec_j}{prec_j + rec_j} \tag{2.4}$$

The overall *F-score* (F) is given by Formula 2.5:

$$F = \frac{1}{k} \sum_{j=1}^{k} F_j \tag{2.5}$$

The value of F for a perfect classifier would be equal to 1.

2.5.3 Decision tree algorithm

As will be extensively discussed in Chapter 5, the most efficient classification algorithm for our purpose is Decision Tree. We introduce this algorithm in this section.

Decision tree is a supervised learning technique that can be used for classification problems. Decision tree provides all the possible solutions to a problem (decision) based on given conditions. In a decision tree, for predicting the class of a given dataset, the algorithm starts from the root node of the tree and compares the values of root attribute with the real dataset attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf of the tree [90].

The Decision tree algorithm does not make any assumption on the distribution of data, and it provides understandable explanation over the prediction when the number

of features are limited. However, the tree may grow to be very complex while training complicated datasets [91].

Chapter 3

State of the art

This chapter provides a review on the state of the art in the research area related to the thesis. First, we discuss the security of fog computing and its related entities such as cloud computing and distributed systems. Second, we focus on the current approaches for automatic privacy compliance checking in fog and cloud environments. Finally, we provide the state-of-the-art in the network traffic analysis and classification techniques as we use these techniques to understand the behavior of fog applications.

3.1 Security in fog computing and its related entities

As fog computing lies at the cross-roads of some well-established research themes (distributed systems, IoT and cloud computing), we survey the relevant state-of-the-art on security issues and potential countermeasures in distributed systems, IoT, cloud computing and fog computing. Finally, we present some examples of security threats in fog computing.

3.1.1 Security in distributed systems

Distributed systems are composed of units (also referred to as nodes) which carry out computational tasks and interact by exchanging messages over communication channels. Units can be geographically dispersed, and may be under the administrative control of different entities. In a distributed system the overall system-level functionality emerges out of the coordinated action of single computation units; no single controller is present, but each unit carries out its activities with a well-defined level of autonomy [92].

From a security standpoint, distributed systems present a number of challenges which are not present (or exist, but in a radically different form) in centralized IT systems. The more radical difference lies in the fact that a unit has little (or no) way to know whether the other units in the system behave correctly. This opens up significant opportunities

for the introduction of faulty or malicious software [93]. The latency inherently related to the usage of communication channels means also that the identification of a security threat or attack may incur high delays, which makes it hard to defend the system in a coordinated fashion. In case units are geographically distributed, it may also be difficult to guarantee the physical security of the different computation units. Messages exchanged among nodes may be eavesdropped or corrupted while in transit, thereby calling for the introduction of appropriate measures to ensure message integrity and confidentiality. At the same time, the fact that the overall functionality emerges out of interactions among different units, and the lack of a single point of failure, may offer advantages in terms of robustness, reliability and resilience [92]. If data is also distributed across the various nodes, a single compromised node may give the attacker access only to a (potentially small) subset of the system-level information, thereby limiting the impact of such attack in terms of data confidentiality and integrity.

As nodes in a distributed system rely on a communication network to interact by exchanging messages, the security of the network itself is crucial for the operations of the overall system. This covers a whole set of aspects, ranging from link-level security (ensuring that an established communication channel between two nodes cannot be disrupted and/or messages passing over such channels cannot be modified or eavesdropped) all the way to network-level security (covering aspects such as – distributed – denial of service, routing security etc.). For a complete description and classification of network attacks, we refer the reader to [94] and [95].

Trust

Trust is a fundamental notion in distributed systems. By definition, computation units in a distributed system cannot be considered fully trusted a priori. This requires the introduction of mechanisms able to ensure correct functionality even in the presence of misbehaving and/or malicious nodes. A well-known example in this case is the Byzantine general problems, which covers the case in which nodes need to achieve consensus (a fundamental function in distributed systems) in the presence of "traitors" [96]. This has given rise to a rich and active line of research, in which distributed algorithms able to ensure correct functionality in the presence of malicious nodes have been proposed. Yet, fundamental limitations do exist on the fraction of malicious nodes that can be sustained by a system in order to ensure correct operations.

A related notion is that of "reputation". As it may be hard to identify in a controlled way which nodes are malicious/misbehaving, a widely used idea in distributed systems is for nodes to monitor over time the behavior of other nodes they interact with. This requires nodes to have: (1) the knowledge of the expected behavior by other nodes [trivial if all nodes behave the same, potentially extremely complex in other situations]; (2) means to observe the behavior of other nodes [typically by logging and analyzing messages originated by the observed nodes]. Over time, a node can therefore build a score representing the "level of trust" of the nodes it interacts with [97]. Such measure is meant to mitigate the impact of a potentially misbehaving node, by basically excluding it from interactions or limiting its involvement in critical interactions.

Authentication and non-repudiation

The authentication of the identity of a communication partner represents a key security issue in distributed systems. Authentication provides a basis for auditing and accounting, and is a prerequisite for the implementation of many access control schemes, required to grant or restrict access to given resources. In the case of distributed systems, authentication refers not only to the identity of users (as in the case of centralized systems) but also of nodes (better: originators of messages). Solutions typically used are based on public key schemes (and certificates) and thereby requires the presence of a trusted certification authority (CA). While this may seem contradictory with the notion of "distributed" systems (a CA may represent a single point of failure for the security of the overall system), it is considered a pragmatic choice and safely used in most cases [98].

Trust management solutions [99], which use a combination of credentials and local security policies to allow direct authorization of security-critical actions, partially alleviate the issues presented by the presence of a centralized CA. Yet, they require a tight coordination in terms of security policies (distribution, updates, revocation etc.) and present a significant complexity, which makes them not suitable for usage in systems where the availability of resources (computing power, communication bandwidth) may be problematic.

Some lines of research explore the possibility of so-called zero-trust schemes [100]; albeit promising, this research line is not yet consolidated.

Non-repudiation is a security property requiring the fact that no node can deny having sent a message originated from it. Non-repudiation is a prerequisite for the implementa-

tion of an audit trail system. Non-repudiation requires both nodes and messages to be identifiable, and it requires means to ensure the authenticity of messages.

3.1.2 Security in IoT

Security is recognized as one of the main threats (and potential show-stoppers) in the adoption of IoT technologies and services [12], [101], [102]. The merge of virtual (digital) and physical dimensions enabled by IoT makes indeed security issues even more pressing than in traditional IT systems. Data breaches could easily reveal personal and even sensitive information about individuals; attacks may prevent the correct working of IoT-enabled systems such as smart home systems and connected cars, with potentially life-threatening effects. We analyze, with a data-centric approach, in detail the security issues in IoT systems and discuss the potential applicability of IoT security solutions to fog computing architectures. We consider four data security dimensions: integrity, authenticity, confidentiality and privacy. For more details on the state-of-the-art, we refer the reader to a number of excellent surveys presenting the state-of-the-art in *security in IoT* [103]–[107].

Our focus is in particular on data in transit, i.e., data that moves dynamically across various nodes. This does not mean, clearly, that security issues related to data at rest are irrelevant, but, rather, that they do not present specific peculiarities in IoT with respect to traditional IT scenarios. For the data in transit part, instead, the limitations of IoT devices in terms of computing/memory/storage/energy represent challenges for existing approaches and require novel solutions [108].

Data integrity

Data integrity refers to ensuring that a given (IoT-generated) data entity has not been altered or tampered with; otherwise stated, that it has not changed from a given 'reference version' (typically: the original one) [109]. In many IoT scenarios data integrity plays a key role: for example, in a health application monitoring physiological parameters, if sensor data is tampered with, this may have noxious consequences.

The traditional approach to ensure data integrity assumes that the attacker does not control the operating system, and is based upon the usage of encryption techniques together with a cyclic redundancy check (CRC). Cyclic redundancy check are commonly used to provide integrity against unintentional alteration of data (due, e.g., to an error

on the communication channel). Yet they are not sufficient in the presence of a malicious device in the system; the latter one could indeed replace the message with a fake version thereof, which respects the CRC structure. Encryption by itself does not guarantee the integrity of data, as encrypted data may be easily replaced by a fake version thereof. If the CRC structure is not known to the attacker, though, a combination of CRC and encryption provides sufficient means for the intended recipient to check whether the data packet has been altered or not. Similarly, checksums are a popular way of checking (not guaranteeing) data integrity; in this case the data entity (packet, message) is provided as input to a hash function; the resulting output is then transmitted to the intended recipient. By repeating the hash function on the received data and by comparing the results, it becomes therefore possible to check whether the data entity has been corrupted/altered.

Encryption however presents the problem of being typically a rather resources-hungry process. In a traditional IoT scenario, where resource-constrained nodes communicate directly with the cloud, the usage of strong encryption may be problematic from an energy consumption point of view [110]. In a fog architecture, this issue is still relevant for the IoT-to-fog communication, whereas it can be safely assumed that the fog node has sufficient resources to run encryption functions before forwarding data to the cloud tier.

Data authenticity

Authenticity of data can be seen as a specific case of data integrity, in which the 'reference version' is the original (or: authentic) one. In traditional IoT architectures, where data is pushed from the IoT device directly to the cloud, the boundary between authenticity and integrity is rather blurred [111]; in the fog case, where this additional middle layer is present, this requires that data is not tampered within the fog node.

Data confidentiality

Confidentiality refers to the ability to hide information from those unauthorized to view it. This includes, in rigorous terms, the content of the data entity itself, as well as the identity of the sender and of the target recipient of said information. In most cases, however, confidentiality is limited only to the data entity itself, leaving in plain sight the id of sender and receiver. In terms of relevance, think of a smart home scenario in which sensors measure the presence of the inhabitants; such information is confidential and should not leak (e.g., to burglars).

Encryption methods (in particular in their asymmetric form) are commonly used to preserve confidentiality of data content [112]. This can be used to protect the information content, yet in general it is not suited to protect the identity of the sender and recipient, which can be accessed by a malicious eavesdropper by just monitoring packets, e.g. over a wireless link. When data is at rest, authentication methods (passwords, biometric verification, etc.) are traditionally used to control (and restrict/prevent) access to confidential data. This requires also the definition of suitable access privileges that map the identity of a user with the data items she is authorized to access.

In the IoT device level, again, the usage of encryption methods is partially limited by the available computational (and energy resources) [113]; if a fog layer is present, depending on whether data is stored locally or is just forwarded to the cloud, an appropriate set of access control methods might be required [114].

Data privacy

The term data privacy is used often as a synonymous of data confidentiality, in particular when personal data (as broadly defined, e.g., by GDPR [14]) are considered. In reality privacy is a legal term, and privacy refers to a normative right by individuals to make sure that some information remains private and not accessed by non-authorized parties.

As IoT data may often refer to individuals (think of wearable devices measuring physiological parameters, all the way to smart meters measuring per-device energy consumption patterns), privacy issues are a primary concern [12], [108], [109], [115]. Solutions aimed at guaranteeing the privacy of data in the IoT sphere include a number of technical measures meant to ensure data confidentiality as well as a set of non-technical measure (e.g., user awareness programs, compliance monitoring systems and internal audit procedures) making sure data is not accidentally disclosed by the relevant controller or processor.

3.1.3 Security in cloud computing

Security is universally recognized as one of the main challenges in cloud computing [116]. Although cloud computing shares many security issues with traditional distributed systems, its specificities such as virtualization and multi-tenancy create a number of specific, challenging issues. Such issues can be broadly classified into issues that are faced by the cloud platform providers, and issues that are faced by the cloud plat-

form tenants. To fully secure a cloud-based system it is therefore important that both the provider and its tenants take appropriate security measures.

The cloud Security Alliance periodically releases a documented list of the top threats to cloud computing [117]. The main ones are described below.

Data breaches: a data breach is an incident in which sensitive, protected or confidential information is released, viewed, stolen or used by an individual who is not authorized to do so. Although data breaches may happen in a wide range of on-line system, the fact that cloud tenants lose their ability to have physical access to the servers hosting their information requires the use of more sophisticated solutions.

Insufficient identity, credential and access management: data breaches and enabling of attacks can occur because of a lack of scalable identity access management systems, failure to use multifactor authentication, weak password use, and a lack of ongoing automated rotation of cryptographic keys, passwords and certificates. It is therefore a responsibility of the cloud tenants to take informed decisions on how to configure the identity access management systems and how to keep their credentials confidential.

Insecure Interfaces and APIs: cloud computing providers expose a set of software user interfaces (UIs) or application programming interfaces (APIs) that customers use to manage and interact with cloud services. The security and availability of general cloud services is dependent on the security of these basic APIs. Cloud providers must therefore secure their UIs and APIs, but also engage in extensive certification processes to convince their tenants of the security of their services.

System vulnerabilities: System vulnerabilities are exploitable bugs in programs that attackers can use to infiltrate a computer system for the purpose of stealing data, taking control of the system or disrupting service operations. Although this type of vulnerability applies to almost all computer systems, in cloud environments systems from various organizations are placed in close proximity to each other, and given access to shared memory and resources, creating a new attack surface.

Account hijacking: although attack methods such as Phishing, fraud and exploitation of software vulnerabilities are not specific to the cloud, hijacking of a cloud account may have very serious consequences such as eavesdropping on activities and transactions, manipulating data, returning falsified information and redirecting clients to illegitimate sites.

Malicious insiders: A malicious insider threat is a current or former employee, contractor, or other business partner who has or had authorized access to an organization's network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization's information or information systems. Cloud computing increases the seriousness of this thread because it increases the number and size of organizations taking part in any cloud-based application.

Since fog computing is defined as an extension of cloud computing, it clearly also inherits all of its security issues and potential solutions. The purpose of the present thesis is to extend this analysis including threats that are specific to fog computing.

3.1.4 Security in fog computing

Security in fog computing can be seen from different perspectives. We can think of fog computing as an entity which adds the attack surface of an IoT-Cloud system and add vulnerabilities to the system. On the other hand, fog computing can be seen as an entity which assists to increase the security of cloud and IoT systems. In this thesis, we first see fog computing with the first perspective and introduce a methodology for systematic security analysis of fog computing systems. Later, in the second contribution of this thesis, we assume we have analyzed the security of a fog computing system using our methodology and gained a secure fog computing platform. We therefore use fog computing as an entity for enhancing the privacy of the system.

In this section we provide the state-of-the-art in fog computing security with the two mentioned perspectives: (1) fog computing as a security problem; and (2) fog computing as a security solution.

Fog computing as a security problem

Although fog computing naturally inherits most of the security issues and possible solution from cloud computing, it also has a number of specificities due to its decentralized architecture or its expected usage. In [118], the authors discuss the commonalities and differences between security issues found in fog computing, mobile edge computing, mobile cloud computing and cloud computing. Most security-related issues in these environments relate to identity and authentication, access control, protocol and network security, trust management, intrusion detection, privacy, virtualization, and forensics. A similar set of

issues is discussed in [49] and [119]. Although these papers try to build an exhaustive list of potential security issues in fog computing, they do not make use of a systematic methodology to explore these issues in depth.

A specific issue in fog computing relates to the fact that fog nodes typically consist of small machines distributed across a large geographical area, and are thus arguably harder to secure physically. This mandates special measures to be taken by clients of the fog nodes to verify the integrity of query results produced by untrusted nodes [120]. Similarly, for the same reasons, we can identify a significant exposure to man-in-the-middle attacks, where an attacker creates fake fog nodes which attempt to obtain private data from IoT devices and/or to influence these devices and make them misbehave. In [121], the authors mention *Integrity attacks against machine learning data* that targets ML training data and hence the output machine learning models rather than the user personal data in the fog layer. Such topics are typically addressed by designing specific authentication and authorization mechanisms [122].

Another source of potential security issues comes from specificities of fog computing applications and the way they handle data and fog computing resources. Khan *et al.* extensively study a set of thirteen reference fog computing applications and analyze which of twelve potential security issues apply to which application [123]. One may expect that some fog computing applications — such as driving assistance applications for communicating vehicles — will be shared by a large number of entities, which hints at Blockchain-based solutions [124], [125] and [126] due to the distributed nature of Blockchain. As Blockchain provides tamper-proof storage and uses consensus mechanism to confirm past events, it may be used as a security means.

In [121], [127], the authors provide a taxonomy of the security issues of edge computing environment along with some potential countermeasures. Although the papers target edge computing, the provided list can be also beneficial for the analysis of fog computing security.

