Rafa Gálvez*, Veelasha Moonsamy, and Claudia Diaz

Less is More: A privacy-respecting Android malware classifier using federated learning

Abstract: In this paper we present LiM ('Less is More'), a malware classification framework that leverages Federated Learning to detect and classify malicious apps in a privacy-respecting manner. Information about newly installed apps is kept locally on users' devices, so that the provider cannot infer which apps were installed by users. At the same time, input from all users is taken into account in the federated learning process and they all benefit from better classification performance. A key challenge of this setting is that users do not have access to the ground truth (i.e. they cannot correctly identify whether an app is malicious). To tackle this, LiM uses a safe semi-supervised ensemble that maximizes classification accuracy with respect to a baseline classifier trained by the service provider (i.e. the cloud). We implement LiM and show that the cloud server has F1 score of 95%, while clients have perfect recall with only 1 false positive in $>$ 100 apps, using a dataset of 25K clean apps and 25K malicious apps, 200 users and 50 rounds of federation. Furthermore, we conduct a security analysis and demonstrate that LiM is robust against both poisoning attacks by adversaries who control half of the clients, and inference attacks performed by an honest-but-curious cloud server. Further experiments with MaMaDroid's dataset confirm resistance against poisoning attacks and a performance improvement due to the federation.

Keywords: Android, malware detection, privacy, poisoning, integrity, federated learning, semi-supervised, machine learning

DOI 10.2478/popets-2021-0062
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

Android dominates the mobile operating system market as the most popular choice amongst smartphone users. At the same time, this makes Android an attractive target for malware authors who want to infect as many devices as possible with malicious applications. Malware classifiers that leverage *machine learning* (ML) techniques have been proposed to tackle this problem, showing varying degrees of success [4, 12, 19, 20, 23, 25]. Often, to produce accurate classification results, machine learning models rely on access to a large and diverse set of features collected from user devices – which can be very revealing of the list of apps installed in each device [5, 25, 32]. This raises privacy concerns: these models expose all users' private data to the centralized entity performing the classification, which may learn private information about the user; and motivates the need for decentralized, privacy-respecting malware classifiers for Android that can effectively protect users from malware infections without requiring them to expose private information to third parties.

ML solutions for malware classification can be grouped in three categories: (i) cloud-based, (ii) client-based and (iii) hybrid, i.e. a combination of (i) and (ii). In cloud-based solutions [23, 32], the ML models are supplied with large sets of features that implicitly reveal users' installed apps, which constitute potentially sensitive data. In client-based solutions [4], data is protected as the ML models are local and compute predictions on the device itself. This is however a resource-consuming approach that results in high false positives, as the models are not trained with sufficient new data as time goes by [25]. Lastly, in hybrid solutions [5, 25] apps that are flagged as suspicious on the device are sent to the cloud for further analysis. Such solutions have a high number of false positives in the local models and reveal significant private data to the cloud.

In general, it cannot be denied that ML algorithms are effective at detecting malware. However, of notable concern is the amount of information required for the ML models to produce good results and the associated repercussions on users' privacy. In particular, it can be observed that the larger the number of raw features

***Corresponding Author: Rafa Gálvez:** imec-COSIC KU Leuven, E-mail: rafa@kuleuven.be
Veelasha Moonsamy: Ruhr University Bochum, E-mail: email@veelasha.org
Claudia Diaz: imec-COSIC KU Leuven, E-mail: claudia.diaz@esat.kuleuven.be

centrally available for training and testing, the better the classification results. Additionally, as demonstrated by Song et al. [27], ML models can memorize and leak detailed information about training datasets. This observation coupled with the fact that the list of apps installed on a mobile device may enable private inferences about the users' personal preferences, profile, behavior, etc. pose a serious threat to their privacy.

To address the aforementioned concern we investigate the following key question: *How can we build a decentralized Android malware classifier that is privacy-preserving?*

In this paper, we present LiM – a hybrid solution that leverages the power of Federated Learning (FL) to provide a malware detection solution that has high classification performance while respecting user privacy. LiM can classify all the apps installed on users' devices regardless of whether they are obtained from an app store or other sources, allowing users to detect malware without relying on the hosts of app stores (e.g. Google for Play Store) for malware classification services. LiM reduces the dependency of users on app stores in a way that benefits both privacy and malware detection performance.

State of the art FL models [22, 30, 31] allow users to keep their testing data locally while the learning process is done collaboratively to improve performance, i.e. users train their client models with a supervised algorithm, while a service provider aggregates the parameters of all models. LiM extends the traditional FL technique to the safe semi-supervised ML paradigm [11], enabling the application of FL in settings where users do not have local access to ground truth, as is the case with malware classification. Safe semi-supervised models combine labeled data available to the cloud server with unlabeled data available to clients. The cloud server trains fully-supervised models and shares them with clients, who in turn retrain with their unlabeled local data without introducing a performance penalty.

We validate the design by implementing a prototype of LiM and evaluating its performance and its security against poisoning and inference attacks. We carry out experiments using a dataset of 25K malware apps and 25K clean apps, simulating federations of 200 clients over 50 rounds. The results show that the cloud can reach 95% F1 score, and clients has as few as 1 false positive. Additionally, faced with a strategic adversary that controls 50% of the clients and whose goal is to perform a poisoning attack, the remaining honest clients are able to correctly identify the targeted, poisoned app. Moreover, we demonstrate that the cloud server is unable to infer whether a specific app was installed by any of the clients, making LiM resistant against membership inference attacks. Finally, we validate LiM's capability to learn from the federation using MaMaDroid's dataset [23], as well as its resistance against poisoning attacks.

Our contributions:

1. We present a first, comprehensive design and implementation of a privacy-respecting Android malware classifier.
2. We demonstrate an effective way to combine FL and safe semi-supervised ensemble learning to enhance malware detection accuracy and privacy at the same time.
3. We conduct a security analysis to illustrate the robustness of LiM against poisoning and inference attacks.
4. In the spirit of open science, we make our code available at https://git.sr.ht/~rafagalvez/lim-python.

Terminology. To improve readability of the remaining sections, we provide our working definitions of key terms used throughout the paper.

- **Client** refers to the ML model that resides locally on the user's mobile device.
- **Cloud** refers to the global ML model which is present at the service provider's side.
- **SAFEW** is an ensemble classifier composed of a set of individual base learners and their weights.
- **Baseline classifier** is a standalone classifier whose performance has to be met by all base learners (combined) as a lower bound; also referred to as *baseline*.

Roadmap. The rest of the paper is organized as follows: in section 2, we provide background knowledge about Federated Learning, semi-supervised learning and Android malware classification. Section 3 describes the threat model of LiM. In section 4, we elaborate on safe semi-supervised FL and how it is implemented in LiM. Section 5 provides the details of the LiM architecture and its associated building blocks, followed by the empirical results and security analysis in section 6. Section 7 presents a discussion based on the empirical results and avenues for future work together with relevant related work in section 8 and concluding remarks in section 9.

2 Background

Federated Learning (FL), also referred to as 'Collaborative Learning', is a technique where an ML algorithm is iteratively trained in a distributed setting with a client-server architecture [13]: a large number of clients contribute locally-computed learning model parameters to a service provider (e.g. a cloud server) that aggregates those client parameters to compute updated parameters for the federated model, which is in turn sent back to clients for the next iteration. Each iteration, or *round*, improves the performance of the client models thanks to 1) the newly available local data that can further refine training, and 2) the aggregated parameters shared by the cloud server, which improve the model based on learning done by the ensemble of all clients.

FL offers privacy advantages compared to purely centralized models where the service provider collects raw input data from all clients in order to train the model. FL models keep data local to the clients and instead share model parameters. However, the client-server architecture creates opportunities for adversaries that compromise a fraction of clients or the service provider to impact performance and privacy guarantees. On the one hand, poisoning attacks [6] may harm the performance of the federated model if adversarial clients submit maliciously crafted parameters to the server. On the other hand, client-provided parameters may be exploited by a curious server that conducts inference attacks [18] to identify the training examples used by clients. In this paper, we tackle both threats by making use of safe semi-supervised learning to: 1) enable the cloud server to exclude poisoned parameters by using locally available data, and 2) allow clients to share hyper parameters rather than the parameters of individual classifiers, which cannot be exploited to infer training inputs.

LiM uses FL for malware classification. One limitation of current FL solutions is that they rely on supervised learning, requiring clients to have access to ground truth for the local training process [11]. While applications such as predictive typing can benefit from this approach (since the client can locally test predictions against actual inputs from the user), this assumption does not hold in malware classification use cases, where clients do not have local knowledge of the ground truth.

LiM solves this problem using a safe semi-supervised learning algorithm that allows clients to use unlabeled data for training, while guaranteeing that the performance will be at least as good as that of a baseline classifier.

2.1 Safe semi-supervised learning

Semi-supervised learning (SSL) uses both labeled and unlabeled data to train a classifier. Labeling data can be expensive, while collecting and learning labels from raw data has become easier with the commoditization of internet access and the plethora of apps installed on smartphones. One of the main challenges for the success of an SSL algorithm is to ensure it indeed learns useful information from unlabeled samples, as there is no ground truth for the algorithm to verify its predictions for those samples.

Safe SSL addresses this concern by ensuring that a minimum baseline performance is always achieved, i.e. that unlabeled information does not worsen the performance of another (fully supervised) classifier. A well-performing strategy to achieve safe SSL is to use an ensemble of learners that, combined through a set of learned weights, are likely to outperform the baseline model [15, 16].

