Towards automated privacy compliance checking of applications in Cloud and Fog environments

Mozhdeh Farhadi, Guillaume Pierre, Daniele Miorandi

Towards automated privacy compliance checking of applications in Cloud and Fog environments

Mozhdeh Farhadi
U-Hopper, Univ Rennes, Inria, CNRS, IRISA
Rennes, France
mozhdeh.farhadi@irisa.fr

Guillaume Pierre
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
guillaume.pierre@irisa.fr

Daniele Miorandi
U-Hopper
Trento, Italy
daniele.miorandi@u-hopper.com

***Abstract*— Internet application users are increasingly concerned about the way applications handle their personal data. However, manually checking whether applications actually respect the claims made in their privacy policy is both error-prone and time-consuming. This paper claims that the privacy compliance of applications hosted in cloud or fog computing platforms can and should be automatically carried by the platform itself. We discuss the feasibility of unintrusive and application-agnostic monitoring in the platform layer to check the privacy compliance of applications. First, the platform may monitor an application's privacy-oriented behavior through signals such as its network traffic characteristics. Second, these signals can be analyzed and compared with the principles found in the application's privacy policy. 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. Our classifiers identify traffic samples of applications with 86% accuracy.**

***Index Terms*—Privacy compliance checking, Cloud computing, Fog computing, Network traffic classification, Machine learning, Kubernetes.**

I. 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 [1], 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 [2]. 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 users' 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 [3], [4]. 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 [5], [6]. 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 paper that privacy compliance checking should not be done manually (which is error-prone and time-consuming) but that the platform that hosts applications constitute a strategic location where the actual behavior of these applications may be automatically monitored and verified against the published privacy policies. The platform is managed by the fog or cloud provider and monitors the virtual machines and containers where the applications are running.

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 in natural language by lawyers; 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 [7], [8], 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.

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

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.

which data types an application is allowed or disallowed to share with specific third parties, is at the heart of the European GDPR regulation [1]. Other combinations of application signals and privacy principles are out of the scope of this paper.

We use the cloud/fog platform that hosts the application to monitor the application behavior because the platform has access to many useful informations about the applications, such as their CPU usage and network traffic and may already use them for performance purposes. In our experiments setup we use fog computing platform, because it is an easily accessible form of a platform. However the proposed model is reproducible in cloud platforms.

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 simple 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 paper is organized as follows. Section II presents related works. Section III describes our privacy compliance checking model. Section IV presents evaluation results. Section V discusses these results. Finally, Section VI concludes.

II. Related work

A. Privacy policy interpretation

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 [9], 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 [7], Wilson *et al* define a taxonomy of privacy policy principles and annotated 115 privacy policy texts based on their defined taxonomy (OPP-115) with the help of law school students. They utilized this dataset to train a model for automatic interpretation of other privacy policies.

In [8], the authors used the taxonomy and the dataset of [7] and created a dataset of 130,000 annotated privacy policies. Using deep learning techniques, they developed 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.

We can derive from these works a number of generic privacy-oriented behaviors that applications may or may not implement. For example:

- **Encrypted data exchange**: The application transfers user data in encrypted form;
- **Authentication**: The application authorizes the data destination before sharing the data with it;
- **Third-party sharing**: The application shares data of type X with third party Y.

In this paper we assume that such privacy principles have been extracted from the application's privacy policy, and focus specifically on verifying third-party sharing principles.

B. 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 [10]. 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's behavior by intercepting requests to the Android access control system [11]. 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 work we are interested in other types of applications that are hosted in cloud or fog computing platforms. We therefore aim to develop generic techniques that may be applied to a wide range of applications. Moreover, we prefer avoiding having to annotate the application or the host platform to circumvent the performance penalty and provide independence from the platform.

C. 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 [12]. The emergence of port obfuscation, port forwarding, protocol embedding and random ports assignments greatly reduce the accuracy of this approach.

A popular alternative is deep packet inspection (DPI) techniques which essentially match 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 [13]. However, allowing a platform to inspect the application's payload which potentially contains personal data, may in itself violate most privacy policies. Also, DPI cannot handle scenarios where network traffic is encrypted or where the protocol is deliberately obfuscated [14].

With the proliferation of applications which encrypt their network traffic and users who utilize Virtual Private Networks, 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, Brissaud *et al* use random forest classifiers to determine if an HTTPS request matches a pre-defined type of action [15].

Machine learning is also being used for network traffic classification and characterization [14]. 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 [16]. Similarly, Deep Packet uses convolution neural networks to classify encrypted network traffic in broad classes such as FTP vs. P2P [17].

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 [16] and [17] identify streaming traffic but they do not distinguish audio from video streaming. This information is important in our case 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.

In this paper, we exploit similar machine-learning-based techniques, and apply them to the new domain of extracting a privacy-oriented behavior (which has not been studied yet according to our best knowledge).