Fog computing as a security solution

Fog computing can also be seen as an enabler for improving the privacy or security of other systems. Fog nodes may for example be used to address security issues experienced by IoT devices [128]. In [129], the authors discuss how the heterogeneity and third-party attribute of fog computing can be leveraged to address the security requirements of Industrial IoT(IIoT).

Fog computing sometimes aids the security of the IoT systems by providing computational support for the implementation of security protocols or cryptographic key generation operations [130]–[132] which require computation power that IoT devices normally do not posses.

The fact that fog platform can process privacy-sensitive data close to the user rather than in a single centralized cloud may be exploited to increase the privacy guarantees for applications which deal with personal data [133]. In [134], the authors utilize fog computing for real-time extraction of relevant information from video data and hence avoid sending personal data (e.g., face) to the application in the cloud.

As fog applications have privileged access to personal data, they normally publish a privacy policy and claim they handle personal data according to it. However, the compliance of these applications to their privacy policy should therefore not be taken for granted but rather be evaluated. In this thesis, we argue that automated privacy compliance checking in fog computing platforms is feasible, and outline a research roadmap towards the development of such systems. We discuss it in further details in Chapter 5 of this thesis.

3.2 Privacy compliance checking

3.2.1 Privacy policy interpretation

A study in 2008 pointed out that for any normal individual reading privacy texts for all applications he/she makes use of will take normally 25 workdays every year [135]. Given the fact that this study was done in 2008, and we use more applications now, the required amount of time to read privacy policies has certainly increased. On the other hand, understanding legal texts, such as an application's privacy policy, requires skills that most users do not possess. As a result, most users tend to accept any privacy policy terms without even reading. This however does not mean people lack interest in the respect of their privacy. To bridge this gap, it is becoming increasingly necessary to automatically analyze and interpret the privacy policies, and to relieve users from this manual burden.

Some studies target building a machine-readable language for describing privacy policy terms. A machine-readable privacy policy is easier to use in automatic privacy compliance checking systems. Although several languages for describing privacy policies exist [136],

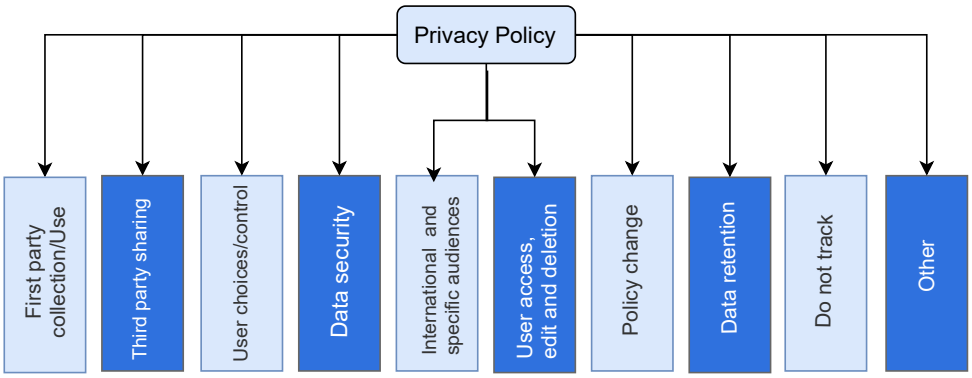

Figure 3.1 – The first level of privacy principles taxonomy introduced in [137].

the main problem is that there is no commonly agreed privacy policy language to be employed by most of the application providers.

Other works aim to interpret the privacy policy and extract meaning from its text. In [137], the authors define a taxonomy of privacy policy principles and annotated 115 privacy policy texts based on their defined taxonomy with the help of law school students. They utilized this dataset to train a model for automatic interpretation of other privacy policies.

In [138], the authors use the taxonomy and the dataset of [137] and created a dataset of 130,000 annotated privacy policies. Using deep learning techniques, they prepare a tool called *Polisis* to automatically extract privacy principles from the privacy text with 88.4% of accuracy. This converts the privacy policy to a set of privacy principles, making it machine-readable. Figure 3.1 illustrates the first level of privacy principles taxonomy used in *Polisis*.

In [21], the author provides a taxonomy of privacy policy principles aligned with GDPR, whereas the taxonomy presented in [137] is more in the direction of the California Online Privacy Protection Act (CalOPPA) [139].

3.2.2 Automatic privacy compliance checking

To automatically check the privacy compliance of an application, one needs to monitor the application's behavior and compare it with its promised privacy policy terms.

TaintDroid is a modified version of the Android operating system which monitors the flow of sensitive information handled by Android applications [140]. The authors show how personal data may be tainted to perform dynamic analysis and track the flow of sensitive

data from where it is generated to the installed applications. The system is powerful in identifying sensitive data, but it does so with no regards to the application's privacy policy. Moreover, it is restricted to unobfuscated Java applications written for Android platforms. The system requires code or platform annotation, and its checks generate a runtime overhead of about 14%.

A similar approach logs Android application behavior by intercepting requests to the Android access control system [141]. The system complements these data with other sources of information such as the privacy policy of the application and user's review about the application to measure the privacy risks of the Android applications.

To the best of our knowledge, currently available tools for checking privacy compliance are designed for Android applications only. On the other hand, in this thesis we are interested in other types of applications that are hosted in cloud or fog computing platforms. This type of applications usually does not run in Android. We therefore aim to develop generic techniques that may be applied to a wide range of applications regardless of their programming language. Moreover, we prefer avoiding having to annotate the application or the host platform to circumvent the performance penalty and provide independence from the platform. While applications are running on their host, a lot of useful information about them are accessible passively. For example, network traffic of the application is a resourceful information about the application, even the encrypted one.

3.3 Network traffic analysis and classification

An application's network traffic clearly carries useful information about applications' privacy-oriented behaviors. Network traffic analysis is the process of capturing the traffic data and finding patterns in it. Network traffic analysis can be used to categorize the traffic data into predefined classes such as normal or abnormal traffic, the type of application (streaming, file download, etc.) or even identifying specific applications (e.g., Skype, Facebook, YouTube).

A simple form of traffic classification relies on the port numbers used by the application. TCP and UDP port numbers are standardized by the Internet Assigned Numbers Authority (IANA) for each type of application, so the choice of port numbers may provide indications regarding the nature of exchanged traffic. However, malicious applications can easily use non-standard port numbers [142]. The emergence of port obfuscation, port for-

warding, protocol embedding and random ports assignments greatly reduce the accuracy of this approach.

A popular alternative is deep packet inspection (DPI) technique which essentially matches a network packet's payload with a set of stored signatures to classify network traffic. Deep packet inspection is for example used in the context of intrusion detection systems [143]. However, allowing a platform to inspect the application's payload which potentially contains personal data, may in itself violate many privacy policies. Also, DPI often cannot handle scenarios where network traffic is encrypted or where the protocol is deliberately obfuscated [144].

With the proliferation of applications which encrypt their network traffic and users who utilize Virtual Private Networks (VPN), it becomes necessary to analyze and classify encrypted data rather than plain network traffic. A number of approaches have demonstrated how machine learning techniques may be used to analyze encrypted network traffic traces. For instance, authors in [145], use random forest classifiers to determine if an HTTPS request matches a pre-defined type of action.

Machine learning is also being used for network traffic classification and characterization [144]. For instance, one may use k-nearest neighbor (k-NN) and C4.5 decision tree algorithms to detect VPN traffic and classify it as browsing vs. streaming [146]. Similarly, Deep Packet uses convolution neural networks to classify encrypted network traffic in broad classes such as FTP vs. P2P [147].

These works demonstrate the potential of machine learning techniques to analyze network traffic traces, even in the case where the communications are encrypted. However, the classes used in these classifiers are relatively coarse-grained. For example, both [146] and [147] identify streaming traffic but they do not distinguish audio from video streaming. This information is important in this thesis as the two forms of streaming may carry different types of privacy-related information. We therefore aim to design finer-grained classifiers capable of distinguishing them. We exploit similar machine-learning-based techniques, and apply them to the new domain of extracting a privacy-oriented behavior.

Chapter 4

A systematic approach in the security of fog computing systems

4.1 Introduction

Fog computing enriches cloud computing with additional compute, storage and networking resources in close proximity with the end-user devices which generate and/or consume data streams [9]. With the development of Internet of Things (IoT) systems and applications, increasing volumes of data are being produced by IoT devices at the edges of the network. In this situation, it is often not feasible to send all IoT data to a remote cloud data center and expect acceptable Quality of Service (QoS), especially for applications with low-latency requirements such as augmented reality, industrial control systems and video streaming [6]. Moreover, applications such as quantified self which use wearable sensors to monitor individuals life often deal with sensitive personal data [148]. In solely cloud-based approaches for these applications, all these sensitive data would be sent to the cloud for processing, leaving the user with little control over the usage of their data.

Several research papers already address the topic of fog computing security [49], [118], [122]. The novelty of our contribution is to introduce a methodology for exploring the security issues of fog computing following a systematic approach. In this chapter we introduce an approach, inspired by the ISO/IEC "Common Criteria" standard [23] and the attack tree model, to help security analysts, designers and developers in identifying and protecting their fog enabled systems against security vulnerabilities. We analyze a generic fog computing system based on this methodology. Our analysis brings to the fore the possible vulnerabilities that a fog system may have. It points out where the system designer should add countermeasures to tackle such vulnerabilities while designing his fog computing system.

This chapter is structured as follows. We present some attack scenarios in fog computing and apply the attack tree representation on them in section two. Our methodology is described in section three. In section four, we identify assets of fog computing and cluster them into device level, system level and service level. Section five represents our threat model. We point out possible vulnerabilities that the fog system can face and discuss possible solutions in section six. Finally, section seven concludes the chapter.

4.2 Attack scenarios in fog computing

In this section we present some potential security threats and model them using attack trees. Scenarios are just means to illustrate some situations where security can be defeated. We describe hereafter three scenarios related to different elements of the fog computing architecture.

Privacy issues with IoT devices

The new generation of wearable fitness tracker devices has the ability to provide seamless integration with on-line social networks which in turn may create security and privacy issues. A well known example is the Fitbit device [1], which sends account and password in plain text to the server and also stores user credentials in plain text. An attacker can connect to the device via Universal Serial Bus (USB) and easily catch the credential. As a consequence, user data is accessible to the attacker and user privacy is threatened. Moreover, as the attacker can access the credentials, she will be able to change the user data, whether it is stored on the IoT device or somewhere else, for example on a fog node.

This example highlights that all elements of the system, ranging from IoT devices all the way to the cloud server, must be securely designed. As the authentication process deals with user credentials, it is a key point and should be designed and managed carefully. It is indeed challenging (if not impossible) to build a secure system if elements of the system have been designed with security as a minor concern. This example also highlights the fact that not all devices should be considered trustworthy.

1. Reversing the protocol: `https://github.com/openyou/libfitbit`

Compromised fog nodes

In [149], the authors show that eavesdropping encrypted communication channel is possible if the fog node is compromised. In this scenario, a gateway that serves as a fog node is replaced by a fake/modified one. The authors evaluate this scenario to demonstrate some issues in fog computing security. After compromising the gateway, they insert malicious code into the system (in fact hooking the system calls of the Linux OS), and then analyz the CPU usage. Their use case consists of a video call between a laptop using WiFi and a 3G cell phone. The communication is encrypted from the laptop to the gateway and then encrypted to be sent through the 3G network. They demonstrate that a man-in-the-middle attack consumes few resources in fog devices and that therefore it cannot be detected by the state-of-the-art anomaly detection systems. This attack works on a large number of gateway devices, which shows the importance of ensuring the integrity of the gateway OS which acts as a fog node.

Network attack

The capability of the network can be largely reduced by Denial of Service (DoS) attacks. These attacks are intended to prevent legitimate users from accessing the services provided by fog infrastructures. At the end-user level, an attacker can jam the wireless communication channel to prevent certain users from communicating with the infrastructure. The attacker can also deplete the resources of fog nodes as a means to prevent them from allocating new resources for other users or delaying their responses. In a DoS attack, even a small set of compromised devices can request unlimited processing/storage services. As a consequence, it can lead to stall the requests made by legitimate devices. The intensity of such attacks rises manifold when a set of nodes simultaneously launch this attack.

Attack tree models

The three scenarios described above target the following properties:

— Scenario 1 (S1) User data privacy,

— Scenario 2 (S2) Fog node OS code integrity,

— Scenario 3 (S3) Availability of fog node.

Figure 4.1 illustrates the attack tree for S1. It is a succession of OR gates expressing that only one event is sufficient to reach the goal, which in this case is breaking the user's

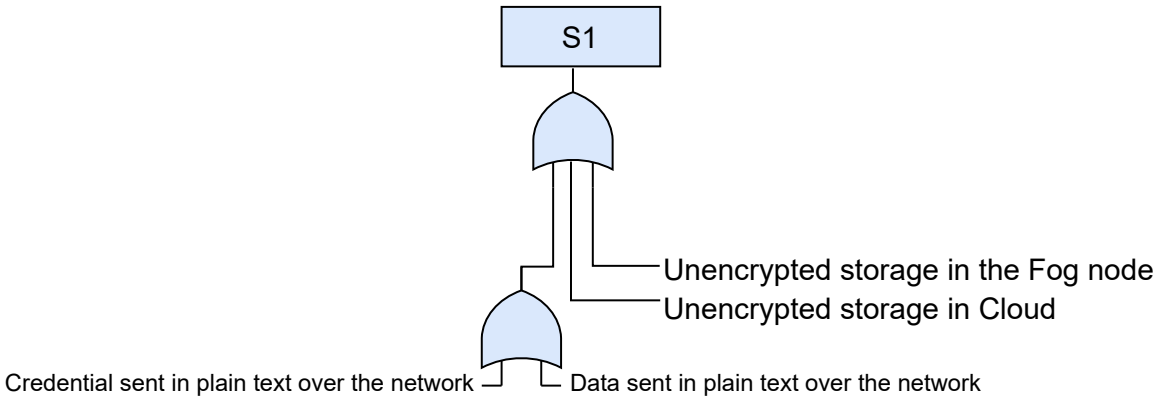

Figure 4.1 – Attack tree for scenario 1 (user data privacy).

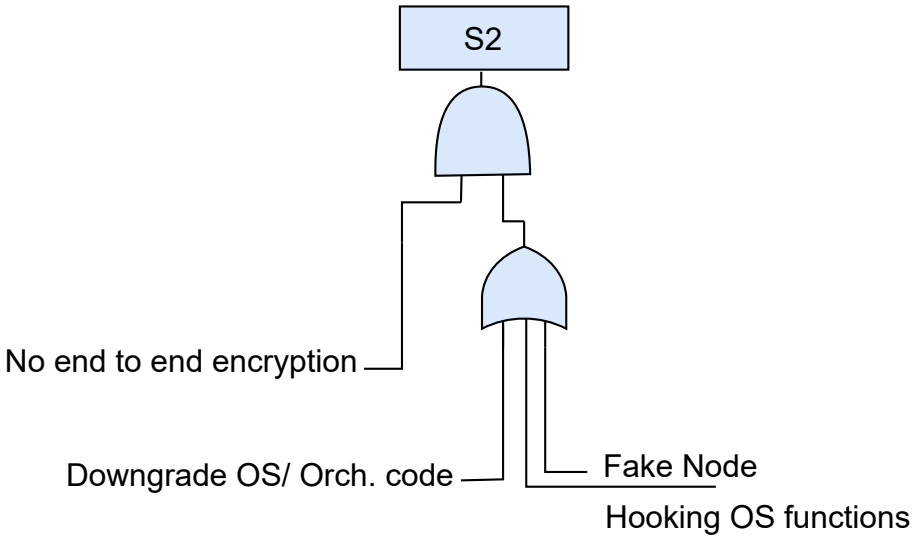

Figure 4.2 – Attack tree for scenario 2 (fog node OS code integrity).

privacy. It means that if one sends or stores the user credential or user data in plain format it would lead to a data privacy breach.

Figure 4.2 illustrates the attack tree of the second scenario. The absence of end to end encryption is a precondition for this attack. If this precondition gets satisfied, the existence of at least one of the three possibilities, which are downgrading the code version, hooking OS functions [150] or using fake nodes can cause this attack.