An example of this kind of classifier is SAFEW [15]. Its goal is to combine a set of classifiers, called *base learners*, through a set of weights $\alpha_i, i \in [1, b]$ that leads the ensemble to perform always better than a given baseline classifier. SAFEW achieves this goal assuming the ground truth label assignment of the unlabeled data f^* can be realized as a convex combination of the predictions from the base learners b, i.e. $f^* = \sum_{i=1}^{b} \alpha_i \mathbf{f}_i$ [15]. Using this assumption, SAFEW can compute the error of the baseline classifier $\mathbf{f}_0$ and final $\mathbf{f} \in \{+1, -1\}$ predictions with respect to the ground truth f^* using the loss function l, and make $l(\mathbf{f}, f^*)$ as small as possible compared to $l(\mathbf{f}_0, f^*)$ even in the worst case, maximizing the minimum difference as shown in equation 1.

$$\max_{\mathbf{f} \in \{+1,-1\}} \min_{\alpha} l(\mathbf{f}_0, \sum_{i=1}^{b} \alpha_i \mathbf{f}_i) - l(\mathbf{f}, \sum_{i=1}^{b} \alpha_i \mathbf{f}_i) \qquad (1)$$

SAFEW does not impose restrictions on which classifiers to use as base learners to predict $\mathbf{f}_i$: results do not depend on the amount of base learners but on their quality, and different learning algorithms can be used, for instance Support Vector Machines together with Random Forests. Similarly, there is no a priori restriction on the loss function that can be used as l. However, to reduce computation time and find optimal weights α^*, Li et al. [15] show that using the hinge loss function turns equation 1 into a convex optimization problem that can be solved with common optimization packages like the ones supported by CVXPY [1, 8]. Prior knowledge can be embedded as constraints in this problem formulation in order to enhance the information extracted from the unlabeled data. The final predictions $\bar{f}$ can then be obtained through equation 2.

$$\bar{f} = \sum_{i=1}^{b} \alpha_i^* \mathbf{f}_i \tag{2}$$

The solution proposed by SAFEW assumes a centralized setting where labeled and unlabeled data rest in the same place. Such scenario requires users to share data with the service provider, potentially leaking sensitive information. LiM addresses this gap by federating SAFEW, letting users keep their data local while sharing only the weights that enable SAFEW to improve performance over baseline. Moreover, LiM also aims to be resistant against membership inference and poisoning attacks, addressing the increasingly important security and privacy concerns that arise from the use of advanced machine learning in the wild.

2.2 Android

2.2.1 Android manifest file

In the Android OS, apps are distributed as Android application package (APK) files. These files are simple archives which contain bytecode, resources and metadata. A user can install or uninstall an app (the APK file) by directly interacting with the smartphone. When an Android app is running, its code is executed in a sandbox. In theory, an app runs isolated from the rest of the system, and it cannot directly access other apps' data. The only way an app can gain access is via the mediation of inter-process communication techniques made available by Android. These measures are in place to prevent the access of malicious apps to other apps' data, which could potentially be privacy-sensitive.

Since Android apps run in a sandbox, they are subject to restrictions on the usage of shared memory and most system resources. The Android OS provides an extensive set of Accessible Programming Interfaces (APIs) that allows access to system resources and services. In particular, the APIs that give access to potentially privacy-violating services (e.g., camera, microphone) or sensitive data (e.g., contacts) are protected by the Android Permission System [7]. Developers have to explicitly mention the permissions that require user's approval in the `AndroidManifest.xml` file (hereon referred to as the Manifest file).

Besides permissions, the Manifest file also includes information about the app components [9], such as activities, services, broadcast receivers and content providers. An *activity* is the representation of a single screen that handles interactions between user and apps. *Services* are components that run in the background of the OS to perform long-running operations while a different application is running in the foreground. *Broadcast receivers* respond to broadcast messages from other applications or the system. They allow an app to respond to broadcast announcements outside of a regular user flow. A *content provider* manages a shared set of app data and stores them in the file system. It also supplies data from one app to another on request.

It is worth noting that the information present in a Manifest file is not obfuscated and can be extracted via static analysis. It is in the app developer's best interest to not obfuscate the file as it would result in breaking the functionalities of the app, rendering it useless. In section 6, we provide further details about the features used by our proposed classifier, LiM, to conduct malware detection.

2.2.2 Android malware classification

There are several proposals for ML classifiers that can detect malicious APKs targeting Android. We divide them in three categories: centralized, local and hybrid.

Centralized approaches use a cloud classifier to predict if an app is malicious or clean. Cloud-based approaches can accurately predict big testing datasets thanks to the advanced feature engineering a cloud infrastructure can handle [23]. Both static and dynamic analysis can be performed, for e.g. such as taking into account the API call graphs of the apps and behavioral characteristics of an app during execution.

Local approaches install an already-trained classifier on the user's device. Due to the constrained resources available to the classifier, the feature set and the detection algorithm must be considerably more lightweight than in centralized approaches [4]. Lightweight dynamic analysis can be performed together with static analysis (for e.g. features from the Manifest file).

Hybrid approaches combine local and cloud models. A first screening of the app is performed on the client device itself using a lightweight feature set, and if necessary more features are collected and sent to the cloud to verify the prediction [5, 25].

Existing centralized and hybrid solutions achieve good performance but expose user inputs to the service provider, enabling potentially sensitive inferences. Local approaches do not expose client data, but suffer from poor performance. In constrast, LiM's performace is comparable to that of centralized and hybrid solutions, while its privacy protection is as in local approaches, obtaining the best of both worlds.

3 Threat model

LiM's functionality is to classify malware in a privacy-preserving manner, using base learners trained with data that never leaves the client device and yet contributes to improving the accuracy of classification for all clients. In our threat model, we distinguish two types of adversaries depending on their attack goals:

1. The goal of *Adversary 1* is to compromise **integrity**: poison the federated learning with maliciously crafted inputs in order to trigger specific apps to be misclassified, as presented in [6].
2. The goal of *Adversary 2* is to compromise **privacy**: infer private information about the training data of clients (list of installed apps), as presented in [18].

Adversary 1 targets the core functionality of the malware classifier, i.e. bypass the malware detection mechanism. To achieve that, the adversary modifies the classification model so that it misclassifies a specific malicious application. In a federated setting, the adversary may control a subset of users (at most 50%) who submit maliciously crafted model parameters to the federation. Further, we assume that the adversary controls the malicious application, which can be tweaked so that its features resemble clean apps and confuse the model into misclassification. In terms of trust model, we assume that non-malicious users compute and submit correctly computed parameters, and that the cloud server correctly follows the learning process.

Adversary 2 aims to compromise user privacy by inferring information about the apps that users have installed on their devices, e.g. the app names, categories, usage patterns, etc. In a federated setting, we are interested in a passive global adversary, as described in [22], that has access to the cloud server's data including all client model parameters uploaded to the cloud. The adversary examines these individual client models to try to infer information about which apps users have installed. Note that LiM relies on information that is statically extracted from app manifests and is thus only concerned with app installs, while inferences on app usage after installation is out of scope.

Both adversaries have white-box access to the cloud model (including architecture, feature set and hyperparameters) in each federation round. Adversary 2 also has white-box access to the models of all clients in each round, while Adversary 1 does not know the hyperparameters of the models of honest users.

4 Safe semi-supervised federated learning

Traditionally, FL employs a decentralized approach to train a neural model. Instead of uploading data for centralized training, clients process their data locally and share the resulting model updates with the service provider. Such distributed approach has been shown to work with unbalanced datasets and data that is not independent or identically distributed across clients.

FL's success is, however, dependent on properly labeled data that can be used to train supervised learning models. Considering the scenario of malware classification, we cannot rely on users assigning correct labels to their data, as it cannot be guaranteed that they can correctly identify malicious apps.

LiM proposes a novel approach to address this challenge: a semi-supervised method that allows FL to train local models without supervision. We assume that labeled data is only available to the cloud, while clients use their unlabeled samples to update the parameters of their local semi-supervised model.

Furthermore, we leverage the practical benefits of safe semi-supervised learning (SSL), as described in section 2.1, to ensure that models trained by the clients are useful, i.e. that they provide a minimum baseline performance and do not introduce noise (via incorrect labels) in the federated model.

In LiM, the federation happens across the weights of the base learners, which clients estimate using their unlabeled testing datasets. The cloud server collects all client weights and aggregates them in a similar fashion as it would do with the weights of e.g. a deep neural network (DNN). It is important to note that the number of base learners is much lower than the number of neurons in a DNN – LiM further compresses client data by taking advantage of the training process that the cloud performs on the base learners. We see this feature

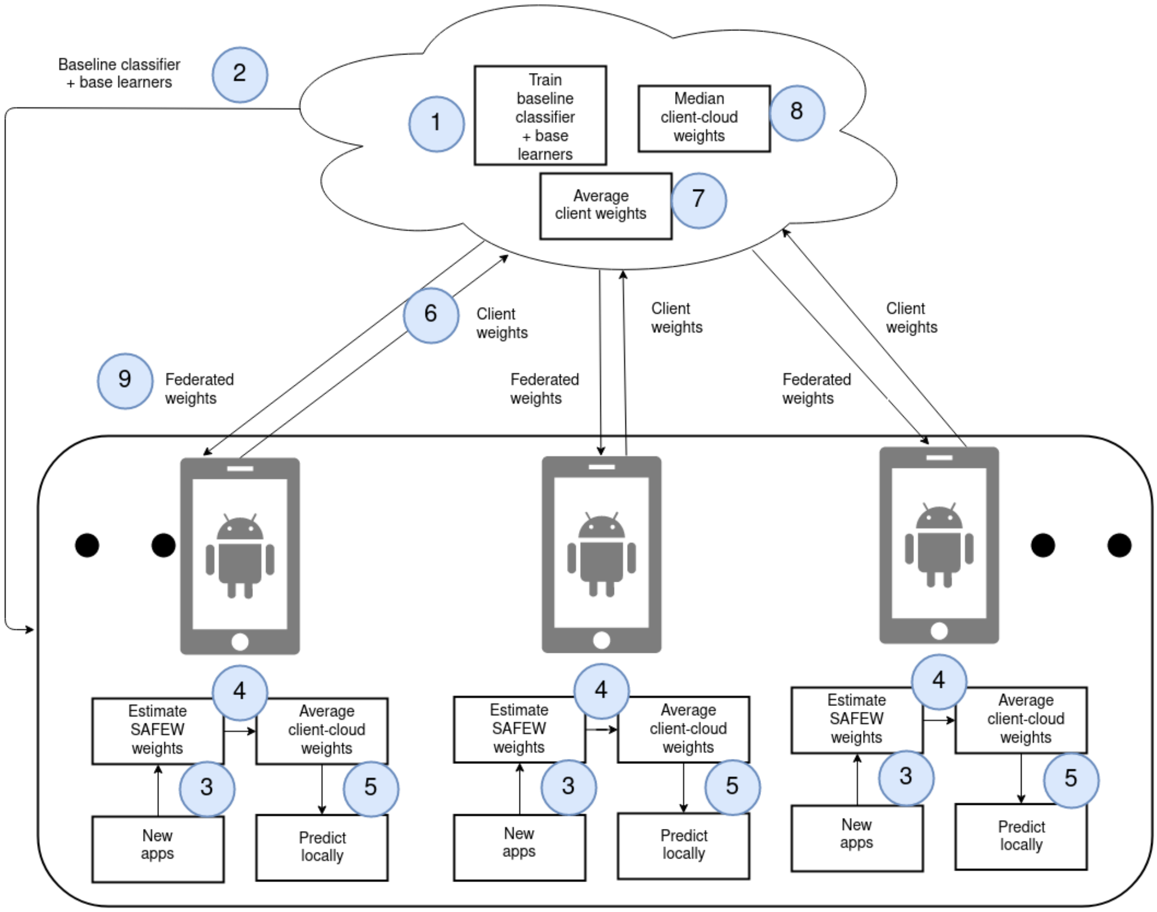

Fig. 1. LiM architecture – the cloud and the clients all train their own SAFEW model, and weights are aggregated twice to distribute the information of the testing datasets of the users.

as a defense mechanism against *privacy attacks* (cf. section 3), as client updates will not be sparse anymore.

