FedCRI: Federated Mobile Cyber-Risk Intelligence

Hossein Fereidooni*, Alexandra Dmitrienko†, Phillip Rieger*
Markus Miettinen*, Ahmad-Reza Sadeghi*, and Felix Madlener‡
*Technical University of Darmstadt, Germany
†University of Würzburg, Germany
‡KOBIL, Germany

***Abstract*—In the present era of ubiquitous digitization more and more services are becoming available online which is amplified by the Corona pandemic. The fast-growing mobile service market opens up new attack surfaces to the mobile service ecosystem. Hence, mobile service providers are faced with various challenges to protect their services and in particular the associated mobile apps. Defenses for apps are, however, often limited to (lightweight) application-level protection such as app hardening and monitoring and intrusion detection. Therefore, effective risk management is crucial to limit the exposure of mobile services to threats and potential damages caused by attacks.**

In this paper, we present FedCRI, a solution for sharing Cyber-Risk Intelligence (CRI). At its core, FedCRI transforms mobile cyber-risks into machine learning (ML) models and leverages ML-based risk management to evaluate security risks on mobile devices. FedCRI enables fast and autonomous sharing of actionable ML-based CRI knowledge by utilizing Federated Learning (FL). FL allows collaborative training of effective risk detection models based on information contributed by different mobile service providers while preserving the privacy of the training data of the individual organizations. We extensively evaluate our approach on several real-world user databases representing 23.8 million users of security-critical mobile apps (since Android 4 and iOS 6) provided by nine different service providers in different European countries. The datasets were collected over the course of six years in the domains of financial services, payments, and insurances. Our approach can successfully extract accurate CRI models, allowing effective identification of cybersecurity risks on mobile devices. Our evaluation shows that the federated risk detection model can achieve better than 99% accuracy in terms of F1-score in most risk classification tasks with a very low number of false positives.

Keywords – *Cyber-Risk Intelligence (CRI), Federated Learning (FL), Cyber-Threat Intelligence (CTI), Mobile Platform*

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.23153
www.ndss-symposium.org

I. Introduction

Today, we are witnessing more and more mobile services and apps launched in different application domains. This trend is amplified by the Corona pandemic that forced many organisations and even traditional businesses to move to online space. Oftentimes these mobile services perform security-critical operations, such as collecting user identifiable information or conduct payments. Consequently, mobile devices are exposed to a variety of attacks, for instance, through users installing potentially malicious apps, or jailbreaking the device to circumvent limitations imposed by the operating system, thereby in effect disabling part of the OS security controls. Hence, mobile service providers are not only concerned with the security of their online services but also need to consider the challenging operating environments of their apps, which need to potentially co-exist with malware on the same device.

For protection, service providers often deploy application-level defense strategies, such as app hardening and intrusion detection in combination with *risk management*. Note that OS vendors typically do not provide the possibility to integrate kernel-level protection mechanisms (e.g., dynamic analysis) in apps. Moreover, hardware-based security mechanisms are either not available on all devices, or when available, they are mainly used by platform vendors for their own purposes and not accessible to third parties. Hence, application-level protection (e.g., features relevant to static analysis) often remains the only available option to service providers. While such protection is limited and cannot, to a full extent, resist advanced attack vectors on mobile platforms (such as kernel-level exploits), the mobile service providers developed risk management techniques that help to limit exposure of their services on risky platforms. For instance, online banking apps may monitor for risk factors, such as detecting jailbreaks, and allow users to view their banking data, but disable other types of transactions unless other out-of-band authorization mechanisms are used.

While risk monitoring and management are common measures used by established service providers, it is often out of reach for small and medium-size organisations due to the lack of expertise, significant costs, and time-to-market

constraints. One promising solution to this problem, which we aim to explore in this paper, is to enable sharing of risk information among providers of various sizes and levels of security expertise. The idea of sharing threat intelligence information has been already explored in the context of Cyber-Threat Intelligence (CTI) systems, which were shown to be helpful in understanding the security and privacy risks, limiting the exposure to threats, and confining the potential damages caused by attacks [7]. CTI sharing systems, however, exchange knowledge about *observed security and privacy breaches* using Indicators of Compromise (IoC) in form of IP addresses, email logs, alerts, incident response reports, event logs, DNS logs, and firewall logs, etc. In contrast, we aim to share knowledge about *risks that increase the probability of the attacks*.

CTI sharing systems raise several challenges and concerns: Firstly, they are difficult to manage due to the vast amount of threat data and are too complex to provide actionable intelligence [45]. Secondly, achieving interoperability and automation [40] is challenging. Thirdly, privacy and liability are other concerns for participating entities [12]. Some organizations may be hesitant to share data due to possible reputation damage from disclosed attack incidents, while others might fear to accidentally disclose data that must be kept private due to liability obligations.