For the last scenario, the DoS attack can be achieved by several means. As represented in Figure 4.3, the major causes of this attack are: ICMP flood attack, TCP SYN flood attack and, UDP flood attack which we describe them later in this section. Each of the branches of the attack tree corresponds to one of the mentioned causes. As the branches

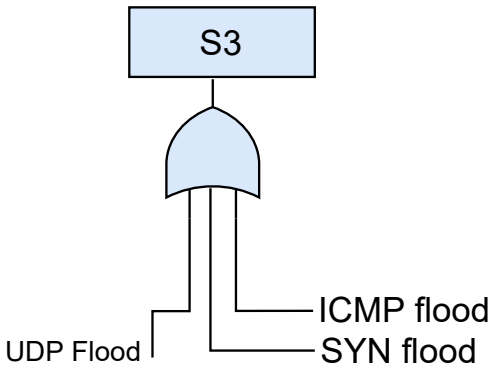

Figure 4.3 – Attack tree for scenario 3 (availability of fog nodes).

are aggregated by an OR gate to the root of the tree, therefore the presence of one of the causes is enough for the attacker to reach her goal.

— ICMP flood attack occurs when the compromised devices send large volumes of ICMP_ECHO_REPLY packets to the victim system. These packets force the victim system to reply and the combination of traffic saturates the bandwidth of the victim's network connection.
— During the SYN flood attack, the attacking system sends SYN request with spoofed source IP address to the victim host. These SYN requests appear to be legitimate. The spoofed address refers to a client system that does not exist. Hence, the final ACK message will never be sent to the victim server system. This results into increased number of half-open connections at the victim side.
— User Datagram Protocol (UDP) Flood attacks. UDP is a connectionless protocol. When data packets are sent via UDP, there is no handshaking required between sender and receiver, and the receiving system will just receive packets which must be processed. A large number of UDP packets sent to a victim system can saturate the network and deplete the available bandwidth for legitimate service requests to the victim system.

In this section, we only used attack trees to demonstrate some possible attack scenarios that the system designer should tackle them. However, the location of the attack in the tree and the type of logical gates that aggregates the attacks provides some hints for the system designer to design effective countermeasures. We utilize attack-defense trees in our analysis of fog computing attacks later in Section 4.6.

4.3 Methodology

The first contribution of this thesis is the introduction of a specific methodology for security analysis of fog computing systems. We study the security of a generic fog platform, irrespective of a specific operating system, orchestrator etc. As we aim our methodology be applicable to a large variety of fog systems we do not include the details about the OS, etc. and try to keep the discussion at an abstract level. The first step of the methodology consists of defining the Target of Evaluation (ToE) as a set of fog nodes, and study which parts of the fog computing system might be valuable for an attacker. This is discussed in Section 4.4.

In the second step, we define the threat model which identifies the capabilities of an attacker. We aim to study a scenario with reasonably powerful attackers, but exclude the most powerful attacks such as one where an attacker may physically detach a fog node from the system and examine its full memory and disk content. This topic is discussed in Section 4.5.

Finally, we analyze the vulnerabilities and possible defenses that our generic fog system may have under the defined attack model. We do so by studying the state of the art of existing attacks, and also by systematically searching for an attack path for an attacker to reach the system's assets. We represent the possible vulnerabilities and their possible countermeasures using attack-defense trees. As we utilize the attack tree modeling in our methodology, the user is able to have a good understanding about the security of her fog system visually. This topic is discussed in Section 4.6.

By using this methodology, the security analyst or the system designer does not need to start from scratch. We analyze a generic fog computing system and provide the result of our analysis in a categorized and systematic manner as a methodology. Thus, anyone who wants to investigate the security of her own fog system or who wants to design a fog system, can use these results as a foundation and take into account the characteristics of her own fog system, such as OS version and network type, and follow the methodology for a complete and non-expensive (in terms of time and money) security analysis.

In Section 4.6.5, we apply our methodology to a concrete fog system to represent how this methodology works through two examples. In these two examples we show the case of threats to OS code integrity and OS data integrity in a fog system. Using the introducing methodology, we also present the required countermeasures for the mentioned vulnerabilities.

4.4 Identification of assets in a fog computing system

We analyze the security of fog computing by identifying the assets of a generic fog computing system and investigating possible threats to these assets. Assets are entities in the system that are valuable for their owner. The owner can be an infrastructure provider, a service provider, an application provider or even an end-user, depending on the asset type. The owner (or a representative of the owner) defines a set of rules for accessing these entities and protects them against any access that does not follow said rules. Any violation of these rules may cause security or functionality problems to the system. Identifying assets of a fog computing system will help us to identify vulnerabilities and, accordingly, required countermeasures.

Fog computing is a multi-dimensional system. It can be seen as a group of single fog nodes, a network of heterogeneous elements or a system that provides different services to its users. We analyze fog computing with the three different perspectives as follows.

— Device level: Each fog node is a single device with its own physical and logical properties.
— System level: fog computing is a networked system with fog nodes as composing elements of this system.
— Service level: fog computing is a system that provides services to its users.

4.4.1 Fog nodes (device level)

Fog computing is a system of fog nodes connected to each other and also to the cloud. Each fog node comprises physical and logical properties and the attacker is interested to access/modify them. In the following, these properties of fog nodes are represented.

Physical assets:

This refers to visible and concrete elements in a fog node that are valuable for their owner. The only physical asset that we identify in a fog node is the memory. Fog nodes have volatile (e.g., RAM) and non-volatile memory (e.g., EEPROM). The memory may contain code, logs and data. We categorize the fog node's memory, either volatile or non-volatile as an asset.

Logical assets:

Each fog node has an Operating System (OS), an orchestrator and executes one or multiple container(s). The OS is the first software layer above the hardware of a fog node.

The orchestrator is a software (distributed or centralized) that logically runs on top of the fog nodes OS and manages resources and workloads of fog nodes in a dynamic circumstance. The operating environment of the orchestrator is dynamic, because fog nodes are autonomous devices that may join and leave the system. Moreover, the IoT device may move, and the workload on a fog node may be transferred to another fog node.

In the software stack of each fog node, the top layer is the application one, which might be containerized. In this subsection, we refer to this layer as container level. Each container belongs to a specific application; multiple containers may be present and active at the same time.

Since the OS and orchestrator code may be open source, the confidentiality of this code is not an asset in general, but ensuring the integrity of OS and orchestrator code is important. The integrity of OS and orchestrator code should be guaranteed to avoid the use of malicious code by the fog node. As the internal data of the OS, orchestrator and container includes security settings, user names or any information related to the running software in the fog nodes, integrity and confidentiality of these data are also considered as assets.

Furthermore, container code and data are specific to each service provider, thus the confidentiality of the container code and data are important and considered as assets. Thus, we identify integrity and confidentiality as properties of the logical assets in the fog nodes.

- Integrity of
  - OS code.
  - OS data.
  - Orchestrator code.
  - Orchestrator data.
  - Container code.
  - Container data.
- Confidentiality of
  - OS data.
  - Orchestrator data.
  - Container code.
  - Container data.

4.4.2 Fog as a networked system (system level)

We now investigate fog computing as a networked system, where each fog node is interacting with other fog nodes or with the cloud layer. In this perspective, the assets are divided into two main groups:

— Fog-cloud communication.

— Fog-fog communication.

In a network perspective, the communication channel between elements, either between fog nodes themselves or the communication channel between a fog node and cloud are considered as assets. As we use the IP address of network elements to identify and access them, we identify them as an asset. The integrity and authenticity of IP addresses of fog nodes are the two properties of this asset.

Fog-to-fog communication:

In fog computing, fog nodes may communicate with each other. In applications in which an IoT device moves, the fog node(s) with which it communicates may also change over time. To ensure a good quality-of-experience for the end-user, the orchestrator has to transfer the latest data and the state of the IoT application to another fog node, which is closer to the new location of the IoT device. In such cases, it is more efficient if the two involved fog nodes communicate with each other directly, rather than using the cloud as the relaying element on the top layer. Thus, we identify two major assets, which are communication channel and IP address of fog nodes.

Communication channel: This refers to the physical or logical transmission medium that enables two or more entities to transfer data with each other [151]. As data are transferred via the communication channel between elements, there is the risk of data manipulation, eavesdropping or even data loss while they are transferred in the channel. Thus, communication channel should ensure data integrity, confidentiality and availability.

Integrity means that the data received at the destination node should be exactly the same as the data sent from the source. The data should be intact, because they will be used as the input for some computation or decision making processes. Therefore, it is important to guarantee the integrity of the data transferred via the channel.

Confidentiality means that data should only be accessible to the source of data and the legitimate, intended destination. In some cases the data transferred via the channel should be kept secret, because they contains sensitive information. For example users of

a remote health care system, may only grant access to their health data to their general practitioner, and not to their neighbors or any arbitrary user.

Availability means that data should be available to the legitimate, intended destination. For example in Industry 4.0 applications, data received from an IoT device will be used for analysis and sending commands to other IoT devices in a production line of the factory. If the data availability of one IoT device were violated, it might affect the correct behavior of the system.

Address of entities: This assets ensures that communicating elements are able to find each other. A fog node contains its end-user information, so it should know whom it is communicating to and avoid sending this information to an unknown network element. Thus, the integrity of IP addresses in the network are assets.

Fog-to-cloud communication:
Fog computing can also be seen an extension of cloud computing [9]. It requires the presence of cloud for further processing or storage. This implies fog-to-cloud communication. In the networked system perspective on fog computing, we identify two main assets:

Communication channel between cloud and fog: The communication channel between fog and cloud should ensure data integrity, confidentiality and availability as described in Section 4.4.2.

Address of entities: This asset enforces the fact that communicating elements should be able to find and recognize each other.

4.4.3 Fog-provisioned services (service level)

As fog computing is meant to provide services to its users, the properties that are instrumental to ensure the provision of fog services are recognized as assets. Service availability, user's privacy and user's data are the assets in the service-level perspective.

Service availability:
The availability of the service provided by the fog infrastructure is an important property of the system. If the user cannot be confident about the availability of the service, this would affect the overall accountability of the system. Thus, we identify service availability as an asset.

User's privacy:

One of the important requirements of services is to protect the privacy of their users, in particular in the IoT context. Thus, privacy is another asset in the system-level perspective. We study privacy in three main categories: privacy of location, privacy of identity and privacy of other personal data related to the user. We define user's privacy as the ability of keeping any information related to each individual private, except for agreed use cases in which said data are disclosed to agreed and legitimate data destinations.

Location privacy: The user location is an important type of data that should be kept private. Therefore, location privacy is clearly an asset in a fog computing system.

Identity privacy (anonymity): When an individual uses a service, his identity should be kept private. In other words, there should be no mapping between the user's identity and the available data in the service. Thus, we categorize identity privacy as another asset.

Other personal data: Privacy of any other data that belongs to a specific user and which is not in the two previous categories resides in this group.

User's data:

The data related to the user and stored in a fog node or accessible by a fog service is an asset. The two main properties of user's data asset are integrity and confidentiality.

User's data integrity: In a fog computing service, users' data should be intact. We identify user's data integrity as a property of the user's data asset.

User's data confidentiality: Individual's data should not be revealed to any entity other than the legitimate, agreed service that the data is meant to be sent to. In this regard, user's data confidentiality is a property of the user's data asset. It should be mentioned that this category of asset is different than the other personal data described in Section 4.4.3. The reason is that, in the privacy context, even an encrypted data, which meets requirements of confidentiality, can still be used to exploit an individual's usage pattern of a service and therefore violate user's privacy.

4.5 Threat modeling

The power of the attacker depends on where in the architecture the attack is set up. We can consider three different parts in the architecture:

— The field level, where IoT devices and smart objects reside.
— The fog level, which controls multiple devices close to the edge of the network. Note that the fog level might be the first security layer in an IoT application, especially when resource-constrained devices are deployed in the field.
— The enterprise and platform level, which reside at the core and where application- and platform-level security measures are applicable.

This architecture implies two attacker models to be considered, the `insider` for the field level and the `outsider` for fog and platform levels.

The `insider` model assumes that the device is in the wild and thus, can be in the hand of the attacker, e.g. a stolen smart watch, a temperature sensor, a home gateway. The adversary can control the device by physically replacing the entire node, or tampering with the hardware of the device. If a node is compromised the important assets can be exposed to the adversary. The adversary can also copy the important assets associated with the captured node to a malicious node, and then fake the malicious node as an authorized node to connect to the IoT system.

Considering this model, the device must defend itself against a large variety of means to extract the assets from the device. If the device can be disconnected from the network, then there is no means to observe remotely at the fog level that the device is being tampered. Some devices can have built-in security but very few can be protected against hardware attacks to retrieve cryptographic materials [152]. From the security and privacy perspective, IoT currently presents two main inherent weaknesses: first, security is often not considered at the design phase and second, currently no solution is offering a complete end-to-end security approach for said kind of devices.

Thus, at the field level, we need to consider in a first step that IoT devices are trusted. This implies that the field level must consider only the `outsider` model. As a consequence, the communication between IoT devices and fog nodes can be considered secure as long as they are encrypted and the keys are correctly stored and manipulated.

The `outsider` model has only a logical access to the devices including the network. The attacker can perform a set of attack remotely as for example:

Malicious code injection attacks The adversary can control a fog node or a device in IoT by injecting malicious code into the memory of the node or device, which is denoted as the malicious code injection attack. The malicious code can not only perform specific functions, but can also grant the adversary access into the IoT system, and even gain the full control of the IoT system.

Side channel attacks Several attacks have been set up using software like cache attacks, Rowhammer, ClockScrew and so on. This is a consequence of the ability of uploading code in the device (node or IoT). The consequences range from extraction of cryptographic materials to privilege escalation.

Eavesdropping Attackers can listen on communication channels to capture transmitting packets and read the content or the meta data of the protocol and they can infer information about the device or the node. Encryption is a partial solution as the node must first identify itself, such that the corresponding key can be associated for a ciphered communication. This points out a privacy issue, as a node or a device can be then easily tracked.

Denial-of-service An attacker can flood the target node with unnecessary requests to make it unavailable to the users. A fog node is vulnerable to denial-of-service attacks due to its limited resources compared with the cloud.

Many attacks of different nature can be set up by an attacker even with only the `outsider` model, considering that she can only access remotely to the device. Nevertheless, the ability to upload program or downgrade firmware is an important attack surface. Thus, these operations must be strictly controlled through a strong authentication mechanism.

The mentioned attacks are examples to show potential attacks in fog computing domain. We refer the readers to [153], for more examples of attacks in this domain with a classification of attacks and countermeasures.

4.6 Analysis of vulnerabilities and possible countermeasures

We now analyze threats to the fog computing assets that are identified in Section 4.4. The goal is to bring to the fore the possible threats that can cause security problems

for a fog computing system. It is worth mentioning that threats are potential attacks that can happen to a system because of the existence of unsolved vulnerabilities. Thus, in this section we utilize both terms whenever proper. The reader should note that our vulnerability analysis is generic. It is not specific to a given fog computing implementation, because fog computing systems may differ widely. For example, just the type of fog node OS provides lots of possible fog computing configuration. However, we apply our methodology to an archetypal fog computing system to show how this method works in practice. We follow the three different perspectives used in assets identification in Section 4.4 for vulnerabilities analysis which were: device level, system level and service level.

4.6.1 Fog as a device

In the device level, we analyze vulnerabilities to assets of fog nodes, where fog nodes are regarded as individual elements with physical and logical properties.

Vulnerabilities of physical assets:

According to our threat model, the attacker is not able to have the IoT device or fog node in her hand. She can only perform attacks remotely or logically. However, it is worth mentioning that, although we omit this possibility in our analysis due to our attack model, physical control of computation nodes is not an impossible attack vector. According to a report by the European Union Agency for Network and Information Security (ENISA), the edge of the network is more prone to attacks because of its easier physical accessibility to the attackers [154].

In the following, we list possible threats to the node's memory, which is the only identified physical asset in the fog node. Memory is a physical entity that we treat it as an individual element and we do not zoom-in into it to reach to its data. As memory contains data, there are overlaps between memory threats and data threats. Memory read, memory write or memory copy (bulk memory read) are three main categories for the threats that can happen by exploiting memory vulnerabilities.