Note that in the case of LiM, the service provider plays a greater role than in the classical, supervised FL to compensate for the lack of ground truth in the clients. The provider also selects the architecture and the feature sets of the different base learners, as well the one to be used as baseline classifier. LiM provides protection against *integrity attacks* (cf. section 3) by comparing the weights contributed by clients with those generated by the provider through its own unlabeled dataset.

5 LiM architecture

We assume that the service provider (cloud server) has access to a ground truth (labeled) dataset of malware and clean apps, and a testing (unlabeled) dataset. At the client-side, users locally scan their installed apps to identify malware. LiM can be incorporated in the package installer of an Android OS and executed as a privileged background service. LiM uses the SAFEW classifier described in section 2.1 to implement the scheme explained in section 4.

Round 0 of FL: The process is depicted in figure 1. First, the server uses (step 1) the labeled dataset to train a baseline classifier and a set of base learners, and the unlabeled dataset to estimate weights for the base learners. Clients receive (step 2) the baseline classifier and trained learners in order to estimate (step 3) their local SAFEW weights using their own testing dataset (i.e. their installed apps). Clients then compute average weights (step 4) and use them to classify their installed apps (step 5). Users then complete the federation round by sending their locally computed weights to the cloud (step 6). The cloud averages all the client weights (step 7) and then further averages that value (step 8) with the weights of its own SAFEW, computed in step 1. Finally, the cloud sends the updated federated weights to the clients to initiate a new round of federated learning (step 9).

Round 1 of FL (and beyond): Once Round 0 is completed, steps 1-2 are optional: the service provider can re-apply them at any point in the subsequent federation rounds. In our evaluation, we did not conduct experiments that consider such updates. Steps 3-9 are applied as described in Round 0, even if some users do not install new apps. We account for this possibility in our evaluation.

Note that in LiM, the service provider does not send its own weights in the initial round (i.e. Round 0) to prevent an adversary from using this information to craft an integrity attack (cf.section 3), and to give clients' inputs greater influence in the federated weights. In later rounds, new federated weights are computed by using both client and cloud weights. The provider computes the mean of all clients' weights and averages that with its own locally computed weights. The design behind this construction is purposefully conservative to counter integrity attacks where malicious inputs degrade overall classification accuracy – guaranteeing the key functionality of a decentralized malware classifier.

LiM uses SAFEW as the local classifier that clients train in order to predict if their installed apps are malicious. To take advantage of the optimizations explained in section 2.1, the individual SAFEWs use the hinge loss function to estimate their weights from unlabeled data. Regarding the choice of SAFEW base learners, it is possible to choose any base classifier as the optimization algorithm of SAFEW only uses the predictions of base learners to compute their weights. There are two main criteria to keep in mind when selecting alternative learners:

– They must provide a reasonable performance on their own, e.g. over 90% F1 score.
– Their predictions must complement each other, i.e. the learners must be heterogeneous. If there is one clearly strong learner, SAFEW will just always copy its predictions (i.e. its weight will be 1).

Additionally, domain knowledge can guide the aforementioned selection process, and it is possible to further constrain the set of possible weights α_i (cf. section 2.1) to reflect, e.g. the confidence that the designer has on each learner relative to others. It is, however, important to note that LiM does not need a lot of expert knowledge for its setup. Weights can be learnt from data without any prior domain knowledge, and there are no assumptions on the distribution of the testing dataset, i.e. no prior knowledge on the proportion of samples per class.

In section 6, we compare the performance of standard learners to find which combination is the most effective for malware detection.

6 Evaluation of the system

We empirically evaluate LiM as a federated malware classifier in a setup with a server and 200 clients iterating over 50 federation rounds. In our evaluation, the clients run as a parallelized (across clients) Python program on a 4 Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz cores using 7.5 GiB of RAM. The goal of the evaluation is to show how LiM performs using different configurations for the individual SAFEWs, as well as its evolution across rounds with respect to the different baseline learners and local (i.e. non-federated) SAFEWs.

Specifically, we set up LiM to use the following learners: k Nearest Neighbours with number of neighbors $n = 3(kNNn3)$, Logistic Regression with regularization parameter $c = 1(LRc1)$, Random Forests with number of decision trees $n = 50(RFn50)$, $100(RFn100)$ and $200(RFn200)$, and Linear SVM with regularization parameter $c = 1(SVMc1)$. All base learners are used in each experiment, rotating the baseline classifier role across them. Clients only use configurations that the cloud has vetted as safe, i.e. where the cloud SAFEW outperformed or matched the performance of the baseline classifier.

We perform 4 experiments per configuration using the Top 100, 200, and 500 features (i.e. 12 experiments per configuration). Half of the experiments are done with 50% of adversarial clients to evaluate resilience against integrity attacks.

The rest of this section describes the datasets used and the results of the performance and security evaluation of LiM.

6.1 Datasets

We use the AndroZoo dataset [3] to obtain 25K clean apps, which were selected from the top 3 most popular stores (Anzhi, Appchina and Google Play Store) as of October 2018. As for the malware samples, we collected 25K samples from the Android Malware

Table 1. Number of features per feature types. In the Top 500, there are 55 features that belong to two categories: 54 of them can be declared as permissions or hardware components, the 55th (com_facebook_facebookcontentprovider) can be either an activity or a content provider.

Types of features	Number of features		
	Top 100	Top 200	Top 500
declared permissions	65	73	97
activities	19	80	232
services	3	13	71
intent filters	0	0	0
content providers	1	3	10
broadcast receivers	6	25	85
hardware components	24	24	24

Genome project [33] and the Android Malware Dataset project [17, 28]. We pick the latest version of apps, removing duplicates within the same store. It is possible that the same app is published in two different stores as different versions, but we consider they are effectively different apps as developers may include different functionality for particular stores.

For each app, we extract the features from the Manifest file, thus following the recommendation proposed by the authors of Drebin in [4] – a key related work in Android malware detection and feature extraction. While dynamic analysis can provide greater performance, it is both more resource intensive and easily obfuscated than static analysis. We then transform the statically extracted features into a vector of numerical binary values indicating the presence of a feature (e.g. a specific permission) in the Manifest file of an app. Finally, out of 370K features we select the top 100, 200, and 500 features ranked according to their chi-squared scores with respect to the ground-truth class. Table 1 depicts the number of features and their corresponding feature types. To maintain consistency, we use the same categories as presented in [4].

Out of the 50K apps, we randomly sample a training set of 10K apps for training the baseline and base learners. The cloud SAFEW testing set has 32K apps, and the overall client testing set has 8K apps. In order to facilitate the learning process between cloud and clients, both testing data sets have an overlap of ~1K apps.

We simulate several rounds of federation as described in section 5. In the first round, clients have a set of 96 preinstalled apps, which have been extracted from an Android Pie emulator and whose manifests have at least one permission. In later rounds, the clients install up to 5 apps drawn randomly from the client testing dataset, using a binomial distribution with bias 0.6 to randomize the number of apps. Each app will be a malware sample with probability 0.1. Moreover, to model the fact that users install popular apps much more frequently than others, we create two sets of 50 apps based on the presence of popular features in the malware and clean datasets, and make clients draw apps from these sets with probability 0.8.

6.2 Performance evaluation

6.2.1 Performance metrics

To evaluate LiM in a realistic setting where users encounter many more clean apps than malicious ones (i.e. where classes are highly imbalanced) we use the F1 score, computed as $F1 = 2 * (precision * recall)/(precision + recall)$. Precision measures how many positive predictions (true positives + false positives) were actual positives (true positives), while recall measures how many actual positives (true positives + false negatives) were classified as positive (true positives). Since users typically install many more clean apps than malicious apps, it is easy for LiM to achieve high recall by predicting many positives at the expense of precision, which is not accounted for in other popular metrics like accuracy.

In order to better understand this balance between precision and recall across SAFEW configurations, we also report the raw number of false positives. We argue that, irrespective of whether the F1 score of two versions of LiM using different configurations of base learners is very similar, end users are rather sensitive to small differences in the number of false positives. More concretely, if a malware classifier repeatedly flags clean apps as malicious, then this will significantly affect user experience.

6.2.2 Performance results

In the cloud, we observe that LiM matches the F1 score of the centralized SAFEW most of the time, and that the number of false positives is the second lowest using RFn50 with 500 features (cf. table 5 from the appendix). Regardless of the configuration used, the F1 score remains around between 94% and 96%.

In figure 2, we can see how this performance gain comes from the reduction of false positives. The evolution shows that learning from the federation allows LiM to drop the number of false positives from 5 (same as SAFEW) to 3, whereas baseline KNN misclassify more clean apps as we advance through the FL rounds.