Our goal. Our aim is to design and develop a *Cyber-Risk Intelligence (CRI)* sharing system for mobile platforms that (i) concentrates on sharing risks, and (ii) tackles the above-discussed challenges of the CTI systems. In particular, CRI concentrates on sharing probabilistic indicators of potential security risks that may or may not result in actual attacks, in contrast to CTI, which focuses on sharing knowledge about actually observed incidents. Sharing such knowledge can improve the ability of the involved entities to encounter security threats by mitigating the risk of exploitation by taking additional measures on risky platforms. This has the advantage that threats can be mitigated dynamically and even ahead of time by alleviating potential damages, e.g., by adjusting the maximum permissible value of mobile banking transactions, or, by requiring additional out-of-band authentication or authorization before executing sensitive transactions like changing access credentials.

To solve the challenges of existing CTI sharing systems, our core idea is to share the knowledge about security risks *in form of machine learning models*. In particular, we apply the concept of Federated Learning (FL) to enable participants of the CRI system to collaboratively build global risk detection model based on risks observed locally by participating entities. This approach reduces the amount of information to share and its complexity and facilitates actionable intelligence since global model can directly be applied for risk detection without additional complex processing. A reduced amount of information for sharing is also likely to simplify interoperability and automation. CRI also addresses the dilemma of security benefits and associated privacy risks along with legal liability concerns. Participating entities in the system can train their risk detection models locally based on their own private training data and aggregate these models to a global model without the need of disclosing local datasets of individual clients to others.

Contributions. In particular, we make the following contributions:

- **CRI Sharing**. We present FedCRI, a *CRI sharing framework* that leverages risk detection models trained by individual service providers and uses FL to enable collaborative learning based on models contributed by clients participating in the system (cf. Section III). The core novelty of our approach is *sharing information about risks in form of Machine Learning models*, instead of utilizing Indicators of Compromise (IoC) directly, as done by Cyber-Threat Intelligence (CTI) sharing systems. This difference allows us to tackle challenges of CTI systems, such as high complexity, an overwhelming amount of information for processing, and liability and privacy concerns, thus facilitating actionable intelligence.

- **Risk Modeling**. We develop *risk models* for mobile platforms (Section IV). To do so, we utilize Deep Neural Networks (DNNs) that process risk indicators collected by real-world service providers and enable efficient and meaningful risk assessment based on complex risk combinations. Models are intended for individual mobile service providers and allow them to mitigate risks on client platforms, e.g., by limiting maximum transaction amounts or adjusting requirements for additional authentication. Furthermore, such risk models can be automatically processed by FedCRI for sharing risks with other entities. This approach enables service providers to unfold full potential of their risk management systems, which were limited so far due to complexity of risk information and significant manual effort required for risk analysis.

- **Risk Dataset**. Together with our industry partner, we collect a real-world large-scale risk dataset on which we evaluate our approach. The dataset represents in total *23.8 million users* of security-critical mobile apps (since Android 4 and iOS 6) provided by nine different service providers (Section V). The datasets were collected over the course of six years in the sectors of financial services, payments, and insurance in multiple countries on the European continent. Our approach can successfully extract accurate risk models, allowing effective identification of cybersecurity risks on mobile devices. Our evaluation shows that the federated risk detection model can achieve better than 99% accuracy in terms of F1-score in most risk classification tasks with a remarkably small number of false positives. To the best of our knowledge, this is the first large-scale evaluation of risk detection models utilizing Federated Learning.

Overall, our approach allows for effective detection and modeling of cybersecurity risks on mobile platforms by individual service providers and sharing risk information among providers through collaborative learning. It enables the utilization of security-as-a-service business models, where more mature and experienced service providers can share their security expertise with newcomers, thus helping them to fill the gap of security experts' shortage and reducing their time to market.

II. Requirements and Challenges

In this section, we discuss the requirements posed on FedCRI and elaborate on the challenges to be tackled.

Requirements. The requirements are motivated by the goal to achieve real-world deployment for FedCRI system. In particular, we identify two essential requirements that concern heterogeneity of mobile platforms and the level of the software stack at which the protection mechanism should reside on mobile platforms.

- *Device heterogeneity:* We need to support heterogeneous mobile platforms in terms of hardware, OSes (e.g., Android or iOS), and various OS versions. This is an important requirement to be fulfilled since mobile service providers in real-word deal with the high diversity of mobile devices and not fulfilling this requirement will likely impede deployability.
- *App-level protection:* The protection mechanism cannot modify the operating system. This requirement is essential for real-world deployment because service providers do not have the possibility to integrate kernel-level protection mechanisms in apps. Hence, the protection module needs to be residing at the application-level. Such protections are often based on risk indicators that do not prove whether an attack has actually taken place, but instead merely indicate a suspicious system state, e.g., the existence of a process with excessive privileges, or the presence of a system library that normally does not exist on a certain device type.

Challenges. We present the challenges posed on the development of an efficient, and private risk sharing framework.