Rowhammer attack: This attack exploits the compact architecture of modern dynamic random-access memory (DRAM), that has the possibility to leak their electrical charges to the nearby memory cells and change their content [155]. The attacker can use a software and stress memory rows in order to cause electrical leakage and affect other

memory cells. As this attack just needs software and can have fatal consequences, such as privilege escalation, it is an strong attack vector for DRAM memory type.

Disturbing memory by electromagnetic emissions: As we do not assume a strong attacker with access to expensive equipment, we skip those attacks with access to specific bits of memory and possible ability to change them. We just assume an attacker with inexpensive equipment and possible remote access to the device. Thus, as mentioned in Section 4.5, the attacker is not able to detach the device from the system and take it to a laboratory to use imaging or micro-probing. techniques. Instead, our attacker is able to use electromagnetic emissions to disturb arbitrary bits of the memory. She is not able to modify specific memory bits, but only random locations of memory can be manipulated. Having integrity check can cause detection of attack but it can not protect the memory against it. Shields against electromagnetic emissions for the memory can prevent such attacks.

Access to the memory using the fog ports: If the attacker is able to use the update port or other ports of the fog node to access the memory, read it, change it or clone it, she will be able to exploit the accessed data for other types of attacks, e.g., building fake fog nodes. A countermeasure against this attack is to perform authentication before granting any access to the memory.

Vulnerabilities of logical assets:

As mentioned in Section 4.4, the identified logical assets are code and data. The code can belong to OS, orchestrator or container. Each of the said codes has its corresponding data. In this section, we investigate possible vulnerabilities for the two properties of both code and data, which are integrity and confidentiality.

Vulnerabilities of OS code integrity: If the OS code of the fog node is not an authorized version published by a well-known provider, the code integrity of the OS is already compromised. Installation of applications or of an orchestrator from unauthorized sources can also threaten the OS code integrity, because applications or the orchestrator can contain Trojans with the ability to change the OS code. Similar issue can be caused by the use of malicious updates.

A countermeasure to this vulnerability is to use signed codes by well-known code providers. The author admit this countermeasure may be idealistic and there are situa-

tions in which we are forced to use unauthorized code, because there is no certified code available. Secure boot is another mechanism for detection of unauthorized changes in the OS code.

Vulnerabilities of OS data integrity: The OS data of the fog node should not be changed out of the set of defined access rules. OS data includes a huge range of valuable information, such as return addresses in stack, stack pointer, indicators of frames in stack, etc., which are only examples of OS data in the stack part of the OS. It is not easy to determine how much OS data exist which should be protected. One solution is that the security analyst should be aware of the state of the art in the attacks to be able to, at least, check if her system is vulnerable against the most popular and known attacks.

Vulnerabilities of orchestrator code integrity: The same as for the integrity of OS code.

Vulnerabilities of orchestrator data integrity: The same as for the integrity of OS data.

Vulnerabilities of container code integrity: The same reasoning as for the integrity of OS code applies to the container code. As container code also includes interfaces and APIs, which will be accessible out of the fog node, it can be target of various attacks. To defeat such attacks, it is necessary that the APIs and interfaces, embed access control and authentication methods to control access to the resources of the fog node.

Vulnerabilities of container data integrity: As fog nodes may support multi-tenancy, data of different containers should be protected against access by code running in other containers active on the same fog node. In addition to this, other vulnerabilities as for the integrity of OS data applies for the container data as well.

Vulnerabilities of OS data confidentiality: Some parts of the OS data such as the OS keys should be kept secret. If an attacker gets access to this data, she will be able to change it and control the fog node or clone this data and create a cloned, fake fog node. As a countermeasure, these sensitive data should be encrypted by a key which is kept in a hidden memory of the fog node.

Vulnerabilities of confidentiality of orchestrator data: The same as for the OS data confidentiality.

Vulnerabilities of confidentiality of container code: As the application running on the fog node containers may not be necessarily open source and owned by a company, it is important to protect it against reverse engineering.

Vulnerabilities of confidentiality of container data: As mentioned in the integrity of container data, the multi-tenancy property of the fog nodes reveals a vulnerability related to container data access. In addition to this, other vulnerabilities as for the confidentiality of OS data applies for the container data as well.

In Section 4.6.5, the vulnerabilities to OS data and OS code will be discussed in more detail as examples of application of our methodology.

4.6.2 Fog as a system

In this section, vulnerabilities against fog computing from a system (network) perspective will be analyzed. As we categorized the assets of fog computing in the system level perspective into three different levels, we keep the same for the vulnerability analysis.

Vulnerabilities of device-fog communication:

According to our `outsider` attack model in Section 4.5, we assume the communication part between Device and fog is secure.

Vulnerabilities of fog-fog communication:

In this section, the vulnerabilities to the communication channel between the fog nodes are discussed. We put these vulnerabilities in two main categories: the communication channel and the address of entities.

Vulnerabilities of communication channel: The communication channel between fog nodes has the following vulnerabilities:

— Modification attack: The message transferred between two fog nodes can be modified unintentionally or intentionally by a malicious entity while in transit. Countermeasure: by using Hypertext Transfer Protocol Secure (HTTPS) the integrity of the communication is protected.

— Eavesdropping: The message can be eavesdropped if it is not encrypted for the recipient of the message. HTTPS also protects confidentiality of the message in the communication channel.
— Message removal: The communication channel may be unavailable because of the high volume of request sent by a malicious entity in the network. Thus, the channel is congested by arbitrary data and will not be available for the real requests.

Fake addresses attack: While fog nodes communicate with each other, they exchange data which includes user data. If the two communicating entities do not control the address which is sending data to, this data may be sent to a rogue fog node. Thus, the security of user data is threatened. Man-in-the-middle attack, rogue fog node attack and phishing are attacks that can exploit this vulnerability in fog system. Thus, it is needed that fog nodes perform authentication between each other before exchange of information. This authentication can be managed by a central element such as cloud.

Vulnerabilities of fog-cloud communication:
The type of vulnerabilities for fog-cloud communication is similar to the fog-fog communication. Thus, we put these vulnerabilities into two main categories: the communication channel vulnerabilities and vulnerabilities to the address of entities, as per the previous section.

4.6.3 Fog as a service

In this section, we analyze vulnerabilities to fog computing from a service perspective.

Vulnerabilities of service availability:
Fog computing provides services to IoT devices and to the cloud. Some of the applications that are using fog service could be sensitive to availability, e.g., health monitoring applications. In this regard, any attack that pushes unnecessary requests to the fog services will threaten the service availability for the real requests. The IoT devices, fog nodes or any other network element are capable of sending fake requests.

Vulnerabilities of user privacy:
As fog nodes usually provide service for the users in proximity, knowing the location of the fog node will reveal the approximate location of the IoT devices connected to it and hence, of the user(s). Thus, it is important to keep the location of the fog node private.

Another threat to user's privacy can be caused by installing applications on the fog node, which use hard-coded credentials or publicly available credentials. It can mostly happen with open-source applications available on the Internet. As the users may not change the hard-coded credentials, a lot of applications can have similar credentials and can become an easy target for attackers.

There are also cases where the user wants to stay anonymous while using a service. For example, users of a service that helps people with drug addiction may want to remain anonymous. On the other hand, if the service becomes part of the public health care system, it may need to access the real identity of the users. Thus, the user provides his identity to the system to register into this service, but the service should guarantee that user identity will not be shared or used for other purposes. If (part of) the user data are stored on fog nodes and get stolen by an adversary, there is the possibility of privacy threat, unless the user names are protected by some encryption scheme. This issue can happen at both IoT and cloud levels. In this example, the problem is not related to the user's credentials, but here even the user name is an asset to be protected. If a service uses on-premise fog nodes and the fog nodes are publicly known for belonging to a special service, even tracking data from user's device to that fog node would be enough to threaten the user's privacy. In this case, the problem is not about the data that the user sends, but even the existence of a data flow between IoT devices and the fog nodes could threaten the user's privacy. Moreover, if an attacker is able to analyze user's data flow pattern with fog node or cloud, he can have some predictions about user's habits.

Vulnerabilities of user data:

As the two properties of user's data assets are user's data integrity and user's data confidentiality, in this section, we discuss possible vulnerabilities to them.

Due to the fact that malicious OS, orchestrator or fog applications can threaten the integrity or the confidentiality of user's data, it is important to keep them secure. In addition, memory vulnerabilities could also lead to threats against user's data confidentiality and integrity.

A generic countermeasure to keep confidentiality and integrity of user's data is to perform authentication before granting access to user's data.

The input data in the form of user's data might be used to perform malicious operations. A family of attacks collectively referred to as code injection, covers various types of attacks such as cross site scripting (XSS) [156] and SQL injection [157]. These attacks

use input data as a gateway to attack a system. In order to mitigate such attacks, the input data should be filtered and checked.

4.6.4 Assets and Vulnerabilities

In this section, we summarize all the discussed assets and vulnerabilities in Table 4.1 for more convenient use. This table represents the discussed materials in three levels which are device level, system level and service level.

Table 4.1 – Identified assets and threats for a generic fog computing system.

	Assets		**Threats**
Device Level	Fog node memory		-Access to the memory using the Fog ports -Disturbing memory by electromagnetic emissions -Rowhammer attack
	Integrity of	OS code	-Usage of unauthorized code -Downgrading code -Exposing APIs with no access control
		OS data	
		Orchestrator code	
		Orchestrator data	
		Container code	
		Container data	
	Confidentiality of	OS data	-Root access grant -Hooking OS/Orch. functions
		Orchestrator data	
		Container code	
		Container data	
System Level	Communication Channel	Integrity of Comm. Channel	-Modification attack
		Confidentiality of Comm. Channel	-Eavesdropping
		Availability of Comm. Channel	-Message removal attack
	Address of entities		-Man in the middle attack -Rogue Fog node -Phishing
Service level	Service availability		-Dos or DDos attacks
	Location privacy		-Location attack -Credential attack -Usage pattern attack
	Identity privacy		
	Other personal data privacy		
	User's data integrity		-Code injection to access users data
	User's data confidentiality		

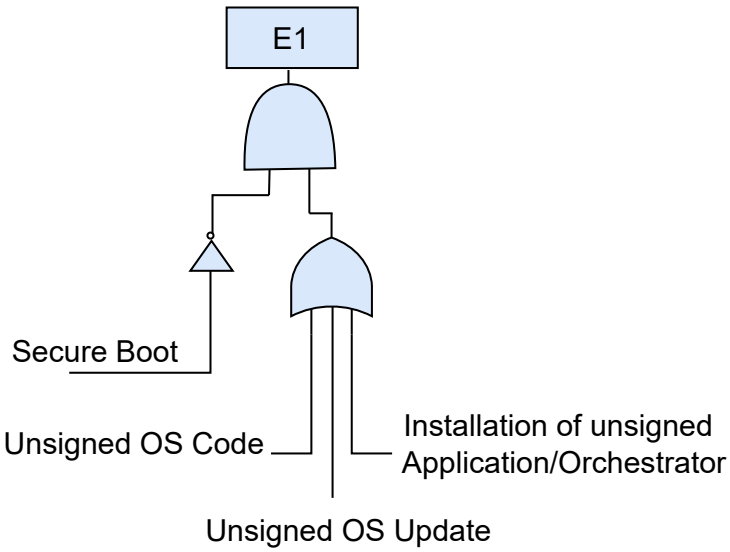

Figure 4.4 – Attack-defense tree for OS code integrity.

4.6.5 Examples of applying our methodology on fog platforms

In order to explain how our methodology can be applied to a concrete fog computing system, we hereby provide two examples:

— Example 1 (E1): Fog node OS code integrity. As defined in Section 4.4.1, OS code is identified as an asset of the fog node. Later, we defined our threat model, which describes the attacker and our assumptions about the environment. In the vulnerability analysis part, we investigate possible methods that could cause attack. In Figure 4.4, the attack-defense tree shows different paths able to cause an E1 attack. We have also presented a possible countermeasure in the tree, which is secure boot. In secure boot, the hash of all or some part of the OS code is compared to the hash of the OS code installed on the fog node during the boot. Thus, it can detect changes in the OS code.

— Example 2 (E2): As OS data includes various data types, we just choose a function return address as an example of the OS data, which is an asset. In OS, when a function is called, the return address to the caller function is written in the OS stack. So the OS knows from which line of code it should continue its execution after finishing the execution of the called function. If an attacker is able to change the return address to another address, in order to skip some lines of code, which could contain access control checking, it can cause access to resources beyond the defined rules and attack the system [158]. There are various techniques to change

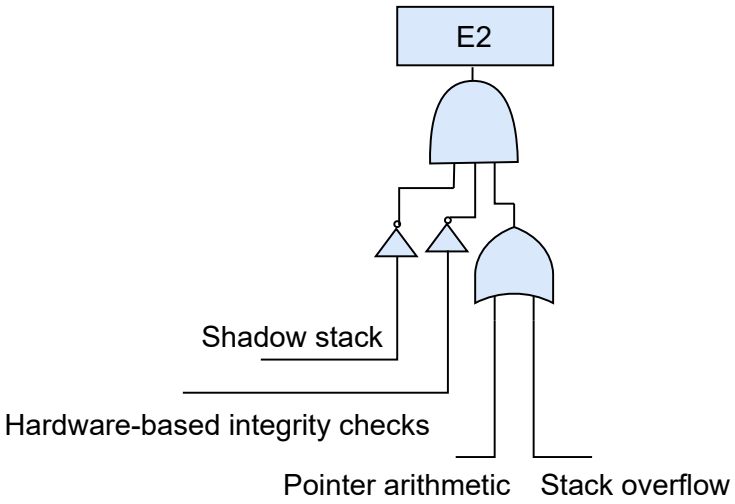

Figure 4.5 – Attack-defense tree for OS Data integrity (Function return address).

the return address of a function. Two known techniques are buffer overflow and pointer arithmetic. In order to defend against these attacks, a solution is to build another stack, called shadow stack. A shadow stack, keeps the state of the stack, including the return addresses, and is used to compare the data inside the stack with it [159]. Although using Canary is another possible countermeasure, we do not include it in the attack tree, because shadow stack covers it and represents a more powerful defense than Canary mechanism [160]. There are also hardware-based solutions that protect the return address against attacks. In Figure 4.5, the attack-defense tree represents attack to one of the OS data types which is function return address.

4.7 Conclusion

This chapter presents an analytical approach to the security of fog computing systems. We commenced with the Common Criteria methodology, and tailored it for analyzing the security of fog computing systems. In our methodology, we identified assets of fog computing and clustered them into device level, system level and service level. For each perspective, we pointed out possible vulnerabilities that the system could have and discussed possible solutions. In order to shape our analysis, we utilized attack-defense-tree and represented vulnerabilities with possible countermeasures.

One of the important identified assets in our study of fog computing security is the user's personal data. As the fog computing layer is located in close vicinity of the end-user layer, it often has more access to the personal data than the cloud layer. Therefore, applications running on the fog layer require precise investigation to assure if they treat personal data in a privacy-preserving manner. Although applications normally claim they handle personal data according to their privacy policy, this claim needs to be verified. Our study highlights the need of monitoring the behavior of applications to which the user may send personal information. This topic is the main discussion of the next chapter of the thesis.

Chapter 5

Automated privacy compliance checking of applications in Fog environments

5.1 Introduction

As online applications blend ever more with our daily lives, users are becoming increasingly concerned about the respect (or lack thereof) of their privacy. Enforcing privacy is governed by national and international regulations [14], and many applications expose a privacy policy describing how they intend to gather, use, disclose, and manage their users' personal data.

However, a recent study of the top 20 most popular Android health applications on Google Play showed that *every studied application* failed to comply with at least part of its own privacy policy terms [21]. Similarly, personal assistants such as Google Assistant and Amazon Alexa have access to very private information through audio recordings in people's homes, access to their emails and online documents, shopping history, and so on. Although these applications and their respective privacy policies were developed by major IT corporations claiming to implement the highest quality standards, several incidents have demonstrated violations of the announced privacy policies [161], [162]. We expect that similar concerns will emerge in future IoT applications, which are normally deployed in cloud/fog platforms and potentially have access to significant parts of their users' personal data. The compliance of these applications to their privacy policy should therefore not be taken for granted but rather be the subject of "Trust, But Verify" strategies.