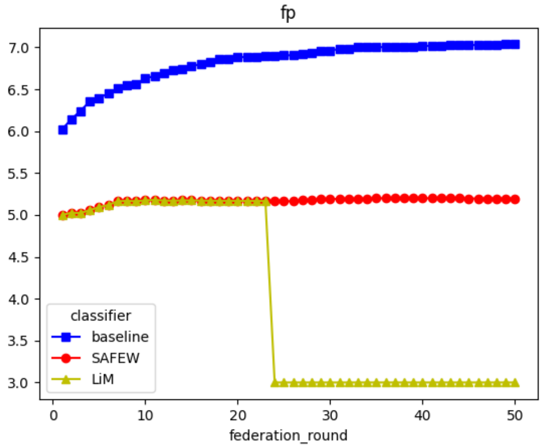

Fig. 2. False positives of 200 clients with 500 features, using kNN (n=3) as baseline classifier. LiM reduces the number of false positives to 3, while both SAFEW (i.e. no FL) and baseline have 5 and 7, respectively.

Table 2 shows the F1 score and the number of false positives (FP) of clients using different baseline classifiers and sets of base learners, and averaged across experiments. The best performance, i.e. F1 score of 73.7% is achieved using RFn50 as baseline classifier and 200 features, thanks to the low number of false positives. Using kNN as baseline classifier gives us a high F1 score of 73.2% using 200 features.

Figure 3 shows the evolution of LiM using KNN as baseline and 500 features. Round 24 of the federation brings a significant improvement in performance for LiM, and then the F1 score slowly grows similarly to how SAFEW (i.e. no FL) and baseline do. By round 50, LiM reaches close to an F1 score of 70%.

6.2.3 Run-time performance

The cloud and the clients compute their LiM predictions in the following three steps:

1. Build a local SAFEW. New weights are computed using testing data.
2. Average local and federated weights
3. Compute new predictions using the new weights

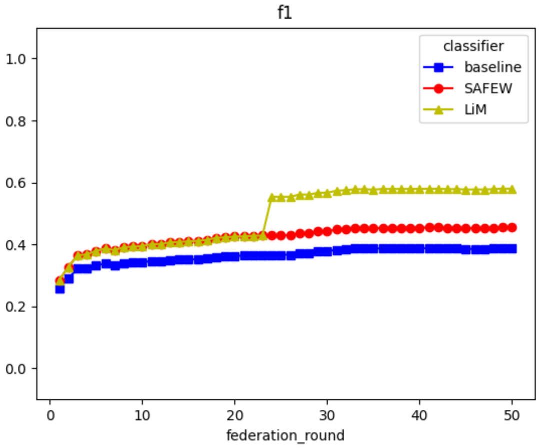

Fig. 3. F1 score of 200 clients with 500 features, using kNN (n=3) as baseline classifier. SAFEW (i.e. no FL) improves on baseline, and LiM improves on SAFEW.

For step 1, the cloud takes 13 seconds and an individual client, at the beginning of the federation, spends 0.1 seconds, due to the varying size of testing datasets. The cost of Step 2 is negligible. Step 3 however, takes almost the same time as step 1, i.e. step 1 spends only 1/10 of the time computing new weights, which in step 3 is not necessary. Thus, clients spent 0.2 seconds computing the LiM final predictions, while the cloud spent 26 seconds.

It is worth noting that we did not apply any optimization to the current implementation of LiM, as our goal for this paper was to show that it is possible to build an FL-based malware detection while maintaining users' privacy. For additional performance gain – at the very minimum, the predictions for base leaners from step 1 can be reused in step 3, making the total time closer to the time of step 1.

Regarding the training of the base learners, on average each model takes 12.7 seconds and the first cloud SAFEW prediction takes 13 seconds to complete.

6.3 Security analysis

In section 3 we introduced two adversaries: one that aims to poison the federation process in order to make a specific malware app be misclassified as clean (Section 6.3.1), and another that wants to learn which apps users have installed (Section 6.3.2).

Table 2. Comparison of client average performance across LiM, SAFEW, and different baselines. Experiments are carried out for 50 rounds of federation using the Top 100, 200 and 500 features which were selected based on the chi2 test.

Baseline	#Features	FP (raw values)			F1 (%)		
		Baseline	SAFEW	LiM	Baseline	SAFEW	LiM
KNN n3	100	8.9	5.6	2.4	36.4	46.5	66.5
	200	7.6	4.1	1.6	39.8	53.8	**73.2**
	500	6.7	3.6	3.0	37.9	55.2	58.5
LR c1	100	5.1	5.4	4.0	47.5	48.5	54.3
	200	3.6	3.6	3.6	55.2	55.4	55.0
	500	4.2	4.2	4.2	52.7	53.1	53.1
RF n100	100	2.8	5.5	2.9	64.3	50.2	63.5
	200	1.6	3.6	3.0	68.2	55.6	58.9
	500	1.6	2.2	2.2	72.2	66.7	66.2
RF n200	100	1.6	2.9	2.6	71.3	59.6	60.6
	200	2.8	2.3	2.3	65.4	69.1	69.1
	500	2.6	2.6	3.1	64.6	64.4	60.7
RF n50	100	2.7	5.2	3.2	64.0	44.7	57.2
	200	2.7	2.8	1.3	59.5	62.4	**73.7**
	500	2.1	4.2	2.6	63.8	48.3	60.2
SVM c1	100	5.8	5.4	5.4	44.6	46.2	46.2
	200	4.2	4.1	4.2	49.1	49.3	49.0
	500	3.7	3.7	3.7	50.0	50.0	50.0

6.3.1 Integrity attacks

The considered adversary participates in the federation by controlling 50% of clients, and attempts to bias the overall federation model by submitting maliciously crafted client models and malware apps. The end goal of the adversary is to trick LiM into misclassifying a malware app as clean.

To make the malware app undetectable to LiM (i.e. a false negative), the adversary modifies the weights submitted by malicious clients so that his `allied base learners` have an honest majority in the different layers of the federation. A base learner is an ally of the adversary if it classifies the malware app as clean. We assume there is at least one ally in the LiM configuration for each round in the FL process.

We formalize the problem in equations 3 through 7.

Let w be the honest weights of a malicious client, and w' the poisoned weights. The adversary's goal is to make the two types of weights as similar as they can be in order for the attack to be stealthy. Equation 3 expresses this goal.

$$\underset{w'}{\text{minimize}} \left\| w - w' \right\| \tag{3}$$

To maximize the chances of successfully poisoning the federation, the weights need to take into account multiple constraints. First, they must add up to one. Let b be the number of base learners; then the first constraint to the optimization problem is expressed in equation 4.

$$\sum_{i=1}^{b} w'_i = 1 \tag{4}$$

Second, for the compromised client to misclassify the targeted app, the weights of the classifiers that err in favour of the adversary must account for an honest majority. Let M be the indices of those allied classifiers in the SAFEW ensemble; then equation 5 expresses the `local constraint`:

$$\sum_{i=1}^{b} w'_i > 0.5, \{i \mid i \in M\} \tag{5}$$

Third, the cloud will average the weights of all the clients. Thus, the averaged weights of the allied classifiers must also hold an honest majority. Since the adversary does not have access to the weights of the honest clients, we can approximate it by averaging the `honest` weights of the malicious clients. Let w^c be the weights of the client c; then equation 6 expresses the `clients constraint`.

$$\frac{1}{N+1} \sum_{c=1}^{N} \sum_{i=1}^{b} (w_i^c + w'_i) > 0.5, \{i \mid i \in M > 0.5\} \tag{6}$$

Finally, the cloud will compute the federated weights by averaging its own weights with the average of the clients. Assuming the weights of the cloud are known, we can make the same honest majority across allied classifiers hold by approximating it with the guessed average in equation 6. Let w^* be these averaged weights and w^{cloud}, the weights of the cloud; then equation 7 expresses the `cloud constraint`.

$$\frac{1}{2}\sum_{i=1}^{b}(w_i^{cloud}+w_i^*)>0.5, \{i \mid i \in M > 0.5\}, \qquad (7)$$

Each round, the adversary has to try to solve this problem and submit the poisoned weights. If no solution is found, the problem is relaxed by first dropping the cloud constraint and then the client constraint. The adversary will always find a way to meet the local constraint.

All the malicious clients install the same app targeted by the adversary. We assume the adversary can partially modify the features of the app, which is crafted so that there is at least one allied base learner whose added weight(s) lie between 0.3 and 0.4. This range makes it possible for the adversary to succeed by redistributing up to 20% of the honest weights, a relatively small percentage.

We evaluate the effectiveness of this attack on LiM for different configurations of learners both at the cloud and at the clients, with special focus on the latter as that is the final target of the adversary. Our results show that the poisoning attack has virtually no effect in the LiM cloud, with the F1 score plateauing at around 94%–96%, regardless of the configuration (cf. table 6 from the appendix).

Table 3 compares the average client performance for honest LiM clients with baseline, SAFEW and adversarial (i.e. poisoned) clients. Overall, we see that the performance of LiM clients mostly outperforms baseline and SAFEW, and it is significantly higher than for adversarial clients. To better understand these results, we look at two specific configurations and analyze the effects of the attack over multiple federation rounds.

We first consider kNN with $n = 3$ and 200 features as baseline classifier. Figure 4 shows that LiM clients perform better than baseline and SAFEW. We can also see that client performance improves over time, with the most significant gains taking place in the first federation rounds. The figure also shows how the F1 score of the adversarial clients approximates that of SAFEW, suggesting that the adversary is indeed submitting poisoned weights that are not too different from honest weights. The actual number of false positives for this configuration is shown in figure 5. As can be seen in the figure, LiM clients have a small number of false positives compared to baseline, SAFEW and poisoned clients. Note that the adversary's goal is to trigger a false negative, and thus the number of false positives in adversarial clients can be minimized to behave as close to an honest client as possible. Due to the small number of malware apps installed by clients, a small increase in the number of false positives has a big impact on the F1 score.