- Firstly, CTI systems rely on the processing of a vast amount of data and are too complex to provide actionable intelligence [45]. Consequently, they are not well suitable for small organizations, since best practice guidelines for such systems require dedicated security teams to manage threat intelligence activities and allocation of an adequate budget [45]. To overcome this challenge, we apply the concept of FL to enable participants of the CRI system to collaboratively build global risk detection model based on risks observed locally by participating entities. This approach reduces the amount of information to share and its complexity and facilitates actionable intelligence since global models can directly be applied for risk detection without additional complex processing.
- Secondly, achieving interoperability and automation [40] is challenging. As pointed out in [55], various standards (e.g., STIX [5], TAXII [6], Open IOC [2], CybOX [1]) used by threat sharing platforms hindered CTI providers and receivers to communicate flawlessly to each other due to lack of support of these standards by applications. Utilizing FL reduces the amount of information for sharing which is likely to simplify interoperability and automation.
- Thirdly, legal liability is another issue for organizations participating in such sharing schemes [12]. When dealing with CTI, privacy and legal aspects regarding governing and sharing data need to be taken into account very carefully. Organizations may be hesitant to share data to avoid reputation damage caused by disclosing attack incidents [45]. Federated-learning-based CRI addresses the dilemma of security benefits and associated privacy risks and legal liability concerns. Participating entities in the system can train their risk detection models locally based on their own private training data and aggregate these models to a global model without the need of disclosing local datasets of individual clients to others

III. FedCRI Design

In this section, we present the design of FedCRI – the Federated Cyber-Risk Intelligence system for mobile platforms. It builds upon our concept of Cyber-Risk Intelligence (CRI) that, similar to CTI, can improve the security of computing systems through collaboration and sharing knowledge. In contrast to CTI, CRI focuses on sharing knowledge about risk factors that indicate that the platform is more likely to be compromised, rather than information about actual attacks. To address limitations of existing CTI sharing systems, FedCRI relies on the concept of federated learning and shares risk knowledge as machine learning models instead of sharing raw data.

In the following, we first provide the high-level overview of FedCRI platform and then describe system components in more detail.

A. High-level Overview

The high-level overview of FedCRI system is depicted in Figure 1. The system model involves the following entities: a CRI Provider, Service Providers, and End Users. Service Providers provide various online services to their customers, End Users, via mobile apps. They can range from online shopping and banking applications offered by established service providers with significant security expertise to services provided by new businesses, such as restaurants and barber shops, or public entities, such as universities and schools, that, for instance, utilize apps to manage attendance lists that became mandatory in many locations due to COVID pandemic. End Users use mobile apps to access these services and may perform security-sensitive operations such as login or performing financial transactions, or entering their personally identifiable information into attendance lists.

The risk management system is run on both Service Providers and mobile platforms of End Users and monitors the runtime environment of mobile devices concerning specific risk indicators that correlate with certain threats. It communicates the risk indicators over a dedicated secure channel to the risk management backend server. The risk indicators are grouped into separate categories, so-called Risk-IDs. Examples of risks used in this work are *jailbreak*, *code injection*, *malicious application*, or *simulator usage*[1]. Users may jailbreak their phones to activate extra functionalities offered by the OS, or they may intentionally or unintentionally install malicious

[1]Note that definition of risk indicators is not a contribution of this work. We use risk indicators utilized by risk monitoring systems deployed in the real world by established service providers.

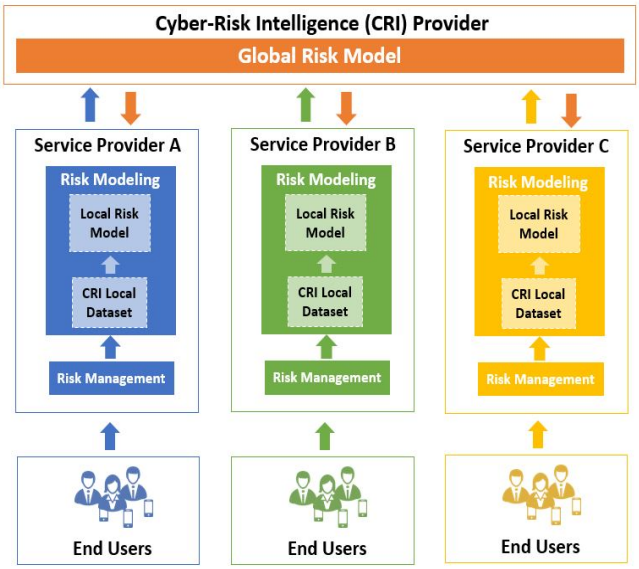

Fig. 1: High-level overview of FedCRI platform

apps, or use a simulator to modify underlying components of the OS to circumvent OS-provided protection measures. An adversary may inject malicious code to bypass a substantial portion of the app's protection features (cf. Appendix IV-A for more details).

Service Providers conduct risk detection and build local risk models (cf. Section IV) using local CRI datasets collected from their End Users and upload them to the CRI Provider. In turn, the CRI Provider is responsible for aggregating the local risk detection models of Service Providers into a global risk model and distributing the global model back to the Service Providers. Once a trained global model is available, the Service Provider can use it for risk detection.