Many applications which have access to their users' personal data are hosted in cloud computing or fog computing platforms [6], [13]. For example, Amazon Alexa performs parts of its tasks on the cloud and hence sends users personal data to the Amazon's cloud platforms. We claim in this thesis that privacy compliance checking should not be

done manually (which is error-prone and time-consuming) but that the platform which hosts applications constitutes a strategic location where the actual behavior of these applications may be automatically monitored and verified against the published privacy policies. First, the platform constitutes a third party which is already trusted explicitly or implicitly by the application developers and the end-users. Second, they have access to many useful information about the applications, such as their CPU usage and network traffic which may already use them for performance purposes. Note that, we aim at designing a privacy monitoring platform that is as little intrusive as possible, therefore we only monitor external signals of the application such as its network traffic.

We assume that the application provider and the end-user trust the platform. It means that the application provider trusts that the platform runs the application correctly. Conversely, the end-user trusts that the platform will not leak her data and will handle personal data according to the platform's privacy policy. Moreover, both the application provider and the end-user trust that the platform correctly identifies privacy violations according to the published precision and recall for the system. Finally, we assume that the application does not actively try to evade monitoring. Malicious applications that develop strategies to mislead the monitoring system are out of the scope of this thesis.

We argue that automated privacy compliance checking is feasible, and outline a research roadmap towards the development of such systems. The problem of automated privacy compliance checking can be split in three issues. First, privacy policies are usually written by lawyers in natural language ; one thus needs to analyze them to extract a machine-readable set of *privacy principles.* A number of research works already propose techniques for such privacy policy analysis [137], [138], and more results can be expected in the future in this domain. Second, the platform must define and monitor a number of *application signals* (e.g., applications' CPU usage profile, network traffic) that may be used to characterize application behavior. This monitoring should be non-intrusive while remaining agnostic to the application's implementation and programming language. Finally, one needs to interpret the application signals to determine whether they match the acceptable behavior specified in the privacy policy.

In this thesis, we propose a general model for automated privacy compliance checking in cloud and fog environments, and apply it to exploit network traffic signals to verify the "data sharing with third party" privacy principle. This principle, which defines which data types an application is allowed or disallowed to share with specific third parties, is

at the heart of the European GDPR regulation [14]. Other combinations of application signals and privacy principles are out of the scope of this thesis.

We use the cloud/fog platform that hosts the application to monitor the application behavior because the platform has access to many useful information about the applications, such as their CPU usage and network traffic and may already use them for performance purposes. In our experimental setup, we use a fog computing platform, because of the critical location of fog platforms which normally provides them with greater access to the users' data. However the proposed model is reproducible in cloud platforms as well.

We show how applications' network traffic may be monitored in a non-intrusive way, while clearly distinguishing the traffic produced by multiple co-located applications and differentiating internal application traffic (between multiple instances of the same application) from external traffic with external entities.

We assume that application traffic is encrypted, and therefore do not aim to access the communication's payload. But we show that machine learning techniques can analyze the packet headers and reliably classify different transmitted data types and communication behavior. Our empirical validations, involving twelve applications belonging to four classes of generated data types, obtain an accuracy level of 86% in classification of application's network traffic to the correct data type. This work leads the way toward unintrusive yet highly-accurate identification of application behavior and automatic privacy compliance checking.

This chapter is organized as follows. Section 5.2 describes our privacy compliance checking model. Section 5.3 presents the network traffic capture procedure. Section 5.4 describes the analysis of the captured network traffic. In Section 5.5, we present the experimental results. Section 5.6 discusses the system costs. Finally, Section 5.7 concludes.

5.2 System model

An automated application privacy compliance checking system essentially needs to derive a set of privacy principles from the application's privacy policy, and to watch the application as a set of privacy-oriented behaviors. Later, these two sets may be compared to report possible differences between the claims made in the privacy policy and the observed application behavior. This comparison may be performed offline or online. In both cases, the system needs to monitor application signals and based on the application signals deduce the application behavior. If a non-compliant behavior is detected, the

application signals can be saved to be referred to as an evidence of the application's non privacy-compliant behavior.

Figure 5.1 presents the flowchart of our proposed model. An automated privacy compliance checking system should monitor the behavior of the application using the chosen application signal(s). We also use the privacy policy as our ground truth for the application behavior. Applications which do not specify a privacy policy, and the investigation of whether the privacy policy is well designed and correctly protects the users' privacy rights, are out of the scope of our work. The remainder of this chapter focuses on the highlighted parts of the figure.

We consider that monitoring applications from their hosting platform provides enough information to understand the applications' privacy-oriented behaviors. We utilize the fog environment for our experiments. Our fog platform is constituted of a set of ten connected Raspberry Pis which represents an easily accessible form of a platform. Moreover, we utilize the network traffic as one application signal to demonstrate the viability of our claim.

A fog computing platform is comprised of many nodes located close to their end-users and that are coordinated with each other through an orchestrator such as Kubernetes [53]. These nodes host containerized server-side fog applications. Containerization makes them largely platform-independent as each application contains all the necessary dependencies for its execution. These fog applications receive data from the user layer which possibly include personal data.

As monitoring the application itself (e.g., scanning its memory state) would potentially be very intrusive and application-dependent, we prefer monitoring only external signals such as the network traffic produced and received by the application. Referring to the flowchart in Figure 5.1, our model requires one to extract privacy principles from the privacy policy, monitor the chosen application signal(s), extract application's behaviors from the signal(s), and compare these behaviors to the application's privacy policy terms. Figure 5.2 presents a high-level view of our implementation for the two highlighted steps from Figure 5.1.

5.3 Network traffic capture

Monitoring the network traffic of a fog or cloud application is not a trivial task. First, the same server are typically used to host multiple independent applications. One

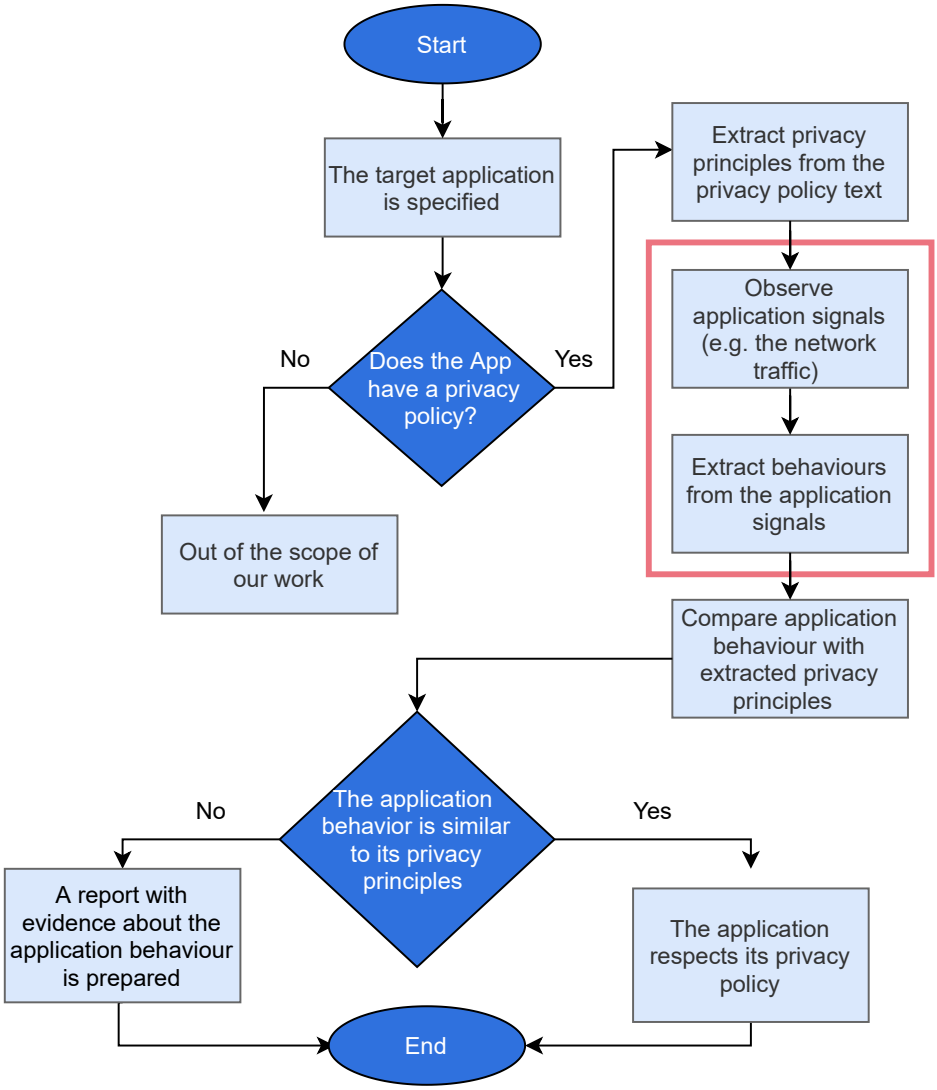

Figure 5.1 – Flowchart of our proposed privacy verification model.

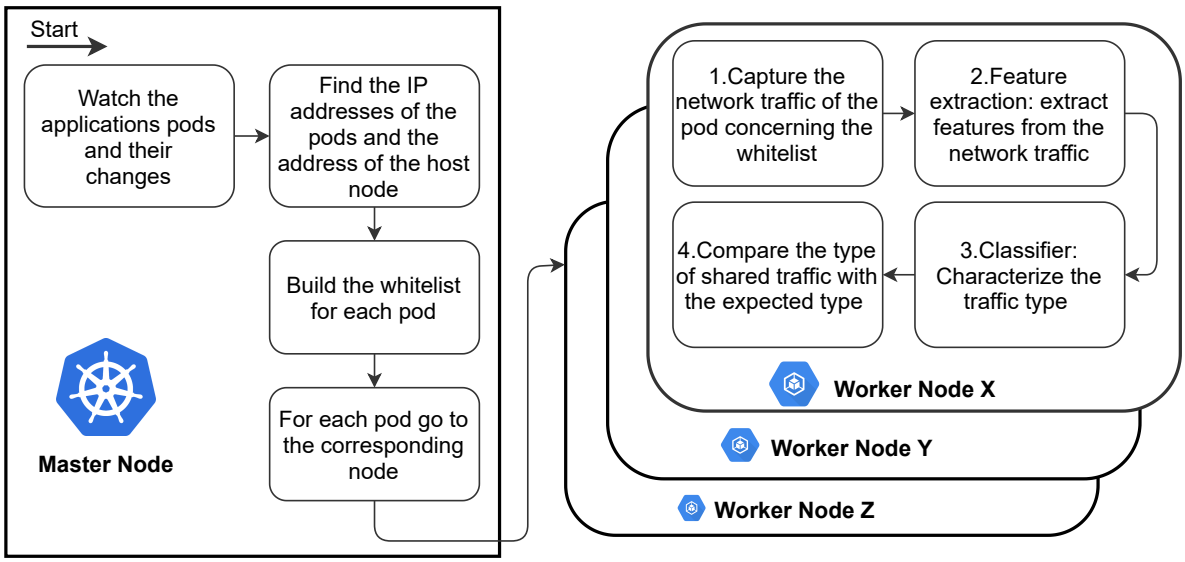

Figure 5.2 – High-level view of our system implementation.

therefore needs to distinguish the traffic exchanged by these different applications to attribute the observed behaviors to the correct applications. Second, any application may be distributed across multiple nodes which communicate with one another. Arguably this internal network traffic should be handled differently than the traffic exchanged with the rest of the world. Finally, even local communication between different containers located on the same node may constitute "external" communication if it involves two different applications. One therefore needs to carefully understand the way cloud/fog applications communicate to attribute network traffic traces to the correct application and internal/external category. Note that, since we assume that all traffic is encrypted, we are interested in capturing only the packet's header data such as the source and destination IP addresses, packet length and inter-packet arrival times.

We assume orchestration is done using Kubernetes (K8s). Although Kubernetes was initially designed for cluster or cloud scenarios, it is now being extended to make it suitable for fog computing scenarios [56], [57].

In this thesis, we define an application as a set of K8s services. Thus the privacy principles for any application must be enforced to the relevant set of applications and services, defined as groups of pods which share the same namespace and application label.

Capturing an application's traffic therefore requires one to identify which K8s pod belonging to which service has issued or received any network packet. To capture the network traffic of any pod, a number of tools such as ksniff [163] are being proposed. However, these tools rely on a tcpdump binary file inserted inside the pod itself that is being used to capture the pod's traffic. Although this indeed captures the specific network traffic handled by the pod, installing an additional binary inside the pod is intrusive as the application providers may not accept any modification to their services. Second, monitoring network traffic from the inside of a pod opens the possibility for the application to disrupt the way traffic is being captured, and therefore possibly to evade the monitoring measures. We therefore prefer transparently monitoring the pod's network traffic from the platform itself.

To do this, we rely on the way Kubernetes networking is organized. We list all network devices on the node and on the pod, then correlate device numbers between the two listings to find the network interface of the target container in the pod [164].

The network traffic of the application's pods can be classified in four categories:

1. Incoming from user layer or other applications' pods;
2. Incoming from other pods of the same application;
3. Outgoing to other applications' pods or to the cloud;
4. Outgoing to the other pods of the same application.

As we are interested in monitoring user personal data that may be disclosed to third parties by the application, we are mostly interested in the third category. We capture this network traffic using the pseudo-code from Algorithm 1. As illustrated in Figure 5.3, we first get the list of private IP addresses of the pods that belong to the application and build a whitelist from these IP addresses. The whitelist enables us to discard the internal traffic that is exchanged between multiple pods of the same application. When capturing a pod's network traffic we can thus obtain only the outgoing traffic being sent to pods of other applications or to the cloud.

Depending on the application's privacy policy, the existence of outgoing traffic toward third parties may be allowed or disallowed. If it is disallowed, then the simple existence of such traffic is sufficient reason to raise an alert. Otherwise, it is useful to *analyze* this outgoing traffic to further understand which types of data are being shared and in which conditions.

Algorithm 1: Capture outgoing packets to other applications.

```
1  get the appName and appNamespace from the input
2  Initialize Whitelist to {}
3  for service in app=appName and namespace=appNamespace do
4  |   for pod in service do
5  |   |   podIP=FindPodIP(pod)
6  |   |   Whitelist.Append(podIP)
7  |   end
8  end
9  for podIP in Whitelist do
10 |   VETH=FindVETH(podIP)
11 |   tcpdump from -interface VETH –direction=Out and 'not in (dst Whitelist)' STORE
   |     OutgoingToOthers.txt
12 end
```

5.4 Network traffic analysis

The network traffic of an application carries useful information about applications' privacy-oriented behaviors. We exploit this traffic in particular to identify the data sharing behavior of the application. If the running Pod shares data with the pods of other applications, then we want to characterize the type of data it is sharing with others. We conduct this analysis using supervised machine learning (ML) techniques.

To assess the ability of ML techniques to extract actionable application behavior from captured network traces, we select a set of applications, containerize them, and collect multiple traffic samples from each of them. Each traffic sample is associated with the type of data that the application generates. These labeled samples enable us to use supervised learning techniques where we randomly select a subset of the samples to train the ML model, and use the remaining (disjoint) subset of samples to test the model.

Feature extraction

To enable an ML model to process traffic samples, we randomly split the samples of our dataset into two disjoint subsets where 80% of the samples are used for training, and the remaining 20% are used for testing the model. We first train the model using the features of a traffic sample, together with a label. We then test the model by giving the features of a traffic sample, and check if the model can accurately output the correct traffic type label.