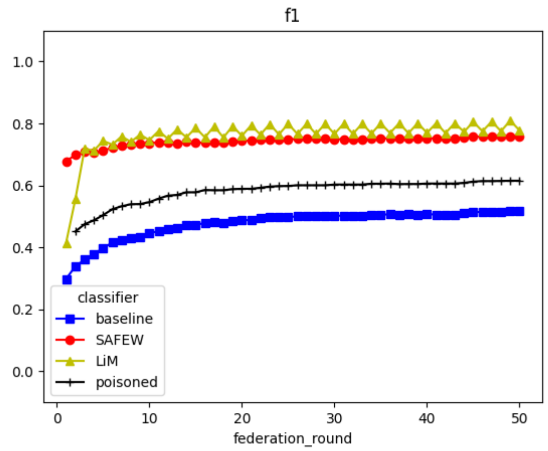

Fig. 4. F1 score of 100 honest + 100 malicious clients, using 200 features with baseline classifier kNN (n=3). LiM outperforms SAFEW (i.e. no FL), which performs better than baseline.

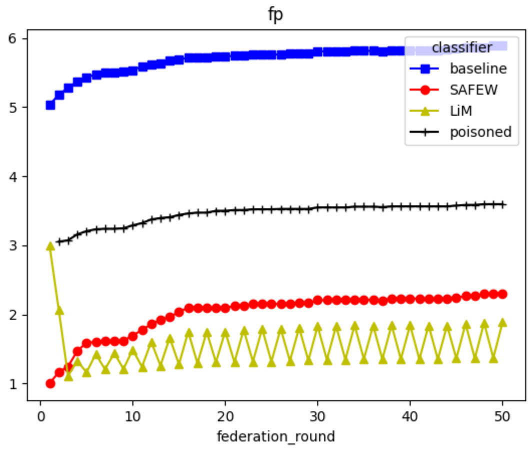

Fig. 5. False positives of 200 clients with 200 features, using kNN (n=3) as baseline classifier.

For the second case, we consider as baseline classifier an RF with 200 decision trees and 200 features. In the results shown in figure 6, we can see that clients are affected by the attack every second round. The F1 score jumps between 0.4 and 0.6 due to slight variations of the weights that make the false positives per client jump from 2 to 2.5, as shown in figure 7.

Table 3. Comparison of client average performance across LiM, SAFEW and different baselines when 50% of the clients are adversarial. Experiments are carried out for 50 rounds of federation using the Top 100, 200 and 500 features which were selected based on the chi2 test.

Baseline	#Features	FP (raw values)				F1 (%)			
		Baseline	SAFEW	LiM	Adv. client	Baseline	SAFEW	LiM	Adv. client
KNN n3	100	10.3	5.7	2.3	2.9	25.3	36.7	59.8	54.0
	200	5.9	2.4	1.7	2.7	39.2	61.0	47.0	27.5
	500	6.9	3.2	2.4	3.9	45.8	66.5	68.9	43.8
LR c1	100	5.6	8.7	8.1	8.9	55.6	50.4	51.8	36.8
	200	3.1	1.6	1.6	1.1	52.7	81.4	81.4	26.7
	500	2.5	2.5	2.5	2.5	61.1	61.1	61.1	42.7
RF n100	100	2.6	4.7	4.1	4.6	61.9	51.2	53.0	37.9
	200	1.1	1.6	1.0	1.6	70.0	75.1	78.9	50.5
	500	2.2	2.2	2.2	2.1	64.1	64.1	64.2	53.8
RF n200	100	2.5	3.0	3.0	3.0	55.0	50.0	50.0	16.3
	200	1.5	2.6	2.1	3.0	66.5	64.5	65.7	39.9
	500	2.2	2.5	2.2	4.6	56.8	54.8	56.8	18.1
RF n50	100	1.8	2.8	2.5	3.3	72.5	64.2	65.3	54.7
	200	2.1	4.2	1.8	2.2	67.6	61.6	73.2	43.9
	500	3.1	2.5	2.5	2.6	58.3	62.3	62.3	36.9
SVM c1	100	5.1	5.1	5.1	4.0	41.5	41.5	41.5	20.5
	200	4.0	3.5	3.5	0.5	49.1	52.7	52.7	55.5
	500	3.7	3.7	3.7	2.6	42.7	42.7	42.7	12.1

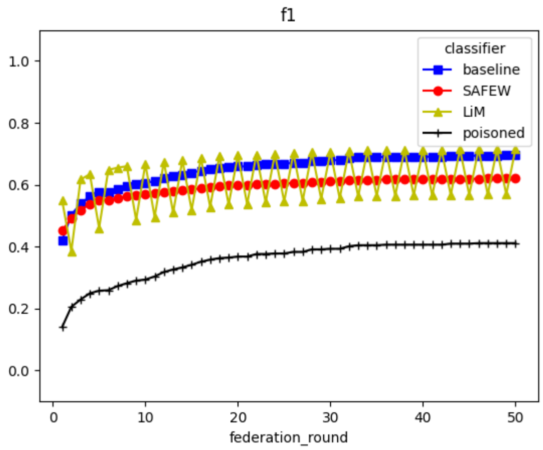

Fig. 6. F1 score of 200 clients with 200 features, using RFn200 as baseline classifier and with poisoning.

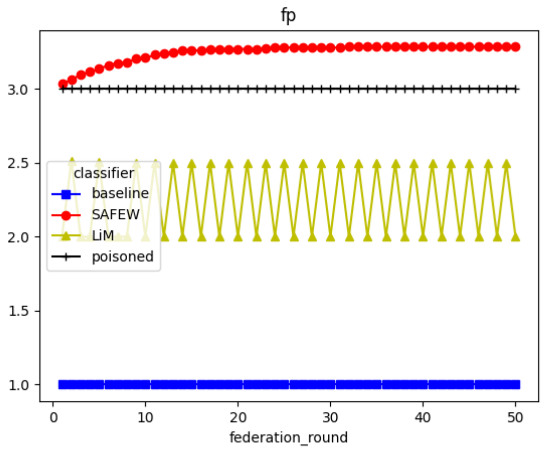

Fig. 7. False positives of 200 clients with 200 features, using RFn200 as baseline classifier and with poisoning.

These results indicate that LiM is resistant against poisoning attacks. This is thanks to the strong influence of the cloud in the federated weights and the use of trusted data (cf. [26] - L2 and L3 from section 7-A) to compute the cloud weights. Moreover, the power of the attacker is limited by the small number of SAFEW weights, which makes it difficult to distribute noise across multiple dimensions (cf. [26] - L1 from section 7-A) This allows LiM to use malicious and honest weights together and still resist poisoning attacks.

6.3.2 Privacy attacks

Federated learning provides a defense mechanism against privacy attacks: hiding the raw features of the installed apps to make it more difficult for the cloud server to infer information. However, the submitted client models may still leak information that can be used to infer, e.g. if a specific app has been part of their training set. Membership inference [18] relies on the fact that updates to the client models may change only a few parameters, and those parameters can reveal information about specific apps used locally in a federation round.

In LiM, updates to client models only change the SAFEW weights of the base learners. It is safe to assume that the number of base learners of SAFEW is significantly smaller than the number of parameters of an individual base learner, e.g. a deep neural network. This data compression greatly reduces the information available to the adversary to perform the membership inference, lowering the probability of success of this attack.

We evaluate if the attack by Melis et al. [18] can infer whether clients installed a target app, using the weights shared by the clients as the gradients. We denote by r the federation round, i is the $i-$th client, and c the cloud. The cloud first computes the weights associated with a specific app, $w_{app}^{r} = SAFEW(baseline\_model, base\_models, app)$, and the weights of the client with no federation for round r, undoing step 5 as described in section 5: $w_{c}^{r,nofederated} = (2 * w_{i}^{r}) - w_{c}^{r-1}$

Clients compute the weights w_c using the data from all the apps they have installed. We assume the cloud does not have access to the list of preinstalled apps of each client, but aims to identify the apps users install in each round. To achieve that goal, the cloud isolates the contribution of a single round by subtracting the weights of the last and the previous rounds, i.e. round r and $r-1$. This is reflected in equation 8. The attack succeeds if equation 8 holds and the identified app was indeed installed by the client in the previous round (i.e. there is a true positive).

$$w_{app}^{r} - (w_{c}^{r,nofederated} - w_{c}^{r-1,nofederated}) = 0 \qquad (8)$$

On the contrary, if there is no app that makes equation 8 hold, then the attack does not succeed. If equation 8 is satisfied by 2 or more apps that leads to false positives. In that case, the best the adversary can do is to guess among the candidate apps, so the more apps that produce the same weights, the more resistant is LiM to privacy attacks.

We implemented this attack and ran experiments with the same SAFEW configurations (sets of baseline and base learners). In order to increase the likelihood of success, we let the cloud have access to the complete dataset of apps: this way, clients cannot install apps that the cloud cannot infer membership of. For each round, the cloud counts the number of apps in its dataset that makes equation 8 hold. We then check if those apps were indeed installed by the corresponding client.

The results show that the cloud is unable to find any app that makes equation 8 hold, i.e. there are 0 true positives and 0 false positives. We observe that the weights resulting from testing a SAFEW model with a single specific app are very different from the difference between the weights sent in the last two rounds. Specifically, we realize that there is not enough information in a single app for SAFEW to associate different weights to each base learner, while in successive federation rounds, clients slightly increase some weights in detriment of others.

6.4 Evaluation using MaMaDroid dataset

In order to further verify the performance of LiM, we also evaluated LiM using the MaMaDroid dataset [23] under the setup described at the beginning of this section. Due to errors in the download and the feature extraction, we used 7K malware and 7K clean apps. We ran 2 experiments per configuration without adversarial clients, and 2 experiments where half of the clients perform the poisoning attack described in section 6.3.1.

First, we will describe the results of the experiments without adversarial clients. Then, we will focus on the results obtained with 50% of clients submitting poisoned weights. For each of the scenarios, we analyze the cloud and the client results.

6.4.1 Without poisoning

We observed no performance difference between LiM, SAFEW, Centralized SAFEW, and baseline in the cloud, as they all obtain an F1 score of around 90%. On the other hand, there was a noticeable difference in the performance of the clients. Similarly to the results presented in previous sections, LiM is able to outperform baseline and SAFEW most of the times, including in configurations that use RF (n=50,200) as baseline.

Overall, the F1 scores were lower with this new dataset due to the higher number of false positives in the baseline learners. This is not surprising, since the dataset itself has fewer distinguishable manifest features and even important dynamic analysis features are present in both clean and malicious apps, resulting in poor distinguishers (cf. section 4.4 from [23]).