Note that FedCRI platform can also be used by small size Service Providers who do not have sufficient users for building their own local risk models. Such entities can receive ready-to-use global model and directly apply it for risk detection. Global model can be provided by the FedCRI system on the basis of subscription to the risk detection service.

B. Trust Model and Assumptions

In the following, we describe the assumptions we make with regards to the attacker capabilities and trusted system components:

- *Mobile Platform:* We assume that End Users may install arbitrary malicious applications, however, the system-level software is trusted. In practice, mobile service providers also consider best-effort detection of system-level risks, such as injection of malicious system libraries. These measures are effective, for example, against untargeted attacks that are not aware of a risk monitoring system installed on a victim's platform.
- *Communication:* We assume that network communication is secured and cannot be affected by an attacker. This assumption can be easily fulfilled through the use of standard secure communication protocols such as SSL/TLS.
- *Data collection consent.* Since there is a software component integrated into the protected app that monitors and collects potential risks, Service Providers are required to obtain End Users' consent for collecting risk information (e.g., during the app installation).
- *Adversarial ML and poisoning.* We assume that all Service Providers are registered in the CRI sharing platform and are trusted by the CRI Provider. Thus, adversarial machine learning attacks such as membership inference [51], generating adversarial examples [39], [48], and backdoor attacks [20] are out of scope for this work, despite the fact that different defense strategies against the above-mentioned attacks have been proposed in the literature [16], [35]. We note that defenses against backdoor attacks can be integrated into our system [11], [43].

C. FedCRI Platform Operation

In the following, we shed light on operation of the FedCRI platform. In particular, overall operational cycle can be divided into three use cases: (i) data collection, (ii) federated model training, and (iii) risk detection. We describe these use cases using Figure 2, which depicts the FedCRI platform in more detail.

Data Collection. Initially, FedCRI platform operates in monitoring mode and passively collects risk indicator traces from End Users. Upon session establishment with End Users (step 1a), the protected app provides risk indicators that it currently has observed to the Risk Management component of Service Provider. The risk indicators are then stored to a local CRI dataset (step 1b). The monitoring phase continues as long as the CRI dataset contains sufficient data for training a risk detection model.

Federated Model Training. The federated training process is locally initiated by Service Provider by downloading an initial random model from the CRI Provider in Step A. Otherwise, it has already been trained through several rounds of the following process. In Step B, the initial model is trained locally by Service Provider using the local CRI dataset as training data (for more details about local model training, we refer the reader to Section IV). Then, in Step C, local updates to the model are uploaded to the CRI Provider, which aggregates them in Step D to obtain an updated global model. Finally, the updated global model is again downloaded by Service Provider for the next training iteration (step A) and/or to be used locally for risk detection (cf. Appendix C for more details on Federated Learning).

Risk Detection. Once a trained global model is available, the Service Provider can use it for risk detection. In particular, upon user operation, for instance, login (step 1a), the risk indicators associated with the login are communicated over a dedicated secure channel to the risk management backend server (step 1b) and evaluated against the global risk detection model (step 2) to obtain a risk value associated with the user login. When the app interacts with its Service Provider, the