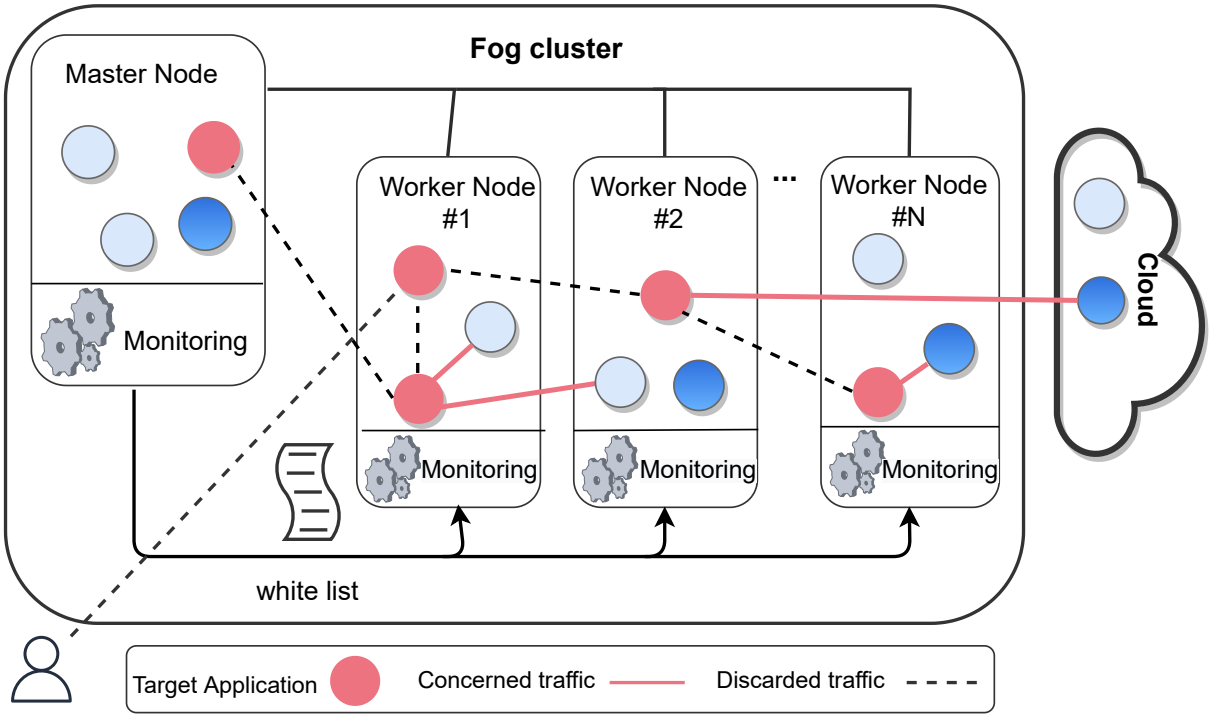

Figure 5.3 – Whitelisting some of the network traffic in the platform. Pods of the target application are specified with red color, other applications with blue.

To define a model which represents our data, first we need to extract number of *features* from each sample. These features are the inputs to the ML model. The features should in principle not be dependent to the properties of their environment, as such dependencies may cause unstable behaviors when the model receives samples which are captured from different environments [165]. We therefore use statistical patterns from these samples which are otherwise not identifiable and not related to the network properties that the application is using.

Table 5.1 presents nine features we extract from the network traffic samples. The first packets are typically used for connection establishment. Therefore, some features extract statistics from the first ten packets to characterize the initialization phase of the application and others extract statistics from the next 200 packets, and hence capture the running behavior of the application. We discuss in the next section how we further refine this initial list to select the most relevant features in our models.

Table 5.1 – Features of the network traffic samples.

Feature Name	Description
prtcl	the most used protocol in the first 10 packets
deltatime_mean_First10	mean value of the delta time between packets in the first 10 packets
deltatime_std_First10	standard deviation of the delta time between packets in the first 10 packets
pcklength_mean_First10	mean value of the packets length for the first 10 packets
pcklength_std_First10	standard deviation of the packets length for the first 10 packets
deltatime_mean_Next200	mean value of the delta time between packets in the next 200 packets
deltatime_std_Next200	standard deviation of the delta time between packets in the next 200 packets
pcklength_mean_Next200	mean value of the packets length for the next 200 packets
pcklength_std_Next200	standard deviation of the packets length for the next 200 packets

Application data type detection

When an application is being monitored, we capture traffic samples from this applications which consist of the IP and TCP/UDP headers of the first 210 packets produced by the application.

Identifying the data types that are being shared using only the packet headers of encrypted traffic is obviously a very challenging task. We however show that it can be done at least partly. We therefore split the studied application into four classes: video streaming, audio streaming, file download, and others. In the next section of this thesis we show that an ML algorithm trained with samples belonging to a subset of all applications can identify the data types shared by entirely new applications which were not used during the training phase.

5.5 Evaluation

We now examine the validity of our claims by showing the performance results of our classifier aimed at identifying the type of shared data in the encrypted network traffic.

5.5.1 Data set

To our best knowledge, no public data set contains labeled network traffic samples to allow us to characterize the applications with their data types. For example, a good available dataset [146] for encrypted traffic data type classification has the following classes: Chat, Email, File Transfer, streaming, Torrent, VoIP, VPN:Chat, VPN:File transfer, VPN:Email, VPN:streaming, VPN:Torrent, VPN:VoIP. These classes of data type are far from the requirements stemming from privacy policies. We therefore created our own data set using the isolated network traffic of different real-world applications according to privacy policy requirements.

We selected twelve different applications and containerized each one separately. We started each container repeatedly and captured the isolated network traffic of the container. As we knew what activities the application was engaged in during capture time, we labeled the data types accordingly.

We created and publicly released a dataset with almost balanced number of samples in each group [166]. Due to the fact that most of the applications encrypt their network traffic, we do not capture the payload of the network packets and we only capture the packets' headers.

Table 5.2 illustrates the selected applications with their respective data types. Each sample contains the features computed out of the first 210 network packets generated by the application after being started.

5.5.2 Evaluation metrics

Our classifier is in fact a multi-class classifier which categorizes each input sample to 1 out of N different classes. This type of system can be evaluated using the *precision* and *recall* metrics. The precision for a given class is the ratio of correct predictions for that class to the full number of samples predicted as belonging to that class. The recall for a given class is the ratio of correct predictions for that class to the number of samples that actually belong to that class. We evaluate the accuracy of our classifier using the standard F1 score metric which considers both the precision and the recall of the test to compute the score [167]:

$$F1 = 2 \times \frac{precision * recall}{precision + recall}$$

Table 5.2 – Dataset for traffic data type.
Each sample includes the features extracted from 210 network packets of the application.

App. type	**Data type**	**App. Name**	**# of samples**	**App. workload**
Streaming	Audio	mplayer	59	Streaming radio stations indexed by www.xatworld.com
		mpg123	50	
		vlc player	59	
	Video	streamlinks	43	Streaming videos from Youtube, Vimeo and Aparat
		youtube-dl + mplayer	50	
		mpv	47	
File Download	File	wget	47	Downloading a file which is stored in a remote machine's file system: latextemplates.com, bensound.com and youtube.com
		curl	47	
		w3m	55	
Other	Other	firefox	47	Manual browsing
		Slack	51	
		Trello	45	
Total			**600**	

F1 scores range from 0 (worst possible score) to 1 (best possible score). The Macro average of F1 score is the average of the F1 scores of each individual class.

5.5.3 Machine learning model training

As previously discussed, an important concern in the GDPR is about clarification of sharing data with third parties and the type of data that is shared. From a privacy perspective, it is crucial to identify cases where an application claims that it is sharing one type of data while in reality it shares another type of data. We now show how traffic samples may be classified according to the data type they carry.

We performed around 12,000 experiments with different classifiers and subsets of the available features to find the best combination of features and algorithm for our classifica-

Table 5.3 – Application data type identification performance (model trained with samples of all applications).

	Precision	**Recall**	**F1-score**	**Support**
Audio stream	1.0	0.72	0.84	29
Video stream	0.90	0.87	0.89	31
File download	0.82	0.88	0.85	32
Other	0.77	0.96	0.86	28
Macro average	**0.87**	**0.86**	**0.86**	**120**

tion problem. As a result, we decided to use a *Decision Tree* classifier with the following features and hyper parameters in the Scikit-learn Python library:

Features list:

prtcl, deltatime_mean2, deltatime_std2, pcklength_mean2, pcklength_std2

Hyper parameters:

ccp_alpha: 0.0, class_weight: None, criterion: gini,
max_depth: 3, max_features: None, max_leaf_nodes: None,
min_impurity_decrease: 0.0, min_impurity_split: None,
min_samples_leaf: 1, min_samples_split: 2,
min_weight_fraction_leaf: 0.0, presort: deprecated,
randoms_state: 0, splitter: best

5.5.4 Application data type identification

We separate this evaluation in two parts. We first train the classifier with a random selection of traffic samples chosen from the full set of applications. Table 5.3 presents the classifier's performance in this case, with a F1 macro average of 86%. This shows that distinguishing different types of applications works with robust performance, even in difficult cases such as distinguishing audio streaming from video streaming.

We then run a similar experiment based on a model trained with samples from a subset of all applications, and tested with samples of the remaining applications. Specifically, we excluded the "mpg" application from the audio streaming class, "mpv" from the video streaming class, "curl" from the file download class, and "trello" from the Others class. This experiment is arguably representative of a scenario where a platform needs to analyze the behavior of a totally unknown application. The performance results are presented in

Table 5.4 – Application data type identification performance (model trained and tested with samples of different applications).

	Precision	**Recall**	**F1-score**	**Support**
Audio stream	0.97	0.78	0.87	50
Video stream	0.80	0.77	0.78	47
File download	0.78	0.83	0.80	47
Other	0.84	1.0	0.91	46
Macro average	**0.85**	**0.84**	**0.84**	**190**

Table 5.4. The F1 macro average is 84%. This is slightly lower than the case where the model could be trained with samples from all applications, showing that this second scenario is more challenging for the classifier. Nevertheless, the accuracy remains very high, showing that identifying application data types may be feasible despite having no access to the data payload. We obtained very similar results when excluding other applications from the samples set.

Note that, in dynamic environments such as our use case where the classifier may face samples from new applications on a regular basis, the model needs periodic retraining to maintain its accuracy.

5.6 Resource consumption

The fog computing testbed that we run the experiments on consists of five Raspberry PI version 4. Each Raspberry PI has a Quad core Cortex-A72 (ARM version 8) 64-bit SoC 1.5GHz and 4GB RAM. Raspberry Pis are often used by researchers to prototype fog computing platforms [168]–[171].

We capture the network traffic of each container using tcpdump [172]. Each traffic sample contains two minutes of network traffic. This operation consumes 3.4 ± 1.7 percent of CPU time in average. This parameter is measured using the Dstat library in Linux (kernel version 5.10.17) [173]. Dstat allows one to view system resources statistics in real-time. It is mainly used for monitoring systems during performance tuning tests, benchmarks or troubleshooting.

The training of the classifier model is done offline and may be executed outside the fog system. Using our dataset [166], we train a classifier outside the fog system and transfer the resulted classifier model into the physical fog machine. However, the inference part is done online on the fog machines. Our system targets one application to be monitored and

Table 5.5 – The average elapsed time and maximum memory usage for the feature extractor and the classifier.

	Elapsed time (ms)	RAM usage (MB)
The feature extractor	39.0 ±17.0	0.04 ±0.0
The classifier	16.4 ±6.5	0.01 ±0.0

captures the network traffic of the application for about 2 minutes. We extract features from the captured network traffic and feed the classifier with it. Both the feature extractor and classifier are written in Python. Table 5.5 presents the average elapsed time and maximum memory used for feature extractor and classifier.

We conclude that capturing network traffic does not affect the performance of the system as it does not consumes significant CPU resources. Moreover, the total time for feature extraction and the classification takes roughly 0.05 second, which is fast compared to the 2 minutes duration of the traffic samples.

5.7 Conclusion

This chapter presented our second contribution which is a model for automatically checking the privacy compliance of applications. Our model assumes applications run in a cloud or fog computing platform. From the technical perspective, our model builds upon the usage of machine learning methods and tools, suitably applied to features extracted from the network traffic captured in the host. We demonstrate that our approach is able to correctly identify shared data types by analyzing the applications' outgoing network traffic. Our system relies only on the header of captured packets, thereby working also in the presence of encrypted traffic. In our experiments the classifier is able to characterize the traffic type of already-known applications with 86% accuracy in terms of F1 score, and to attribute network traffic of unknown applications to the correct class with 84% accuracy. To allow the research community to expand along this research line and to benefit from the work carried out, we released (under a liberal CC BY 4.0 license) an open dataset of network traffic data of twelve real-world applications [166].

Chapter 6

Conclusion and Future work

6.1 Summary

The growth of IoT and emergence of new types of applications make the centralized model of cloud computing insufficient in some aspects. Bandwidth-intensive applications, latency-sensitive applications, applications with rapid mobility patterns, and applications that need to be deployed in geographic areas with unreliable connection to cloud servers are examples where cloud computing shows limitations.

Fog computing appears as a solution for the said types of applications by providing compute, network and storage resources close to the end-user. Fog computing is a location-aware computing paradigm which decreases transfer of data over long-distance networks by processing data close to its source.

The critical location of fog nodes between the end-user layer and the cloud layer provides them with privileged access to the data that are either produced by the end-user and sent to the cloud layer, or that are sent by the cloud layer to the end-user. These data often carry personal information about user's locations, activities, preferences, etc. Therefore, studying fog computing from the security and privacy perspective is essential.

In this thesis, we analyze the security of fog computing systems following a systematic approach from multiple perspectives: device level, system level, and service level. For each perspective, we identify assets of the fog system and their possible vulnerabilities. We also underline some possible countermeasures to protect the fog assets from possible threats.

In our study of the fog platform's security, one of the important identified assets is the user's personal data. As fog nodes are deployed in proximate location to the user, fog applications have access to significant parts of their users' data. A fog user's data may contain personal information and therefore be subject to the European General Data Protection Regulation (GDPR).

According to GDPR, every application that has access to personal data should expose a privacy policy describing how it handles users' personal data. Although most applications

indeed provide a privacy policy, the actual compliance of applications to their privacy policy remains to be verified. For example, personal assistants such as Google Assistant and Amazon Alexa have access to personal information such as user's voice and location. Although these applications expose a privacy policy claiming to handle personal data in a privacy-preserving manner, several incidents have demonstrated violations of the announced privacy policies [161], [162].

Manually checking whether applications actually handle personal data according to the claims made in their privacy policy is both error-prone and time-consuming. We therefore study the feasibility of automated privacy compliance checking of applications from their cloud or fog hosting platform. We argue that the hosting platform already has access to useful information about the application that can be utilized to understand about its behavior concerning its privacy claims. We therefore use fog as the hosting platform and show the feasibility of automated privacy compliance checking by presenting an empirical procedure to identify one type of privacy-oriented behavior.

In the first contribution of this thesis, we presented a methodology for analyzing the security of fog computing platforms systematically. Currently, there is no specific tool or methodology for analyzing the security of fog computing systems in a comprehensive way. General security evaluation procedures applicable for most information technology products are time-consuming, costly, and badly suited to the fog context. We applied this methodology to a generic fog computing system, showcasing how this approach can be purposefully used by security analysts and system designers. We believe that our approach can provide a valuable tool for both researchers and practitioners.

In the second contribution of this thesis, we argued that the privacy compliance of applications hosted in cloud or fog computing platforms can and should be automatically carried out by the platform itself. We discussed the feasibility of unintrusive and application-agnostic monitoring in the platform layer to check the privacy compliance of applications. In our proposed model, the platform monitors an application's privacy-oriented behavior through signals such as its network traffic characteristics. These signals can be analyzed and compared with the principles found in the application's privacy policy. We presented a procedure based on machine-learning techniques to identify the type of data being shared by applications with external third parties even if the application uses encrypted communications. Our classifiers identify traffic samples of applications with 86% of F1-score.

6.2 Future directions

6.2.1 Analysis of fog computing security

In terms of extensions and enhancements to the introduced methodology for analyzing the security of fog computing systems, a significant contribution would be represented by the creation of a catalog of possible attacks and defenses in fog computing systems. Yet, it is worth remarking that, as the fog computing landscape is still in its infancy, the emergence of (de facto or formal) standards and the consolidation of business models for fog computing infrastructures and services may require frequent updates to said catalog.

The methodology in this thesis is also based on some assumptions about the security of other elements in an IoT system. For example, "the IoT device is safe" is a very strong assumption that we put in our analysis to simplify the problem of analyzing the security of a fog system, while in reality this assumption needs to be satisfied. Satisfying all of these assumptions is not a trivial work.