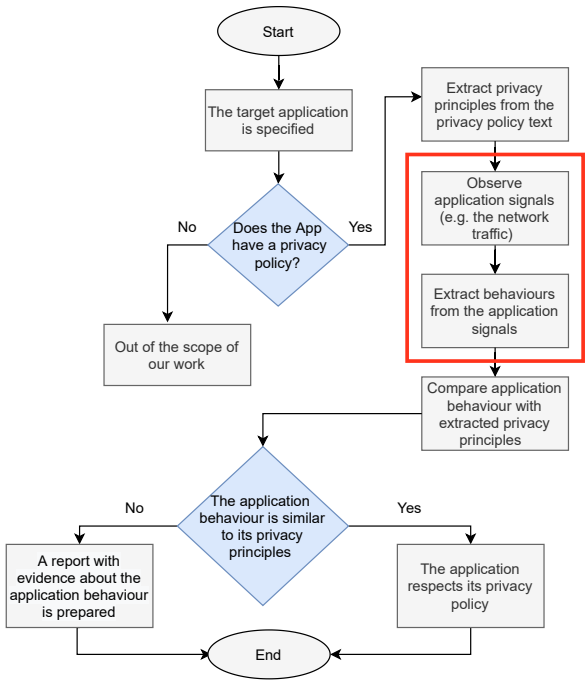

Fig. 1: Flowchart of our proposed privacy verification model.

III. 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 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 paper 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.

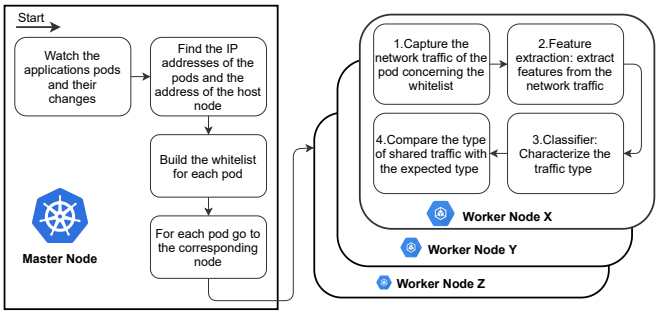

Fig. 2: High-level view of our system implementation.

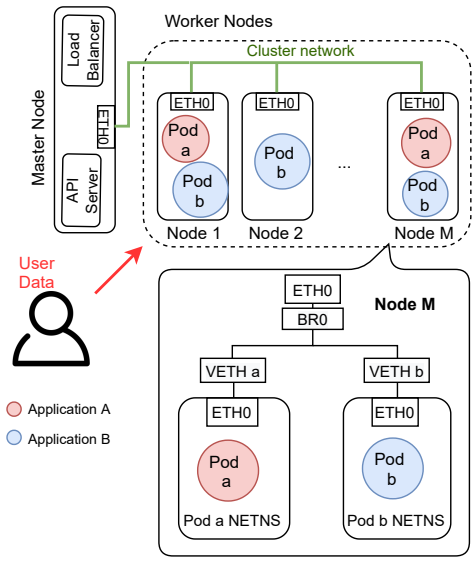

Fig. 3: A simplified version of Kubernetes networking inside one worker node.

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 [18]. 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 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 2 presents a high-level view of our implementation for the two highlighted steps from Figure 1.

A. *Application 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 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 [19], [20].

K8s is an open-source, vendor-neutral container orchestration platform which automates the deployment, resource management and scaling of applications in distributed computing structures. K8s is installed on a collection of worker nodes which form a cluster, and which are managed by a master node. Each K8s cluster has one master node and multiple worker nodes.

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. 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*. 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 [21] 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.