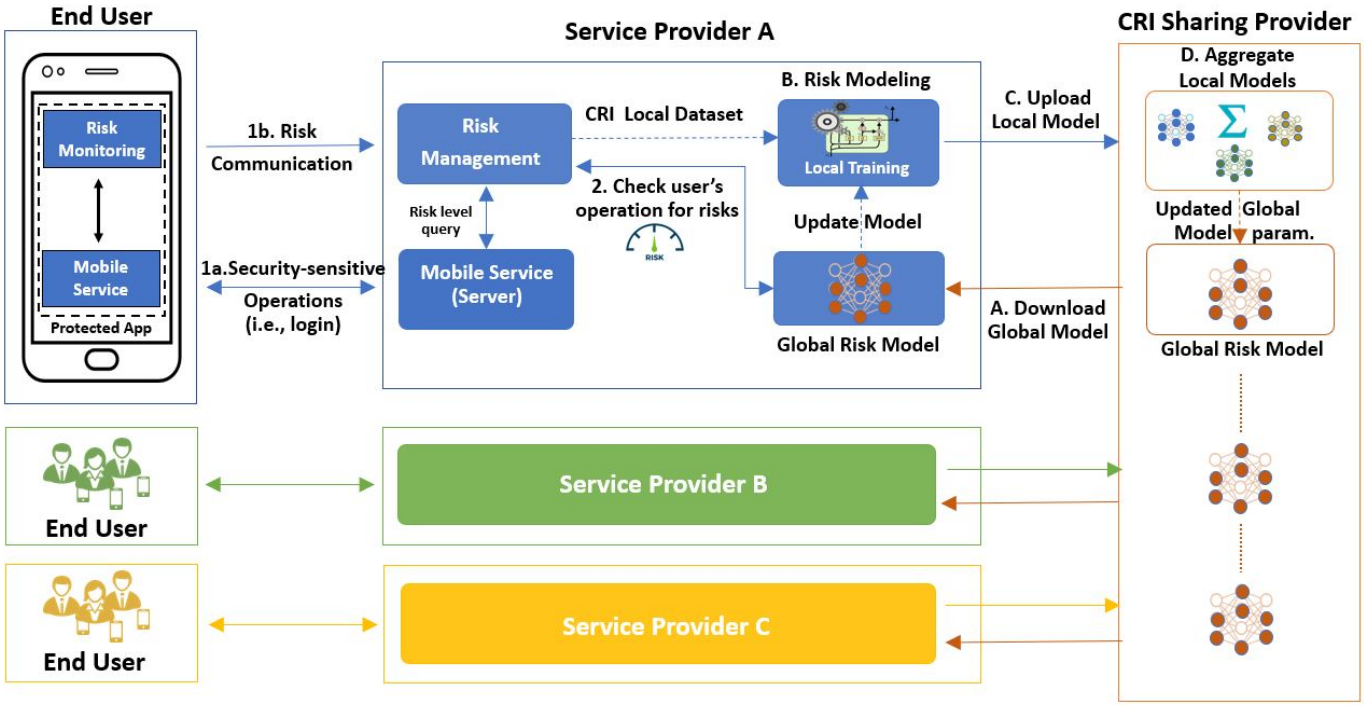

Fig. 2: Risk detection using FedCRI platform. We show execution flows with solid arrows and data flows with dashed ones

Mobile Service (server) will query the risk management for the App's risk level and consequently adapt the services provided to the App accordingly. For example, if the risk level is deemed to be beyond a specific threshold, security-critical functions may be disabled or sensitive information made inaccessible to the app.

IV. Risk Modeling

In this section, we first briefly describe the risk management and monitoring system and risk indicators used for modeling risks. Then, we explain how we incorporate these risk indicators into machine learning models to automatically detect risks and deny suspicious actions (e.g., login attempts or payment transactions). Next, we present the technical details of our risk modeling. At the core of our approach, there is a Machine Learning (ML) pipeline whose task is to model and classify risks on mobile platforms. We build a classifier using Deep Neural Networks (DNNs), a subset of ML algorithms that naturally lend themselves to federation methods developed for models based on Stochastic Gradient Descent (SGD) [34].

A. Risk Management

The risk management system is run on both Service Provider and End Users' mobile phones to monitor risk indicators. As shown in Figure 3, the system consists of several components. The protected App that includes a Mobile Service connecting to its Service Provider over a service-specific communication channel and a Risk Management component that monitors the run-time environment of the smartphone concerning specific risk indicators that correlate with certain threats.

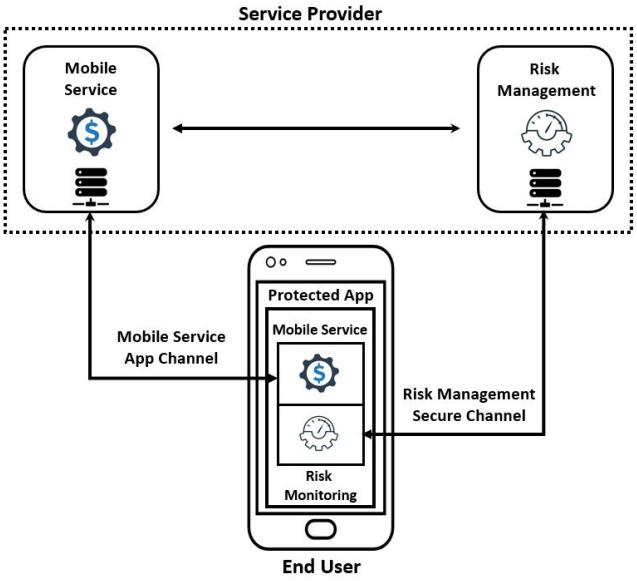

Fig. 3: Risk Management System.

The risk indicators monitored by protected Apps are grouped into separate categories, so-called Risk-IDs. We can easily extend Risk-IDs when new attacks or risk factors are identified. We focus on the following Risk-IDs:

- **JailBreak**: Indicates the presence of Jailbreak software or another indicator of a 'rooted' device. These may indicate a wider attack surface against the protected app and may allow attacks that are not possible without root permissions. However, jailbreaks are also

often utilized by power users merely to enable additional smartphone features. Thus, they cannot as such be regarded as proof of an actual attack, but only as a risk factor.

- **MaliciousApp**: Indicates the presence of an installed app on the smartphone that may pose a potential security risk. Some applications are targeting power-users, who have access and control of filesystems, and require a jailbreak to function properly. However, since they can also be installed in a non-functional state without jailbreaking the phone, they can serve as a weaker additional jailbreak indicator. This risk can also indicate the presence of relevant malware apps similar to the way a virus scanner application would do.
- **CodeInjection**: Indicates that the application code of the app is not consistent with a known benign baseline state. If the code is controlled by an attacker, he/she may easily bypass a substantial portion of the app's protection features. Code modification is rarely used by power users and should thus also not occur otherwise.
- **Simulator**: Indicates that the app runs on a simulator rather than a real device. Running the app on an emulator would allow an attacker to easily modify underlying components of the operating system, and thus, circumvent OS-provided protection mechanisms.