6.4.2 With poisoning

More interesting were the results of experiments where 50% of the clients poisoned their weights to trigger a false negative, i.e. so that a specific malicious app is misclassified as clean. Table 4 shows that LiM still outperforms baseline and SAFEW in most of the cases. The highest performance gain with respect to the baseline and SAFEW is using kNN with 100 features as baseline.

An example of such configuration is shown in figure 8. We can see that LiM outperforms both SAFEW and baseline, stabilizing at 0.4 even as the poisoned clients worsen their performance. This shows that LiM, under a targeted integrity attack, can still learn from the federation.

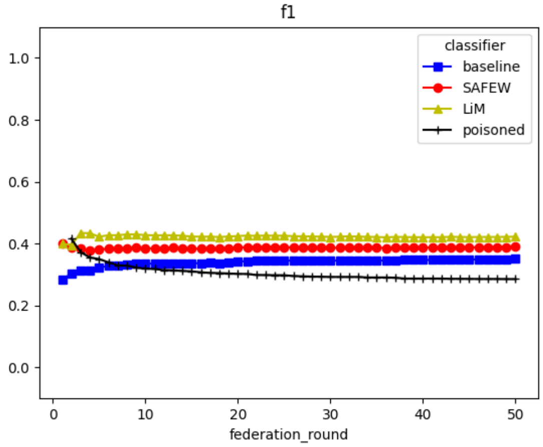

Fig. 8. F1 score of 200 clients and the MaMaDroid dataset with 500 features, using kNN (n=3) as baseline classifier

Finally, we found the membership inference attack unsuccessful, as the cloud is unable to find any single app that makes equation 8 hold in any of these experiments (0 true positives, 0 false positives), further indicating that LiM is resistant to membership inference.

7 Discussion and future work

Our evaluation shows that LiM enables federated clients to learn from each other even when they do not have local access to ground truth. Our experiments show the importance of the learner selection in the individual SAFEWs, which must maximize variance across their predictions as well as their individual performance. We observe that random forests provide a high F1 score by themselves, but LiM can outperform them by fine-tuning the weights of weaker but complementary learners. The best results were achieved when all learners influenced the final predictions, rather than defaulting to the predictions of the strongest learner. In this paper we have used standard learners, but we expect LiM to greatly benefit from a more careful selection of base learners. Future work is needed to verify this hypothesis.

Further measures to protect users from privacy attacks could include differential privacy. Previous work has shown that adding noise to the client parameters can prevent information leakage from client models [29]. LiM takes instead a data aggregation approach, compressing client parameters to a smaller set of bounded hyper-parameters to significantly decrease the amount of information available to the adversary (cf. section 6.3). This approach does not compromise on the quality of the models shared by the users, and it does not introduce limitations regarding the minimum number of users in the federation [29]. Nevertheless, LiM's protection does not hide whether or not users have installed new applications, as users' models remain unchanged while no new applications are installed; differential privacy can stop the leakage of such information.

Regarding performance results, we highlight that 1) LiM does not sacrifice performance for privacy, as it matches and sometimes even outperforms a privacy-invasive Centralized SAFEW, and 2) the number of false positives in clients decreases across federation rounds. We observe a drastic improvement in the first round of federation, when client weights contribute to updating the initial model created by the cloud. This leads us to believe that increasing the number of clients may reproduce this effect along further rounds, as there will be more information coming from additional client weights. Simulating LiM at scale can help clarify the relationship between the number of clients and LiM performance.

Interestingly, LiM can avoid virtually any privacy loss with respect to a centralized SAFEW as the federation of the clients provides enough information to arrive to the same weights of the privacy-invasive model. Even though the differences between the weights of the cloud without federation and the weights of the centralized SAFEW can be relatively high (e.g. to 0.1 in a single weight, 0.37 vs 0.47), clients provide enough information to equalize them.

We envision LiM as a system that can be practically deployed in real-world smartphones. While market interests may dissuade powerful organizations like Google to deploy LiM in stock Android, we believe third-party

Table 4. Comparison of client average performance using the MaMaDroid dataset across LiM, SAFEW and different baselines when 50% of the clients are adversarial. Experiments are carried out for 50 rounds of federation using the Top 100, 200 and 500 features which were selected based on the chi2 test.

Baseline	#Features	FP (raw values)				F1 (%)			
		Baseline	SAFEW	LiM	Adv. client	Baseline	SAFEW	LiM	Adv. client
KNN n3	100	8.5	8.5	5.7	5.4	25.1	23.8	32.5	24.3
	200	7.3	5.9	5.2	5.2	32.5	41.3	39.5	21.7
	500	6.1	5	4.6	4.5	34.4	38.7	40.6	30.7
LR c1	100	7.8	7.5	7.1	7.1	26.4	27.2	28.8	11.1
	200	8.4	8.6	8.6	7.5	20.4	15.2	17.5	0
	500	5.3	5.0	4.1	3.6	17.8	33.0	37.5	39.8
RF n100	100	6.7	8.9	6.8	6.8	26.4	26.9	32.9	15.9
	200	8.3	10.8	9.5	9.3	44.2	38.4	40.2	29.8
	500	5.4	5.4	5.3	4.3	36.7	36.7	36.6	22.0
RF n200	100	7.3	10.1	7.8	6.9	42.0	34.2	39.8	39.7
	200	5.1	5.1	5.1	5	28.5	28.5	28.5	0.9
	500	4.3	4.1	3.2	3.8	38.5	39.1	46.5	25.8
RF n50	100	6.4	9.9	7.0	5.9	29.1	19.8	22.5	0.3
	200	5.2	6.5	5.2	4.9	40.1	35.4	41.0	49.1
	500	5.9	6.5	5.4	5.0	42.6	41.0	44.6	43.8
SVM c1	100	7.2	7.7	7.3	7.2	23.2	24.0	23.3	10.7
	200	10.2	7.6	6.2	5.2	20.2	25.9	28.5	25.6
	500	2	4	4	5.2	60.9	41.8	41.8	35.1

Android distributions differentiating themselves by being more privacy conscious can develop the client app as a privileged service executed upon the installation of (one or more) apps. This practical implementation would perform static analysis over the manifest files of the newly installed apps, without the need to trust the LiM service provider with sensitive client models.

Limitations & Future work: We expect future work to address the following limitations of the current formulation of LiM, namely:

- Malware family-wise classification is out of the scope in this paper. Our figures do not take into account the specific characteristics of the malware apps installed by clients.
- Real world experiments at scale. In this paper, we perform experiments with standard learners, 50 rounds of federation, a static set of users, and predefined parameters to simulate probabilities of installing malware, clean and popular apps. Increasing the number of rounds and performing a thorough selection of the SAFEW learners would provide a more thorough understanding of the performance of LiM in different practical scenarios.

We believe LiM is equipped to act as a self-evolving system that can update itself using the ever-enlarging set of apps users install on their devices. This low maintenance overhead of the service provider, which does not need to label all data to make use of it, together with the geographic distribution of malware, can help LiM detect malicious apps faster [10] and thus prevent malware to disseminate in large numbers. Other applications where self-updating models are of interest could explore LiM as a solution where no burden is placed to the user nor the service provider.

8 Related work

8.1 Machine learning for Android malware classification

ML techniques to detect mobile malware have been extensively investigated, leveraging a few characteristics of the mobile applications (for example, call graphs [20], permissions [14], or both API calls and permissions [12]), and the results obtained were promising. Classification approaches have also been proposed to model and approximate the behaviors of Android applications and discern malicious apps from benign ones. The detection accuracy of a classification method depends on the quality of the features (for example, how specific the features are [21]).

In [19], Milosevic et al. implemented an app that detects malware locally through a pre-trained SVM classifier. They explicitly created a permission based model to detect malware. There is no information to recreate the model, although the model itself is available as part of the OWASP Seraphimdroid project [24]. The dataset they used has 200 benign and 200 malicious apps; however, since 2015, their dataset is no longer publicly available. For more recent state of the art related work in this area, we refer the reader to [23].

8.2 Semi-supervised federated learning

Semi-supervised FL has already been proposed by [2] to take advantage of the abundant unlabeled data in the smart city context. Contrary to LiM, they assume a subset of clients have labeled samples, and use them to train a classifier that will provide the missing labels to retrain another local model. They report an improvement of 8% accuracy compared to the fully supervised FL, but do not consider privacy and integrity attacks nor provide a lower bound for the performance of the clients and the server.

9 Conclusion

We have presented LiM, the first FL-based malware detection framework that works successfully without user access to ground truth by making use of safe semi-supervised learning techniques. We demonstrate its utility as a hybrid system where users keep their apps secret from the service provider while successfully detecting most of the malicious apps they install without raising many false alarms. LiM is resistant against a strategic adversary that compromises 50% of the clients and crafts a malicious app in order to bypass the detection mechanism. Our proposed tool can also withstand membership inference attacks that exploit client updates to try to determine if a specific app was installed by a client. While we have evaluated LiM in the malware detection domain, we note that it can be applied to other problems where users cannot provide ground truth labels to their clients' models, but still benefit from FL – both in terms of performance improvements and privacy properties.

Acknowledgements

This research is partially supported by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI, by the Research Council KU Leuven under the grant C24/18/049, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CaSa - 390781972, by CyberSecurity Research Flanders with reference number VR20192203, and by the United States Air Force and DARPA under Contract No. FA8750-19-C-0502. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force and DARPA.

We also thank Brecht Van der Vliet for his helpful review of the code and the identification of an important implementation issue.

References

[1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, January 2018.

[2] A. Albaseer, B. S. Ciftler, M. Abdallah, and A. Al-Fuqaha. Exploiting Unlabeled Data in Smart Cities using Federated Edge Learning. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1666–1671.

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[4] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings 2014 Network and Distributed System Security Symposium*, San Diego, CA, 2014. Internet Society.

[5] Saba Arshad, Munam A Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. Samadroid: a novel 3-level hybrid malware detection model for android operating system. *IEEE Access*, 6:4321–4339, 2018.

[6] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. *arXiv:1807.00459 [cs]*, July 2018.

[7] Android Developers. https://developer.android.com/guide/topics/permissions/overview – accessed on 28 June 2019.

[8] Steven Diamond and Stephen Boyd. CVXPY: A Python-Embedded Modeling Language for Convex Optimization.