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
```

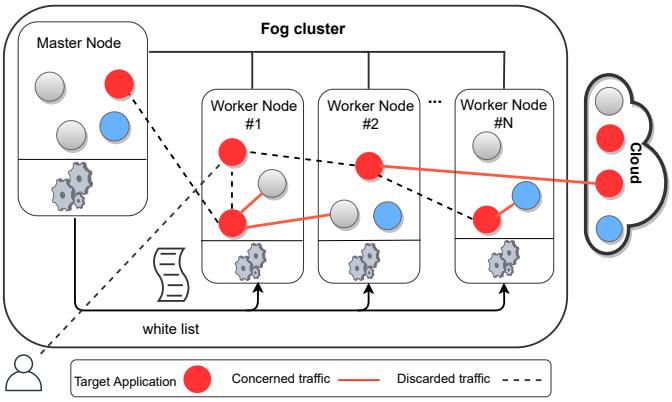

Fig. 4: Whitelisting some of the network traffic in the platform. Pods of the target application are specified with red color, other applications with grey and blue.

To do this, we rely on the way Kubernetes networking is organized (see Figure 3). 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. 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 [22].

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 4, 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.

B. Application 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.

1) 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.

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 [23]. 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 I 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 Section IV how we further refine this initial list to select the most relevant features in our models.

2) Application data type detection: When an application is being monitored, we capture traffic samples from this

TABLE I: 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

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 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.

IV. Experimental results

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.

A. 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 [16] 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 [24]. 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 II 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.

B. 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 [25]:

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

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.

C. 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 classification 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

TABLE II: Dataset for traffic data type.
Each sample includes the features extracted from 210 network packets of the application.

App. type	Data type	Application Name	# of samples	Application 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 (www.latextemplates.com, www.bensound.com and Youtube)
		curl	47	
		w3m	55	
Other	Other	firefox	47	Manual browsing
		Slack	51	
		Trello	45	
Total			**600**	

TABLE III: 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**

TABLE IV: 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**

D. 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 III 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 IV. 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.

V. Discussion

This paper demonstrates the potential of automatic mechanisms to assess whether an application truly respects its own privacy policy. However, a number of additional research directions remain to be explored before a practical implementation of this idea becomes feasible.

a) Other application signals and privacy-oriented behaviors: This study introduces a model for automated privacy compliance checking of applications. We demonstrated how network traffic may be used as an application signal for detecting data sharing behavior. Our work currently focuses on a single privacy-oriented behavior (data sharing), and 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 issued system calls. 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 online or offline and it may be performed continuously or based on randomly selecting an application for a limited time period.

b) Malicious applications: Due to the GDPR and similar regulations enforcement, 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. Malicious applications may develop strategies to mislead the monitoring system. For instance, an application may intentionally produce misleading signals to

evade detection of incorrect behavior. This may be considered as a security challenge rather than a privacy protection one, and is out of the scope of this paper.

c) 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 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 their users that their applications are actually being monitored using an unmodified version of the monitoring platform [26].

d) Finer-grained classification: In contrast to similar papers 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. Although we obtained promising performance results, we should consider the fact that our dataset size remains fairly small. More research based in larger data sets remains necessary to classify traffic samples in larger numbers of application behavior classes.

VI. Conclusion

This paper introduced 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 demonstrated 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 [24].

In our future research we intend to focus on designing a more comprehensive monitoring system based on this proposed model. This system will be capable of observing various application's signal on the platform to deduce the privacy-oriented behaviors from these observations. More research into using other application's signal rather than network traffic would be of interest. Combining the application's signal to improve the findings about the application's privacy-oriented behaviors would be another interesting research line. Further study on using the available tools to extract privacy principles from the privacy policy is also needed.

References

[1] EU Parliament, "Regulation (EU) 2016/679 of the European Parliament," 2016, https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1590072813969&uri=CELEX:32016R0679.

[2] M. Hatamian, "Engineering privacy in smartphone apps: A technical guideline catalog for app developers," *IEEE Access*, vol. 8, 2020.

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

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

[5] A. Ahmed and et al., "Fog computing applications: Taxonomy and requirements," *CoRR*, vol. abs/1907.11621, 2019. [Online]. Available: http://arxiv.org/abs/1907.11621

[6] M. Farhadi, J.-L. Lanet, G. Pierre, and D. Miorandi, "A systematic approach towards security in Fog computing: assets, vulnerabilities, possible countermeasures," *Software: Practice and Experience*, vol. 50, no. 6, 2020.

[7] S. Wilson and et al., "The creation and analysis of a website privacy policy corpus," in *Proc. ACL*, 2016.

[8] 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*, 2018.

[9] 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.

[10] W. Enck and et al., "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, vol. 32, no. 2, 2014.

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

[12] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. ACM CoNEXT*, 2006.

[13] T. J. Parvat and P. Chandra, "A novel approach to deep packet inspection for intrusion detection," *Procedia Computer Science*, vol. 45, 2015.

[14] 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, no. 2, 2018.

[15] P.-O. Brissaud, J. François, I. Chrisment, T. Cholez, and O. Bettan, "Transparent and service-agnostic monitoring of encrypted web traffic," *IEEE Trans. on Network and Service Management*, vol. 16, no. 3, 2019.

[16] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *Proc. ICISSP*, 2016.

[17] 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, no. 3, 2020.

[18] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, 2016.

[19] J. Santos, T. Wauters, B. Volckaert, and F. Turck, "Towards network-aware resource provisioning in Kubernetes for fog computing applications," in *Proc. IEEE NetSoft*, 2019.

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

[21] "ksniff for kubernetes," https://github.com/futuretea/ksniff.

[22] B. Boucheron, "How to inspect Kubernetes networking," DigitalOcean, 2018, https://do.co/2VEMxBs.

[23] O. Mula-Valls, "A practical retraining mechanism for network traffic classification in operational environments," Master's thesis, Universitat Politecnica de Catalunya, 2011.

[24] M. Farhadi, "Application specific network traffic with specified activities," Jul. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3965834

[25] Wikipedia, "F1 score," https://en.wikipedia.org/wiki/F1_score.

[26] K. R. Jayaram and et al., "Trustworthy geographically fenced hybrid clouds," in *Proc. ACM/IFIP/USENIX Middleware*, 2014.