B. Data Collection

We develop our approach based on datasets collected over the course of six years from different Service Providers, most of them with classical End Users, in the sectors of financial services, payments, and insurance. Our datasets contain 23.8 million (more precisely 23.808.000) End Users (since Android 4 and iOS 6) from multiple countries in the European continent. Note that utilizing FL, the data do not leave Service Providers' premises to avoid the risk of disclosing sensitive information of organizations, and the training processes are launched locally by Service Providers. The CRI Provider performs only the aggregation of local models into a global one, which can be redistributed and used for risk detection, as explained in Section III.

The risk data on End Users' device is monitored and collected using a software component integrated into the protected app (cf. Section IV-A). The collected risk information (i.e., Risk-IDs and associated bitmasks) along with metadata, as is explained in Section IV-C, are then transferred to a server and stored in SQL databases. In order to ensure data quality and mitigate the effects of noisy data, we clean the data using Python scripts by identifying and removing errors (e.g., typographical errors and inconsistencies) and duplicate data. The data is also carefully labeled by security experts. We elaborate more on data labeling process in Section IV-D.

C. Feature Selection

We select a set of informative and distinctive features for the risk classification task. In addition to Risk-IDs provided by the risk monitoring component of the protected app, we found that application metadata characterizing the app itself and the properties of the platform have a significant impact on the quality of classification results. The resulting list of relevant features is shown in Table I. Taking a look at the feature list, it is worth mentioning that bitmask is an additional 3-byte array providing additional condensed proprietary information on Risk-ID properties. Its primary function is to help analysts in problem analysis, in case, potential risks are identified.

TABLE I: List of Features

Feature	Type
Platform Type	Metadata
OS Version	Metadata
Protected app Version	Metadata
CPU Architecture	Metadata
MaliciousApp[bitmask]	Risk-ID + associated bitmask
JailBreak[bitmask]	Risk-ID + associated bitmask
CodeInjection[bitmask]	Risk-ID + associated bitmask
Simulator[bitmask]	Risk-ID + associated bitmask

D. Ground Truth Labeling

To establish ground truth for ML model training and testing, we label the identified risks (individual and combinations thereof) into different risk-level categories (Low, Medium, High, and False Positive). We note that false positives are unexpected risks that might occur due to specific changes to the OS (e.g., update). The false positives pose serious challenges for current risk management systems since it requires significant manual processing efforts to identify them. Out of 23.8 million available data entries 3.867.753 entries were associated with risks and false positives. We labeled the data manually using the support of the data experts of the industry partner with many years of expertise in commercial application security. The security team including 4 experts in several face-to-face meetings went over (single/combined) risk entries and discussed which label they should have. As an example, in Android OS, *JBreak* Risk-ID is labelled as "Medium" while the combination of *CodeInjection, MaliciousApp,* and *JBreak* that indicates privacy and/or security breaches is labelled as "High".

E. Data Pre-processing

Before feeding the features to the DNN model, they need to be transformed into a format the model can process. First, we shuffled the data to ensure the model will not be affected by the ordering of data, and then the input data was converted into sequences of tokens and the tokens were mapped to numerical indices ready to feed to Word Embedding layer [37] in the DNN model to generate a dense vector of real values. The data is split into three sets of 70%, 10%, and 20% for training, validation, and testing, respectively. We note that the ratio of dataset splitting completely depends on the dataset (its size, distribution, and statistics), the application, and the type of model being used. There is no general rule applicable and one needs to start from a reasonable initial ratio and find the optimal one with trial and error. Since our dataset size is relatively large, the 7/1/2 ratio seems appropriate [4].

F. Model Building and Training

We utilize Recurrent Neural Networks (RNN) [25], a class of supervised learning algorithms (cf. Appendix A) designed

for the effective handling of sequential data but also useful for non-sequential data. An RNN has a looping mechanism that allows information to flow from one step (in sequence) to the next. However, RNNs suffer from short-term memory, meaning that they are not able to memorize data for a long time and begin to forget previous inputs. We use a variant of RNN so-called Gated Recurrent Unit (GRU) [30] that is used to solve the problem of short-term memory using a mechanism called gates (cf. Appendix B).

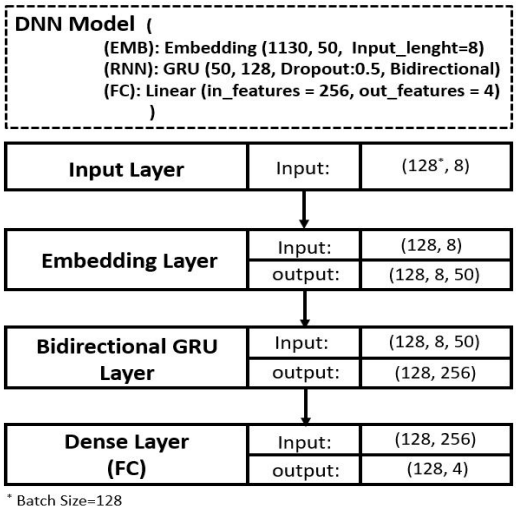

Fig. 4: DNN Classifier Architecture.