Moreover, the proposed methodology does not discuss countermeasures for all possible threats. We only introduced some directions for tackling said threats. As a further work, defining classes of countermeasures will provide more detailed guidance to security analysts. Furthermore, by defining said classes of countermeasures, the security analyst can utilize the attack-defense tree of a fog system and check if all the identified vulnerabilities are tackled by proper countermeasures introduced beforehand. Involving countermeasures in the analysis can lead to grading the level of security of a fog system. As a result, the user of a fog system can have a sense of the level of security of the fog system that she is using. Thus, it will be a good direction for further work to include the countermeasures and a checklist to grade the fog system against existence of such countermeasures.

6.2.2 Automatic privacy compliance checking of applications

In the second contribution we proposed the vision of automatic mechanisms to assess whether an application truly respects its own privacy policy. However, this study constitutes a first step, and numerous additional research directions remain to be explored before a practical implementation of this idea may become feasible.

Other application signals and privacy-oriented behaviors

Our work focused on a single privacy-oriented behavior (namely data sharing). To expand the same idea to other aspects of users privacy, the detection of other privacy-oriented behaviors would require further study. Moreover, we utilized a single application signal (network traffic). A promising extension of this work would be to make use of additional sources of signal such as CPU usage and system calls issued by the application. As a result we foresee the design of a complete privacy compliance checking system which exploits different application signals to detect a wide range of privacy-oriented behaviors. Depending on the available resources on the platform, the monitoring may be performed online or offline, either continuously or based on randomly monitoring a subset of all applications for a limited time period.

Malicious applications

Due to the GDPR and similar enforcement regulations, we rely on the assumption that applications and their users accept to be monitored by the platform, and that the application does not actively try to evade this type of monitoring.

However, it could be imagined that with the development of privacy monitoring platforms, applications that do not respect the user's privacy may develop strategies to obfuscate their behavior and mislead the monitoring system. For instance, an application may intentionally produce misleading signals to evade detection of incorrect behavior. Therefore, research on improving techniques to detect these behaviors will be necessary. However, this topic may be considered as a security challenge rather than a privacy-protection one.

Finer-grained classification

In contrast to similar works in the literature, we have used a fine level granularity for our data type classes. We separated streaming class to video and audio streaming classes for the sake of specificity, as encouraged in GDPR privacy policies. However, research on the development of ML models to classify data in finer-grained classes which targets privacy concepts remains unexplored and yet essential.

Exploring other machine learning techniques for privacy-oriented behavior detection

In this thesis, we applied supervised machine-learning techniques to infer the type of shared data with third parties. Although the results where promising, we needed access to a dataset an train the model using this dataset. However, exploring the other machine learning techniques such as unsupervised-learning and deep learning would be interesting research direction and may eliminate the effort of creating a dataset.

Monitoring the behavior in other platforms

In this thesis, we defined the application as a set of one or more services that expose one privacy policy. These services may be deployed in multiple environments. For example, some of the services may be deployed in fog and other parts in the cloud. An IoT data analytics platform may be comprised of services that are preprocessing IoT data in the fog platform and services that are deployed in the cloud platform for aggregating data or storing historical data. Another example is the multi-cloud applications. As a result, we may want to monitor the application in more than one place. Although we used a fog platform in our prototypical implementation, we can replicate the same method for a cloud platform. However, it could be imagined that some application uses the same code for all the platforms, for example in multi-cloud scenario, and therefore monitoring the behavior of the application in one place provides useful information about the application's overall behavior.

Privacy compliance of the monitoring platform itself

Our envisaged privacy monitoring platform is designed to be as little intrusive as possible, for example by not adding any code in the application nor monitoring the applications' payload. However, this monitoring system should in principle also provide a privacy policy of its own, and convince its users that it actually respects this privacy policy. However, contrary to arbitrary cloud or fog computing applications which cannot realistically all be checked by hand, the situation is arguably different for a single platform focused toward privacy policy enforcement. Once a platform has demonstrated (e.g., through manual code analysis) that it respects its own privacy policy, various techniques may be used to prove to its users that their applications are actually being monitored using an unmodified version of the monitoring platform.

BIBLIOGRAPHY

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, « Cloud computing and grid computing 360-degree compared », *in Proc. IEEE grid computing environments workshop*, 2008.

[2] Google Cloud, *Expanding our global infrastructure with new regions and subsea cables*, `https://www.blog.google/products/google-cloud/expanding-our-global-infrastructure-new-regions-and-subsea-cables/`.

[3] Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper*, `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`.

[4] B. Varghese, N. Wang, D. S. Nikolopoulos, and R. Buyya, « Feasibility of fog computing », *in Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things*, Springer, 2020.

[5] *CLAudit project, Planetary-scale cloud latency auditing platform*, `http://claudit.feld.cvut.cz/index.php`.

[6] A. Ahmed, H. Arkian, D. Battulga, J. A. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. O. Gutierrez, G. Pierre, P. R. Souza Jr., M. A. Tamiru, and L. Wu, « Fog Computing Applications: Taxonomy and Requirements », *CoRR*, vol. abs/1907.11621, 2019, `http://arxiv.org/abs/1907.11621`.

[7] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, « Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing », *in Proc. IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, 2014.

[8] *IoT Data Volume*, `https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/`.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, « Fog computing and its role in the Internet of Things », *in Proc. workshop on Mobile cloud computing*, 2012.

[10] *Pico Cluster*, https://www.picocluster.com/collections/pico-10.

[11] McKinsey, « The Internet of Things: Five critical questions », https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-internet-of-things-five-critical-questions, 2015.

[12] S. Ganguli and T. Friedman, « IoT Technology Disruptions: A Gartner Trend Insight Report », Gartner, Tech. Rep., 2018, https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders/.

[13] M. Farhadi, J.-L. Lanet, G. Pierre, and D. Miorandi, « A systematic approach toward security in Fog computing: Assets, vulnerabilities, possible countermeasures », *Software: Practice and Experience*, vol. 50, *6*, 2020.

[14] EU Parliament, *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 (General Data Protection Regulation)*, https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1590072813969&uri=CELEX:32016R0679, 2016.

[15] J. Mohan, M. Wasserman, and V. Chidambaram, « Analyzing GDPR compliance through the lens of privacy policy », *in Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, Springer, 2019.

[16] Information Commissioner's Office, *The UK GDPR*, https://ico.org.uk/for-organisations/dp-at-the-end-of-the-transition-period/data-protection-now-the-transition-period-has-ended/the-gdpr/.

[17] State of California Department of Justice, *California Consumer Privacy Act (CCPA)*, https://oag.ca.gov/privacy/ccpa.

[18] M. Day, G. Turner, and N. Drozdiak, *Thousands of Amazon workers listen to Alexa users' conversations*, Time, https://time.com/5568815/amazon-workers-listen-to-alexa/, 2019.

[19] ——, *Amazon workers are listening to what you tell Alexa*, Bloomberg news, https://www.bloomberg.com/news/articles/2019-04-10/is-anyone-listening-to-you-on-alexa-a-global-team-reviews-audio, 2019.

[20] A. Cuthbertson, *Amazon admits employees listen to Alexa conversations*, The Independent, https://www.independent.co.uk/life-style/gadgets-and-tech/news/amazon-alexa-echo-listening-spy-security-a8865056.html, 2019.

[21] M. Hatamian, « Engineering Privacy in Smartphone Apps: A Technical Guideline Catalog for App Developers », *IEEE Access*, vol. 8, 2020.

[22] Wikipedia, *Trust, but verify*, https://en.wikipedia.org/wiki/Trust,_but_verify.

[23] Common Criteria, *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model version 3.1, Revision 3 (CCMB-2009-07-001)*, 2009.

[24] B. Schneier, « Attack trees », *Dr. Dobb's journal*, vol. 24, *12*, 1999.

[25] « Kubernetes », https://kubernetes.io/.

[26] T. Nelson, « Computer Lib/Dream Machines (1974) », *Multimedia: From Wagner to Virtual Reality*, 2001.

[27] E. Gorelik, « Cloud computing models », Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[28] IBM, https://www.ibm.com/.

[29] Wikipedia, *Sun Microsystems*, https://en.wikipedia.org/wiki/Sun_Microsystems.

[30] HP, https://www8.hp.com/fr/fr/home.html.

[31] Wikipedia, *Digital Equipment Corporation*, https://en.wikipedia.org/wiki/Digital_Equipment_Corporation.

[32] B. Hayes, « Cloud computing », *Communications of the ACM*, vol. 51, *7*, 2008.

[33] Microsoft, *What is cloud computing?*, https://azure.microsoft.com/en-gb/overview/what-is-cloud-computing/.

[34] N. Galov, *Incredible Cloud Adoption Stats*, HostingTribunal, https://hostingtribunal.com/blog/cloud-adoption-statistics/, 2021.

[35] A. Desai, R. Oza, P. Sharma, and B. Patel, « Hypervisor: A survey on concepts and taxonomy », *International Journal of Innovative Technology and Exploring Engineering*, vol. 2, *3*, 2013.

[36] N. Jain and S. Choudhary, « Overview of virtualization in cloud computing », *in Proc. IEEE Symposium on Colossal Data Analysis and Networking*, 2016.

[37] T. Wood, E. Cecchet, K. K. Ramakrishnan, P. J. Shenoy, J. E. van der Merwe, and A. Venkataramani, « Disaster recovery as a cloud service: economic benefits & deployment challenges », *in Proc. HotCloud workshop*, 2010.

[38] A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, « Exploring container virtualization in IoT clouds », *in Proc. IEEE International Conference on Smart Computing*, 2016.

[39] P. Bellavista and A. Zanni, « Feasibility of fog computing deployment based on Docker containerization over Raspberry Pi », *in Proc. International conference on distributed computing and networking*, 2017.

[40] M. Eder, « Hypervisor vs. container-based virtualization », *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications*, vol. 1, 2016.

[41] Amazon, *Amazon Solutions*, `https://aws.amazon.com/`.

[42] Google Cloud, *Google Cloud*, `https://cloud.google.com/`.

[43] A. Ahmed, « Efficient cloud application deployment in distributed fog infrastructures », Ph.D. dissertation, Université de Rennes 1, France, 2020.

[44] D. Bermbach, F. Pallas, D. G. Pérez, P. Plebani, M. Anderson, R. Kat, and S. Tai, « A research perspective on fog computing », *in Proc. International Conference on Service-Oriented Computing*, Springer, 2017.

[45] S. Pallewatta, V. Kostakos, and R. Buyya, « Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments », *in Proc. IEEE/ACM International Conference on Utility and Cloud Computing*, 2019.

[46] IEEE Standards Association, *1934-2018 – IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*, `https://standards.ieee.org/standard/1934-2018.html`, 2018.

[47] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, « All one needs to know about fog computing and related edge computing paradigms: A complete survey », *Journal of Systems Architecture*, vol. 98, 2019.

[48] A. Fahs, « Proximity-Aware Replicas Management in Geo-Distributed Fog Computing Platforms », Ph.D. dissertation, University of Rennes 1, 2020.

[49] S. Yi, Z. Qin, and Q. Li, « Security and privacy issues of fog computing: A survey », *in Proc. International conference on wireless algorithms, systems, and applications*, 2015.

[50] S. A. Noghabi, L. Cox, S. Agarwal, and G. Ananthanarayanan, « The emerging landscape of edge computing », *GetMobile: Mobile Computing and Communications*, vol. 23, *4*, 2020.

[51] A. J. Fahs, G. Pierre, and E. Elmroth, « Voilà: Tail-latency-aware fog application replicas autoscaler », *in Proc. IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2020.

[52] S. Taherizadeh, V. Stankovski, and M. Grobelnik, « A capillary computing architecture for dynamic Internet of Things: Orchestration of microservices from edge devices to fog and cloud providers », *Sensors*, vol. 18, *9*, 2018.

[53] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, « Borg, Omega, and Kubernetes », *ACM Queue*, vol. 14, *1*, 2016.

[54] The Cloud Native Computing Foundation, *CNCF survey 2019*, `https://www.cncf.io/wp-content/uploads/2020/08/CNCF_Survey_Report.pdf`.

[55] Datadog, *11 facts about real-world container use*, `https://www.datadoghq.com/container-report/`.

[56] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, « Geo-distributed efficient deployment of containers with Kubernetes », *Computer Communications*, vol. 159, 2020.

[57] J. Santos, T. Wauters, B. Volckaert, and F. Turck, « Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing applications », *in Proc. IEEE Conference on Network Softwarization*, 2019.

[58] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, « Instability in Geo-Distributed Kubernetes Federation: Causes and Mitigation », *in Proc. IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2020.

[59] Kubernetes, *Set up a High-Availability Control Plane*, `https://kubernetes.io/docs/tasks/administer-cluster/highly-available-master/`.

[60] C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin, and J.-Y. Jeng, « A Kubernetes-based monitoring platform for dynamic cloud resource provisioning », *in Proc. IEEE Global Communications Conference*, 2017.

[61] P. Mytilinakis, « Attack methods and defenses on Kubernetes », M.S. thesis, University of Piraeus, 2020.

[62] Kubernetes, *Container runtimes*, `https://kubernetes.io/docs/setup/production-environment/container-runtimes/`.

[63] *CRI-O*, `https://cri-o.io/`.

[64] *Containerd*, `https://containerd.io/`.

[65] Kubernetes, *Don't Panic: Kubernetes and Docker*, `https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/`.

[66] B. Ismail, E. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, « Evaluation of Docker as edge computing platform », *in Proc. IEEE Conference on Open Systems*, 2015.

[67] *flannel*, `https://github.com/flannel-io/flannel`.

[68] Calico, *What is Calico?*, `https://www.projectcalico.org/`.

[69] Weaveworks, *Introducing Weave Net*, `https://www.weave.works/docs/net/latest/overview/`.

[70] P. Johnston, S. Sharma, and O. Filiz, *Microsoft Azure Container Networking*, `https://github.com/Azure/azure-container-networking/blob/master/docs/cni.md`.

[71] M. Betz, *Understanding Kubernetes networking*, `https://medium.com/google-cloud/understanding-kubernetes-networking-pods-7117dd28727`.

[72] Docker, *Docker network setting*, `https://docs.docker.com/engine/reference/run/#network-settings`.

[73] Wikipedia, *Common Criteria*, `https://en.wikipedia.org/wiki/Common_Criteria`, 2019.

[74] S. Bistarelli, F. Fioravanti, and P. Peretti, « Defense trees for economic evaluation of security investments », *in Proc. IEEE International Conference on Availability, Reliability and Security*, 2006.

[75] A. Roy, D. S. Kim, and K. S. Trivedi, « Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees », *Security and Communication Networks*, vol. 5, *8*, 2012.

[76] M. Bouissou and J.-L. Bon, « A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes », *Reliability Engineering & System Safety*, vol. 82, *2*, 2003.

[77] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, « Interpretable machine learning: definitions, methods, and applications », *arXiv preprint arXiv:1901.04592*, 2019.

[78] MIT technology review, *What is machine learning?*, `https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/`, 2018.

[79] T. M. Mitchell, « Machine learning and data mining », *Communications of the ACM*, vol. 42, *11*, 1999.

[80] Wikipedia, *Machine learning*, `https://en.wikipedia.org/wiki/Machine_learning`, 2018.

[81] S. B. Kotsiantis, I Zaharakis, and P Pintelas, « Supervised machine learning: A review of classification techniques », *Emerging artificial intelligence applications in computer engineering*, vol. 160, *1*, 2007.

[82] D. Soni, *Supervised vs. Unsupervised Learning*, `https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d`.

[83] J. Erman, M. Arlitt, and A. Mahanti, « Traffic classification using clustering algorithms », *in Proc. SIGCOMM workshop on Mining network data*, 2006.

[84] R. Kohavi and P. Foster, « Special issue on applications of machine learning and the knowledge discovery process », *Journal of Machine Learning*, vol. 30, 1998.

[85] M. J. Zaki, W. Meira Jr, and W. Meira, *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[86] C. Zhang and Y. Ma, *Ensemble machine learning: methods and applications*. Springer, 2012.

[87] Wikipedia, *Hyperparameter (machine learning)*, `https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)`.

[88] S. Arlot and A. Celisse, « A survey of cross-validation procedures for model selection », *Statistics surveys*, vol. 4, 2010.

[89] *Precision and recall*, https://en.wikipedia.org/wiki/Precision_and_recall.

[90] A. Shastri, *Three decision tree-based algorithms for Machine Learning*, https://towardsdatascience.com/3-decision-tree-based-algorithms-for-machine-learning-75528a0f03d1.