Journal of Machine Learning Research, 17(83):1–5, 2016.

[9] Google. Application fundamentals. https://developer.android.com/guide/components/fundamentals.

[10] S. Hutchinson, B. Zhou, and U. Karabiyik. Are We Really Protected? An Investigation into the Play Protect Service. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4997–5004.

[11] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, and Mehdi Bennis et al. Advances and Open Problems in Federated Learning. *arXiv:1912.04977 [cs, stat]*, December 2019.

[12] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*, 14(3):773–788, 2018.

[13] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[14] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[15] Yu-Feng Li, Lan-Zhe Guo, and Zhi-Hua Zhou. Towards Safe Weakly Supervised Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019.

[16] Yu-Feng Li and Zhi-Hua Zhou. Towards making unlabeled data never hurt. *IEEE transactions on pattern analysis and machine intelligence*, 37(1):175–188, 2014.

[17] Yuping Li, Jiyong Jang, Xin Hu, and Xinming Ou. Android malware clustering through malicious payload mining. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 192–214. Springer, 2017.

[18] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, volume 1, San Francisco, CA, US, May 2019.

[19] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided Android malware classification. *Computers & Electrical Engineering*, 61:266–274, July 2017.

[20] Omid Mirzaei, Guillermo Suarez-Tangil, Jose M de Fuentes, Juan Tapiador, and Gianluca Stringhini. Andrensemble: Leveraging api ensembles to characterize android malware families. *Proceedings of 14th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2019)*, 2019.

[21] Veelasha Moonsamy, Jia Rong, and Shaowu Liu. Mining permission patterns for contrasting clean and malicious android applications. *Future Generation Computer Systems*, 36:122–132, July 2014.

[22] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, volume 1, pages 1021–1035, San Francisco, CA, US, May 2019.

[23] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.

[24] OWASP. OWASP SeraphimDroid Project - OWASP. https://www.owasp.org/index.php/OWASP_SeraphimDroid_Project.

[25] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable and Secure Computing*, 15(1):83–97, January 2018.

[26] Virat Shejwalkar and Amir Houmansadr. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. page 18. Internet Society.

[27] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, pages 587–601, 2017.

[28] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 252–276. Springer, 2017.

[29] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. 15:3454–3469.

[30] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[31] Xin Yao, Tianchi Huang, Chenglei Wu, Ruixiao Zhang, and Lifeng Sun. Towards faster and better federated learning: A feature fusion approach. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 175–179, 2019.

[32] Hanlin Zhang, Yevgeniy Cole, Linqiang Ge, Sixiao Wei, Wei Yu, Chao Lu, Genshe Chen, Dan Shen, Erik Blasch, and Khanh D. Pham. Scanme mobile: A cloud-based android malware analysis service. *SIGAPP Appl. Comput. Rev.*, 16(1):36–49, April 2016.

[33] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

A List of features

A.1 Top 100 features

- android_hardware_camera
- android_hardware_camera_autofocus
- android_hardware_microphone
- android_hardware_screen_landscape
- android_hardware_screen_portrait
- android_hardware_touchscreen_multitouch
- android_hardware_touchscreen_multitouch_distinct
- android_hardware_wifi
- android_permission_access_assisted_gps
- android_permission_access_coarse_location
- android_permission_access_coarse_updates
- android_permission_access_fine_location
- android_permission_access_gps

- android__permission__access__location
- android__permission__access__location__extra__commands
- android__permission__access__wifi__state
- android__permission__call__phone
- android__permission__change__wifi__state
- android__permission__get__tasks
- android__permission__install__packages
- android__permission__kill__background__processes
- android__permission__mount__unmount__filesystems
- android__permission__process__outgoing__calls
- android__permission__read__call__log
- android__permission__read__contacts
- android__permission__read__logs
- android__permission__read__phone__state
- android__permission__read__profile
- android__permission__read__settings
- android__permission__read__sms
- android__permission__receive__boot__completed
- android__permission__receive__sms
- android__permission__restart__packages
- android__permission__send__sms
- android__permission__system__alert__window
- android__permission__use__credentials
- android__permission__write__apn__settings
- android__permission__write__contacts
- android__permission__write__external__storage
- android__permission__write__settings
- android__permission__write__sms
- android__support__v4__content__fileprovider
- cn__domob__android__ads__domobactivity
- com__adfeiwo__ad__coverscreen__sa
- com__adfeiwo__ad__coverscreen__sr
- com__adfeiwo__ad__coverscreen__wa
- com__adwo__adsdk__adwoadbrowseractivity
- com__airpush__android__deliveryreceiver
- com__airpush__android__messagereceiver
- com__airpush__android__pushads
- com__airpush__android__pushservice
- com__airpush__android__userdetailsreceiver
- com__android__browser__permission__read__history__bookmarks
- com__android__browser__permission__write__history__bookmarks
- com__android__launcher__permission__install__shortcut
- com__android__launcher__permission__uninstall__shortcut
- com__android__vending__billing
- com__bving__img__ag
- com__bving__img__rv
- com__bving__img__se
- com__facebook__ads__interstitialadactivity
- com__facebook__facebookactivity
- com__facebook__loginactivity
- com__google__android__c2dm__permission__receive
- com__google__android__gms__ads__adactivity
- com__google__android__gms__ads__purchase__inapppurchaseactivity
- com__google__android__gms__analytics__analyticsreceiver
- com__google__android__gms__analytics__analyticsservice
- com__google__android__gms__analytics__campaigntrackingreceiver
- com__google__android__gms__analytics__campaigntrackingservice
- com__google__android__gms__appinvite__previewactivity
- com__google__android__gms__auth__api__signin__internal__signinhubactivity
- com__google__android__gms__auth__api__signin__revocationboundservice
- com__google__android__gms__common__api__googleapiactivity
- com__google__android__gms__gcm__gcmreceiver
- com__google__android__gms__measurement__appmeasurementcontentprovider
- com__google__android__gms__measurement__appmeasurementinstallreferrerreceiver
- com__google__android__gms__measurement__appmeasurementreceiver
- com__google__android__gms__measurement__appmeasurementservice
- com__google__android__providers__gsf__permission__read__gservices
- com__google__firebase__iid__firebaseinstanceidinternalreceiver
- com__google__firebase__iid__firebaseinstanceidreceiver
- com__google__firebase__iid__firebaseinstanceidservice
- com__google__firebase__messaging__firebasemessagingservice
- com__google__firebase__provider__firebaseinitprovider
- com__google__update__dialog
- com__google__update__receiver
- com__google__update__updateservice
- com__kuguo__ad__boutiqueactivity
- com__kuguo__ad__mainactivity
- com__kuguo__ad__mainreceiver
- com__kuguo__ad__mainservice
- com__mobclix__android__sdk__mobclixbrowseractivity
- com__soft__android__appinstaller___finishactivity
- com__soft__android__appinstaller___firstactivity
- com__soft__android__appinstaller___rulesactivity
- com__soft__android__appinstaller__memberactivity
- com__soft__android__appinstaller__questionactivity
- com__startapp__android__publish__appwallactivity
- net__youmi__android__adactivity

A.2 Top 200 features

- android__hardware__camera
- android__hardware__camera__autofocus
- android__hardware__camera__front
- android__hardware__microphone
- android__hardware__screen__landscape
- android__hardware__screen__portrait
- android__hardware__touchscreen__multitouch
- android__hardware__touchscreen__multitouch__distinct
- android__hardware__wifi
- android__permission__access__assisted__gps
- android__permission__access__coarse__location
- android__permission__access__coarse__updates
- android__permission__access__fine__location
- android__permission__access__gps
- android__permission__access__location
- android__permission__access__location__extra__commands
- android__permission__access__wifi__state
- android__permission__call__phone
- android__permission__camera
- android__permission__change__configuration
- android__permission__change__network__state
- android__permission__change__wifi__state
- android__permission__clear__app__cache
- android__permission__get__tasks
- android__permission__install__packages
- android__permission__kill__background__processes
- android__permission__mount__unmount__filesystems
- android__permission__process__outgoing__calls
- android__permission__read__calendar
- android__permission__read__call__log
- android__permission__read__contacts
- android__permission__read__external__storage
- android__permission__read__logs
- android__permission__read__phone__state
- android__permission__read__profile
- android__permission__read__settings
- android__permission__read__sms
- android__permission__receive__boot__completed
- android__permission__receive__sms
- android__permission__receive__wap__push
- android__permission__record__audio
- android__permission__restart__packages
- android__permission__send__sms
- android__permission__system__alert__window
- android__permission__use__credentials
- android__permission__vibrate
- android__permission__write__apn__settings
- android__permission__write__calendar
- android__permission__write__contacts
- android__permission__write__external__storage
- android__permission__write__settings
- android__permission__write__sms
- android__support__v4__content__fileprovider
- biz__neoline__android__reader__bookmarksandtocactivity
- biz__neoline__android__reader__libraryactivity
- biz__neoline__android__reader__neobookreader
- biz__neoline__android__reader__textsearchactivity
- biz__neoline__app__core__core__application__shutdownreceiver
- biz__neoline__app__core__ui__android__dialogs__dialogactivity
- biz__neoline__app__core__ui__android__library__crashreportingactivity
- biz__neoline__test__donationactivity
- cn__domob__android__ads__domobactivity
- com__adfeiwo__ad__coverscreen__sa
- com__adfeiwo__ad__coverscreen__sr
- com__adfeiwo__ad__coverscreen__wa
- com__adwo__adsdk__adwoadbrowseractivity
- com__adwo__adsdk__adwosplashadactivity
- com__airpush__android__deliveryreceiver
- com__airpush__android__messagereceiver
- com__airpush__android__pushads
- com__airpush__android__pushservice
- com__airpush__android__smartwallactivity
- com__airpush__android__userdetailsreceiver
- com__amazon__device__messaging__permission__receive
- com__anddoes__launcher__permission__update__count
- com__android__browser__permission__read__history__bookmarks
- com__android__browser__permission__write__history__bookmarks
- com__android__launcher__permission__install__shortcut
- com__android__launcher__permission__uninstall__shortcut
- com__android__vending__billing
- com__biznessapps__layout__maincontroller
- com__biznessapps__player__playerservice
- com__biznessapps__pushnotifications__c2dmmessagesreceiver
- com__biznessapps__pushnotifications__c2dmregistrationreceiver
- com__bving__img__ag
- com__bving__img__rv
- com__bving__img__se
- com__chartboost__sdk__cbimpressionactivity
- com__elm__lma
- com__elm__lmr
- com__elm__lms
- com__elm__lmsk
- com__facebook__ads__audiencenetworkactivity
- com__facebook__ads__interstitialadactivity
- com__facebook__customtabactivity
- com__facebook__customtabmainactivity
- com__facebook__facebookactivity
- com__facebook__facebookcontentprovider
- com__facebook__loginactivity
- com__feiwothree__coverscreen__sa
- com__feiwothree__coverscreen__sr
- com__feiwothree__coverscreen__wa
- com__google__android__apps__analytics__analyticsreceiver
- com__google__android__c2dm__permission__receive
- com__google__android__gcm__gcmbroadcastreceiver
- com__google__android__gms__ads__adactivity
- com__google__android__gms__ads__purchase__inapppurchaseactivity
- com__google__android__gms__analytics__analyticsreceiver
- com__google__android__gms__analytics__analyticsservice
- com__google__android__gms__analytics__campaigntrackingreceiver
- com__google__android__gms__analytics__campaigntrackingservice
- com__google__android__gms__appinvite__previewactivity
- com__google__android__gms__auth__api__signin__internal__signinhubactivity
- com__google__android__gms__auth__api__signin__revocationboundservice
- com__google__android__gms__common__api__googleapiactivity
- com__google__android__gms__gcm__gcmreceiver
- com__google__android__gms__measurement__appmeasurementcontentprovider
- com__google__android__gms__measurement__appmeasurementinstallreferrerreceiver
- com__google__android__gms__measurement__appmeasurementjobservice
- com__google__android__gms__measurement__appmeasurementreceiver
- com__google__android__gms__measurement__appmeasurementservice
- com__google__android__providers__gsf__permission__read__gservices
- com__google__firebase__iid__firebaseinstanceidinternalreceiver