We build our deep learning model by assembling layers (in this case, Embedding, GRU, and Fully Connected layers) as shown in Figure 4. Since we do multi-class classification, our model outputs per-class probability scores, and the activation function of the last layer is softmax. We use categorical cross-entropy as the objective (loss) function to train the model.

TABLE II: Model Hyperparameters.

Variable	Setting
Layer Type	[Embedding, Bi-GRU, FC]
Hidden Units	[Bi-GRU:128 ,FC:4]
Optimizer	Adam
Loss Function	Cross-Entropy
Learning Rate	0.001
Dropout Values	0.5
Epochs	100
Batch Size	128

After constructing the model architecture, we take a grid-search approach and loop through predefined hyperparameters and fit the model on the training set. At the end, we select the best performing parameters, as shown in Table II. The training process involves making a prediction and calculating how incorrect the prediction is, and then updating the model parameters to minimize prediction error and to improve the model prediction. We repeat this process until the ML model has converged and can no longer learn from training data. The training process finishes an epoch once the entire training dataset has been exposed to the model. At the end of each epoch, we use the validation dataset to evaluate model performance and see how well the model is learning. The final evaluation is done using unseen batches of test data, all at once, in one round.

V. Evaluation

To evaluate FedCRI, we apply it on nine real-world datasets each of which originates from a different Service Provider from diverse backgrounds such as financial, payments, and insurance services. We also detail an appropriate set of metrics to evaluate the performance of the trained ML models.

A. Datasets

We show detailed information about the number of End Users in each Service Provider's dataset in Table III. Each Service Provider has different distribution in terms of the number of End Users, Mobile OS, and devices that ensure a diverse distribution of feature sets over different Service Providers. This ensures that the local models do not learn the same information from each other. The datasets include data from nine Service Providers labeled A-L with altogether 23.8 million of End Users. Certainly, the non-anonymized user datasets cannot be made publicly available due to obvious ethical concerns and privacy regulations. Yet we share some useful insights about data distributions. Figure 5 depicts the distribution of unique risk entries (single/combined Risk-IDs without duplicates) and Figure 6 shows the distribution of unique false positives for all Service Providers.

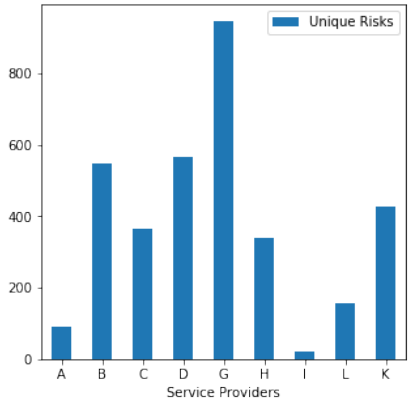

Fig. 5: The distribution of unique risk entries across Service Providers

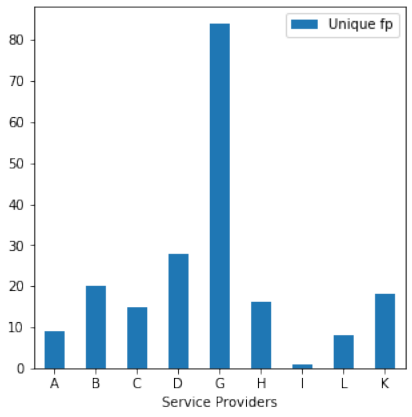

Fig. 6: The distribution of unique false positives across Service Providers

TABLE III: Dataset Overview - Number of End Users by Service Provider

	Service Providers								
	A	B	C	D	G	H	I	K	L
Android	134K	1.4M	450K	1.2M	9.3M	1.4M	2K	1.3M	135K
iOS	100K	1.6M	650K	743K	3.3M	910K	2K	1.1M	95K
Total	234K	3M	1.1M	1.94M	12.6M	2.3M	4K	2.4M	230K