[91] D. Varghese, *Comparative Study on Classic Machine learning Algorithms*, https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222.

[92] A. S. Tanenbaum and M. van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[93] J. D. Moffett, « Security & Distributed Systems », 1995.

[94] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, « Network attacks: Taxonomy, tools and systems », *Journal of Network and Computer Applications*, vol. 40, 2014.

[95] A. Simmonds, P. Sandilands, and L. van Ekert, « An ontology for network security attacks », *in Proc. Asian Applied Computing Conference*, Springer, 2004.

[96] L. Lamport, R. Shostak, and M. Pease, « The Byzantine generals problem », *ACM Transactions on Programming Languages and Systems*, vol. 4, *3*, 1982.

[97] H. Li and M. Singhal, « Trust management in distributed systems », *Computer*, vol. 40, *2*, 2007.

[98] J. G. Steiner, B. C. Neuman, and J. I. Schiller, « Kerberos: An Authentication Service for Open Network Systems », *in Usenix Conference*, 1988.

[99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, « The role of trust management in distributed systems security », *in Secure Internet Programming*, Springer, 1999.

[100] J. Kindervag, « Build Security Into Your Network's DNA: The Zero Trust Network Architecture », *Forrester Research Inc*, 2010, http://www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf.

[101] M. A. Khan and K. Salah, « IoT security: Review, blockchain solutions, and open challenges », *Future Generation Computer Systems*, vol. 82, 2018.

[102] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, « Internet of Things: Vision, applications and research challenges », *Ad hoc networks*, vol. 10, *7*, 2012.

[103] M. Ammar, G. Russello, and B. Crispo, « Internet of Things: A survey on the security of IoT frameworks », *Journal of Information Security and Applications*, vol. 38, 2018.

[104] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, « Demystifying IoT security: an exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations », *IEEE Communications Surveys & Tutorials*, vol. 21, *3*, 2019.

[105] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, « A survey of IoT-enabled cyberattacks: Assessing attack paths to critical infrastructures and services », *IEEE Communications Surveys & Tutorials*, vol. 20, *4*, 2018.

[106] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, « IoT security: ongoing challenges and research opportunities », *in Proc. IEEE International conference on service-oriented computing and applications*, 2014.

[107] K. Zhao and L. Ge, « A survey on the Internet of things security », *in Proc. IEEE International conference on computational intelligence and security*, 2013.

[108] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, « Security, privacy and trust in Internet of Things: The road ahead », *Computer networks*, vol. 76, 2015.

[109] M. Abomhara and G. M. Køien, « Security and privacy in the Internet of Things: Current status and open issues », *in Proc. IEEE International conference on privacy and security in mobile systems*, 2014.

[110] C. Liu, C. Yang, X. Zhang, and J. Chen, « External integrity verification for outsourced big data in cloud and IoT: A big picture », *Future generation computer systems*, vol. 49, 2015.

[111] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, « Security of the Internet of Things: perspectives and challenges », *Wireless Networks*, vol. 20, *8*, 2014.

[112] D. Chen and H. Zhao, « Data security and privacy protection issues in cloud computing », *in Proc. IEEE International Conference on Computer Science and Electronics Engineering*, 2012.

[113] Q. Gou, L. Yan, Y. Liu, and Y. Li, « Construction and strategies in IoT security system », *in Proc. IEEE International conference on green computing and communications and internet of things and cyber, physical and social computing*, 2013.

[114] M. Wazid, A. K. Das, R. Hussain, G. Succi, and J. J. Rodrigues, « Authentication in cloud-driven IoT-based big data environment: Survey and outlook », *Journal of Systems Architecture*, vol. 97, 2019.

[115] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, « Evaluating critical security issues of the IoT world: Present and future challenges », *IEEE Internet of Things Journal*, vol. 5, *4*, 2017.

[116] Flexera, « RightScale State of the Cloud Report », `https://www.rightscale.com/lp/state-of-the-cloud`, 2019.

[117] Cloud Security Alliance, « Top Threats to Cloud Computing–The Egregious 11 », `https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/`, 2019.

[118] R. Roman, J. Lopez, and M. Mambo, « Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges », *Future Generation Computer Systems*, vol. 78, 2018.

[119] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar, « Security and privacy in fog computing: Challenges », *IEEE Access*, vol. 5, 2017.

[120] H. Pang and K.-L. Tan, « Authenticating query results in edge computing », *in Proc. IEEE International Conference on Data Engineering*, 2004.

[121] A. Alwarafy, K. A. Al-Thelaya, M. Abdallah, J. Schneider, and M. Hamdi, « A survey on security and privacy issues in edge computing-assisted internet of things », *IEEE Internet of Things Journal*, 2020.

[122] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, « An overview of fog computing and its security issues », *Concurrency and Computation: Practice and Experience*, vol. 28, *10*, 2016.

[123] S. Khan, S. Parkinson, and Y. Qin, « Fog computing security: a review of current applications and security solutions », *Journal of Cloud Computing*, vol. 6, *1*, 2017.

[124] H. Liu, Y. Zhang, and T. Yang, « Blockchain-Enabled Security in Electric Vehicles Cloud and Edge Computing », *IEEE Network*, vol. 32, *3*, 2018.

[125] M. Farhadi, D. Miorandi, and G. Pierre, « Blockchain enabled fog structure to provide data security in IoT applications », *Proc. Middleware Doctoral Symposium*, 2019.

[126] X. Xu, Y. Chen, X. Zhang, Q. Liu, X. Liu, and L. Qi, « A blockchain-based computation offloading method for edge computing in 5G networks », *Software: Practice and Experience*, 2019.

[127] M. Yahuza, M. Y. I. B. Idris, A. W. B. A. Wahab, A. T. Ho, S. Khan, S. N. B. Musa, and A. Z. B. Taha, « Systematic review on security and privacy requirements in edge computing: State of the art and future research opportunities », *IEEE Access*, vol. 8, 2020.

[128] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng, « Fog Computing for the Internet of Things: Security and Privacy Issues », *IEEE Internet Computing*, vol. 21, *2*, 2017.

[129] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, « A systematic survey of industrial internet of things security: Requirements and fog computing opportunities », *Communications Surveys & Tutorials*, vol. 22, *4*, 2020.

[130] T. Khalid, M. A. Abbasi, M. Zuraiz, A. N. Khan, M. Ali, R. W. Ahmad, J. J. Rodrigues, and M. Aslam, « A survey on privacy and access control schemes in fog computing », *International Journal of Communication Systems*, vol. 34, *2*, 2021.

[131] L. Wang, H. An, and Z. Chang, « Security Enhancement on a Lightweight Authentication Scheme With Anonymity Fog Computing Architecture », *IEEE Access*, vol. 8, 2020.

[132] R.-H. Hsu, J. Lee, T. Q. Quek, and J.-C. Chen, « Reconfigurable security: Edge computing-based framework for IoT », *IEEE Network*, vol. 32, *5*, 2018.

[133] C. Esposito, A. Castiglione, F. Pop, and K.-K. R. Choo, « Challenges of Connecting Edge and Cloud Computing: A Security and Forensic Perspective », *IEEE Cloud computing*, vol. 4, *2*, 2017.

[134] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, « Edge intelligence: the confluence of edge computing and artificial intelligence », *IEEE Internet of Things Journal*, vol. 7, *8*, 2020.

[135] A. M. McDonald and L. F. Cranor, « The cost of reading privacy policies », *Journal of Law and Policy for the Information Society*, vol. 4, *3*, 2008.

[136] S. Kasem-Madani and M. Meier, « Security and Privacy Policy Languages: A Survey, Categorization and Gap Identification », *CoRR*, vol. abs/1512.00201, 2015, `http://arxiv.org/abs/1512.00201`.

[137] S. Wilson, F. Schaub, A. A. Dara, F. Liu, S. Cherivirala, P. G. Leon, M. S. Andersen, S. Zimmeck, K. M. Sathyendra, N. C. Russell, T. B. Norton, E. Hovy, J. Reidenberg, and N. Sadeh, « The creation and analysis of a website privacy policy corpus », *in Proc. Annual Meeting of the Association for Computational Linguistics*, 2016.

[138] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, « Polisis: Automated analysis and presentation of privacy policies using deep learning », *in Proc. USENIX Security Symposium*, 2018.

[139] Wikipedia, *Online Privacy Protection Act*, `https://en.wikipedia.org/wiki/Online_Privacy_Protection_Act`.

[140] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, « TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones », *ACM Transactions on Computer Systems*, vol. 32, *2*, 2014.

[141] M. Hatamian, N. Momen, L. Fritsch, and K. Rannenberg, « A multilateral privacy impact analysis method for Android apps », *in Proc. Annual Privacy Forum*, 2019.

[142] L. Bernaille, R. Teixeira, and K. Salamatian, « Early application identification », *in Proc. ACM CoNEXT conference*, 2006.

[143] T. J. Parvat and P. Chandra, « A Novel Approach to Deep Packet Inspection for Intrusion Detection », *Procedia Computer Science*, vol. 45, 2015.

[144] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, « Towards the deployment of machine learning solutions in network traffic classification: A systematic survey », *IEEE Communications Surveys & Tutorials*, vol. 21, 2018.

[145] P.-O. Brissaud, J. François, I. Chrisment, T. Cholez, and O. Bettan, « Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic », *IEEE Transactions on Network and Service Management*, vol. 16, *3*, 2019.

[146] G. Draper-Gil, A. Habibi Lashkari, M. S. I. Mamun, and A. A. Ghorbani, « Characterization of encrypted and VPN traffic using time-related features », *in Proc. International conference on information systems security and privacy*, 2016.

[147] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, « Deep packet: A novel approach for encrypted traffic classification using deep learning », *Soft Computing*, vol. 24, *3*, 2020.

[148] W. Zhou and S. Piramuthu, « Security/privacy of wearable fitness tracking IoT devices », *in Proc. IEEE Conference on Information Systems and Technologies*, 2014.

[149] L. Zhang, W. Jia, S. Wen, and D. Yao, « A man-in-the-middle attack on 3G-WLAN interworking », *in Proc. IEEE Conference on Communications and Mobile Computing*, 2010.

[150] G. Bouffard and J.-L. Lanet, « Reversing the operating system of a Java based smart card », *Journal of Computer Virology and Hacking Techniques*, vol. 10, *4*, pp. 239–253, 2014.

[151] Wikipedia, *Communication channel*, `https://en.wikipedia.org/wiki/Communication_channel`, 2019.

[152] T. Alladi, V. Chamola, B. Sikdar, and K.-K. R. Choo, « Consumer IoT: Security vulnerability case studies and solutions », *IEEE Consumer Electronics Magazine*, vol. 9, *2*, 2020.

[153] D. Puthal, S. P. Mohanty, S. A. Bhavake, G. Morgan, and R. Ranjan, « Fog Computing Security Challenges and Future Directions [Energy and Security] », *IEEE Consumer Electronics Magazine*, vol. 8, *3*, 2019.

[154] C. Skouloudi and G. Fernández, « Towards secure convergence of Cloud and IoT », The European Union Agency for Cybersecurity, `https://www.enisa.europa.eu/news/enisa-news/towards-secure-convergence-of-cloud-and-iot`, 2018.

[155] D. Gruss, C. Maurice, and S. Mangard, « Rowhammer.js: A remote software-induced fault attack in javascript », *in Proc. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.

[156] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, « Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis », *in Proc. Network and Distributed System Security Symposium*, 2007.

[157] W. G. Halfond, J. Viegas, and A. Orso, « A classification of SQL-injection attacks and countermeasures », *in Proc. IEEE International Symposium on Secure Software Engineering*, 2006.

[158] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, « Control-flow integrity principles, implementations, and applications », 1, vol. 13, 2009.

[159] M. Conti, S. Crane, L. Davi, M. Franz, P. Larsen, M. Negro, C. Liebchen, M. Qunaibit, and A.-R. Sadeghi, « Losing control: On the effectiveness of control-flow integrity under stack attacks », *in Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[160] T. H. Dang, P. Maniatis, and D. Wagner, « The performance cost of shadow stacks and stack canaries », *in Proc. ACM Symposium on Information, Computer and Communications Security*, 2015.

[161] A. Hern, *Amazon staff listen to customers' Alexa recordings, report says*, The Guardian, `https://bit.ly/2NHg74P`, 2019.

[162] T. Haselton, *Google admits partners leaked more than 1,000 private conversations with Google Assistant*, CNBC, `https://cnb.cx/3iqvkpc`, 2019.

[163] « ksniff for Kubernetes », `https://github.com/futuretea/ksniff`.

[164] B. Boucheron, *How To Inspect Kubernetes Networking*, DigitalOcean, `https://do.co/2VEMxBs`, 2018.

[165] O. Mula-Valls, « A practical retraining mechanism for network traffic classification in operational environments », M.S. thesis, Universitat Politecnica de Catalunya, 2011.

[166] M. Farhadi, *Application specific network traffic with specified activities*, version 1.0, `https://doi.org/10.5281/zenodo.3965834`, Jul. 2020.

[167] Wikipedia, *F1 score*, `https://en.wikipedia.org/wiki/F1_score`.

[168] H. Arkian, D. Giouroukis, P. Souza Junior, and G. Pierre, « Potable Water Management with integrated Fog computing and LoRaWAN technologies », *IEEE IoT Newsletter*, 2020.

[169] A. van Kempen, T. Crivat, B. Trubert, D. Roy, and G. Pierre, « MEC-ConPaaS: An experimental single-board based mobile edge cloud », *in Proc. IEEE Mobile Cloud*, Apr. 2017.

[170] P. Bellavista and A. Zanni, « Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi », *in Proc. ACM ICDCN*, 2017.

[171] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, « Fogernetes: Deployment and management of fog computing applications », *in Proc. IEEE/IFIP NOMS*, 2018.

[172] « TCPDUMP », `https://www.tcpdump.org/`.

[173] « Dstat », `http://dag.wiee.rs/home-made/dstat/`.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Vérification automatisée de la conformité de la confidentialité des applications dans les environnements Fog distribués.

Mot clés : Fog computing, sécurité, confidentialité.

Résumé : Le “fog computing,” comme toute nouvelle technologie, soulève des inquiétudes des utilisateurs concernant la sécurité et la confidentialité. Dans cette thèse, nous analysons la sécurité des systèmes fog en suivant une approche systématique sous plusieurs angles : niveau matériel, niveau système et niveau service. Pour chaque perspective, nous discutons des vulnérabilités possibles que le système peut avoir et mettons en évidence quelques solutions possibles. L'un des aspects importants identifiés dans notre étude de la sécurité de la plate-forme fog est constitué des données personnelles de l'utilisateur. En raison de la proximité des nœuds fog par rapport à l'utilisateur, les applications fog ont accès à des parties importantes des données personnelles de leurs utilisateurs. Bien que les applications exposent une politique de confidentialité décrivant comment elles traitent les données personnelles des utilisateurs, la conformité des applications à leur politique de confidentialité ne doit pas être considérée comme acquise mais vérifiée expérimentalement. Cependant, vérifier manuellement si les applications respectent réellement les clauses formulées dans leur politique de confidentialité est à la fois sujet aux erreurs et chronophage. Dans cette thèse, nous montrons que la vérification automatisée de la conformité à la confidentialité dans un environnement fog est faisable, et présentons une feuille de route de recherche vers le développement de tels systèmes.

Title: Automated application privacy compliance checking in distributed Fog environments.

Keywords: Fog computing, security, privacy.

Abstract: Fog computing, like any other new technology, raises concerns regarding the security and privacy of its users. In this thesis, we analyze the security of fog computing systems following a systematic approach and from multiple perspectives: device level, system level, and service level. For each perspective, we discuss the possible vulnerabilities that the system may have and highlight some possible solutions. One of the important identified assets in our study of fog platform's security is the user's personal data. Because of fog nodes' proximate location to the user, fog applications have access to significant parts of their users' personal data. Although applications expose a privacy policy describing how they handle users' personal data, the compliance of applications to their privacy policy should not be taken for granted but verified. However, manually checking whether applications actually respect the claims made in their privacy policy is both error-prone and time-consuming. In this thesis, we argue that automated privacy compliance checking in fog environment is feasible and outline a research roadmap towards the development of such systems.