- com_google_firebase_iid_firebaseinstanceidreceiver
- com_google_firebase_iid_firebaseinstanceidservice
- com_google_firebase_messaging_firebasemessagingservice
- com_google_firebase_provider_firebaseinitprovider
- com_google_update_dialog
- com_google_update_receiver
- com_google_update_updateservice
- com_htc_launcher_permission_update_shortcut
- com_klpcjg_wyxjvs102320_browseractivity
- com_klpcjg_wyxjvs102320_mainactivity
- com_klpcjg_wyxjvs102320_vdactivity
- com_kuguo_ad_boutiqueactivity
- com_kuguo_ad_mainactivity
- com_kuguo_ad_mainreceiver
- com_kuguo_ad_mainservice
- com_majeur_launcher_permission_update_badge
- com_mobclix_android_sdk_mobclixbrowseractivity
- com_nd_dianjin_activity_offerappactivity
- com_onesignal_gcmbroadcastreceiver
- com_onesignal_gcmintentservice
- com_onesignal_notificationopenedreceiver
- com_onesignal_permissionsactivity
- com_onesignal_syncservice
- com_parse_gcmbroadcastreceiver
- com_parse_parsebroadcastreceiver
- com_parse_pushservice
- com_paypal_android_sdk_payments_futurepaymentconsentactivity
- com_paypal_android_sdk_payments_futurepaymentinfoactivity
- com_paypal_android_sdk_payments_loginactivity
- com_paypal_android_sdk_payments_paymentactivity
- com_paypal_android_sdk_payments_paymentconfirmactivity
- com_paypal_android_sdk_payments_paymentmethodactivity
- com_paypal_android_sdk_payments_paypalfuturepaymentactivity
- com_paypal_android_sdk_payments_paypalservice
- com_sec_android_provider_badge_permission_read
- com_sec_android_provider_badge_permission_write
- com_soft_android_appinstaller__finishactivity
- com_soft_android_appinstaller__firstactivity
- com_soft_android_appinstaller__rulesactivity
- com_soft_android_appinstaller__services_smssenderservice
- com_soft_android_appinstaller__sms_binarysmsreceiver
- com_soft_android_appinstaller_memberactivity
- com_soft_android_appinstaller_questionactivity
- com_software_application__c2dmreceiver
- com_software_application__checker
- com_software_application__main
- com_software_application__notificator
- com_software_application__offertactivity
- com_software_application__showlink
- com_software_application__smsreceiver
- com_software_application_permission_c2d_message
- com_sonyericsson_home_permission_broadcast_badge
- com_sonymobile_home_permission_provider_insert_badge
- com_startapp_android_publish_appwallactivity
- com_startapp_android_publish_fullscreenactivity
- com_startapp_android_publish_overlayactivity
- com_tencent_mobwin_mobinwinbrowseractivity
- com_umeng_common_net_downloadingservice
- com_uniplugin_sender_areceiver
- com_unity3d_ads_android_view_unityadsfullscreenactivity
- com_unity3d_player_unityplayeractivity
- com_unity3d_player_unityplayernativeactivity
- com_urbanairship_corereceiver
- com_urbanairship_push_pushservice
- com_vpon_adon_android_webinapp
- com_waps_offerswebview
- io_card_payment_cardioactivity
- io_card_payment_dataentryactivity
- net_youmi_android_adactivity
- net_youmi_android_adbrowser
- net_youmi_android_adreceiver
- net_youmi_android_adservice
- net_youmi_android_appoffers_youmioffersactivity
- net_youmi_android_youmireceiver
- tk_jianmo_study__bootbroadcastreceiver
- tk_jianmo_study__killpoccessserve
- tk_jianmo_study__mainactivity

B Additional results

Table 5 presents an overview for the performance of the cloud participating in the federation of 200 clients. In this table, we also report results of a `Centralized SAFEW` in order to portray what happens when clients submit the (raw) feature vectors of their apps directly to the cloud, i.e. when user privacy is not guaranteed. Table 6 presents the same overview under the presence of an integrity attack, where 50% of the 200 clients submit poisoned weights.

Table 5. Comparison of cloud average performance across LiM, SAFEW, and different baselines. Experiments are carried out for 50 rounds of federation using the Top 100, 200 and 500 features which were selected based on the chi2 test.

Baseline	#Features	FP (raw values)				F1 (%)			
		Baseline	SAFEW	Centralized SAFEW	LiM	Baseline	SAFEW	Centralized SAFEW	LiM
KNN n3	100	1011.0	1176.0	1182.8	718.6	94.7	95.0	95.0	95.7
	200	1354.0	1021.0	1024.0	627.8	93.9	95.5	95.5	96.1
	500	854.5	978.0	981.1	624.4	95.3	95.8	95.8	96.3
LR c1	100	1035.0	1441.0	1449.2	790.7	94.2	94.5	94.5	95.5
	200	803.5	1321.0	1328.2	746.0	94.5	95.0	94.9	94.7
	500	736.5	1196.0	1201.3	654.3	95.1	95.3	95.3	95.4
RF n100	100	638.5	765.5	775.2	667.7	95.9	95.7	95.7	95.8
	200	596.5	839.0	843.4	691.1	96.1	95.8	95.8	96.0
	500	596.0	875.5	878.9	620.3	96.3	96.0	96.0	96.1
RF n200	100	662.0	933.0	938.8	689.4	95.8	95.5	95.5	95.8
	200	602.0	728.5	734.9	607.0	96.2	96.0	96.0	96.2
	500	590.5	973.5	978.3	607.8	96.3	95.8	95.8	96.3
RF n50	100	663.5	942.5	946.9	671.4	95.8	95.5	95.4	95.8
	200	607.5	880.5	884.3	672.6	96.1	95.7	95.7	96.1
	500	531.0	815.0	817.8	560.2	96.3	96.0	96.0	96.3
SVM c1	100	945.5	1169.0	1177.5	1039.6	94.2	95.0	95.0	94.2
	200	767.5	1029.0	1036.2	832.4	94.6	95.5	95.5	94.5
	500	638.0	938.0	944.7	728.7	95.5	95.9	95.9	95.1

Table 6. Comparison of cloud average performance across LiM, SAFEW, Centralized SAFEW and different baselines when 50% of the clients are adversarial. Experiments are carried out for 50 rounds of federation using the Top 100, 200 and 500 features which were selected based on the chi2 test.

Baseline	#Features	FP (raw values)				F1 (%)			
		Baseline	SAFEW	Centralized SAFEW	LiM	Baseline	SAFEW	Centralized SAFEW	LiM
KNN n3	100	1006.0	1159.0	1163.3	682.2	94.7	95.1	95.1	95.9
	200	999.0	1093.5	1074.5	687.8	95.0	95.4	95.4	95.8
	500	887.5	1008.0	1010.8	684.2	95.3	95.8	95.8	95.9
LR c1	100	1027.5	1431.0	1443.8	1093.7	94.3	94.5	94.5	94.9
	200	813.0	1676.0	1683.5	653.5	94.4	94.0	94.0	95.7
	500	748.0	1240.5	1242.5	603.9	94.9	95.2	95.2	95.8
RF n100	100	676.0	940.5	947.1	852.5	95.8	95.4	95.4	95.0
	200	583.0	701.5	703.5	594.3	96.1	96.0	96.0	96.2
	500	577.0	753.0	756.3	532.7	96.4	96.2	96.2	96.4
RF n200	100	671.5	793.5	795.5	681.5	95.8	95.7	95.7	95.8
	200	614.5	697.0	698.8	637.7	96.1	96.0	96.0	96.1
	500	623.0	854.5	857.9	635.0	96.4	95.9	95.9	96.2
RF n50	100	663.5	765.0	771.7	675.8	95.8	95.8	95.7	95.8
	200	637.5	916.5	922.6	648.5	96.0	95.6	95.6	96.1
	500	520.0	794.0	795.5	595.8	96.4	96.1	96.1	96.3
SVM c1	100	1016.5	1162.5	1165.9	1062.5	94.2	95.0	95.0	94.2
	200	750.0	1041.0	1043.9	708.0	94.6	95.4	95.4	95.6
	500	639.0	921.0	924.4	664.9	95.5	95.9	95.9	95.7