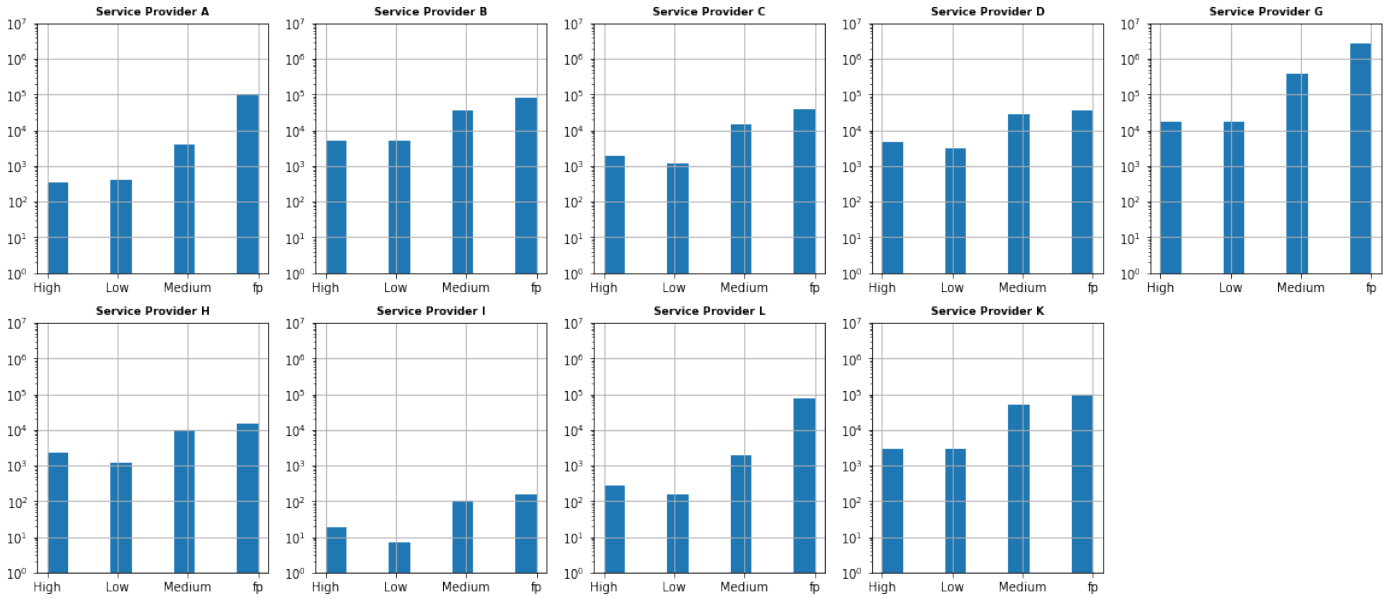

Fig. 7: Label distributions across Service Providers

In addition, Figure 7 demonstrates the distributions of risk labels for each Service Provider in log scale for better visibility. These plots show how false positive (cf. Section IV-D) and risk categories are distributed among different Service Providers. As it is shown in the figure, some Service Providers such as G, with a large number of End Users (See Table III), have a high number of false positives. The reasons for this are likely to be a high user quantity and OS version diversity in the dataset. We cannot disclose more information (i.e., Risk-IDs and Mobile platform/architecture) in the datasets due to the risk of leaking information about Service Providers and potentially causing reputation damage.

B. Evaluation Metrics

To evaluate the performance of the trained models we use common performance metrics such as Precision, Recall, and F1-score. We also utilize False Positive, True Positive, and False negative rates which detail the performance of the models per predicted risk labels. The selection of these metrics is motivated by the fact that Service Providers' datasets are heavily imbalanced, i.e., the number of labels for each class is very different (See Figure 7). Therefore, the commonly-used measures of Accuracy and ROC Curve (AUC) are not reliable performance metrics for evaluating the performance of the learned model [8].

We define basic terminologies as shown in Table IV. They are needed to understand performance evaluation metrics. We define the performance metrics as follows.

TABLE IV: Basic terminologies definition.

Measures	Definition
TN	The actual value is False, and ML predicts False
FP	The actual value is False, and ML predicts True
FN	The actual value is True, and ML predicts False
TP	The actual value is True, and ML predicts True

- False Positive Rate (FPR): The ratio between the wrongly classified negative samples to the total number of negative samples.

$$FPR = \frac{FP}{TN + FP} \tag{1}$$

- False Negative Rate (FNR): According to formula 2, it shows the ratio of positive samples that were incorrectly classified.

$$FNR = \frac{FN}{TP + FN} \tag{2}$$

- True Positive Rate (TPR): It defines the ratio of positive samples that were correctly classified.

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

- Precision: Fraction of observations predicted to be relevant that really are relevant, i.e., the relation of the number of true positives (TP) to the sum of true positives and false positives (FP):

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

- Recall: Denotes the fraction of observations predicted by the algorithm to be relevant out of the total set of relevant observations, i.e., number of true positives in relation to the sum of true positives and false negatives:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

- F1-score: Harmonic mean of Precision and Recall calculated as:

$$\text{F1-score} = 2 \times \frac{Precision \times Recall}{Precision \ + Recall} \tag{6}$$

C. Experimental Setup

We implement the federated learning algorithm using the PyTorch deep learning framework [3]. All experiments are conducted on a Tensorbook (Ubuntu 18.04.5 LTS OS) having Intel Core i7 processor, with 8 and 32 GB of RAM for GPU and CPU, respectively.

D. Evaluation Results

We conduct a set of experiments to evaluate our approach in different scenarios. Experiments are divided into two setups. In the first, client-wise setup, we train separate risk detection models for each Service Provider using their local dataset and evaluate their performance against both the local dataset and on datasets of other Service Providers. In the second, federated setup, we utilize FL to train a global model that aggregates the local models of individual Service Providers. We then evaluate the performance of the global model on the datasets of all Service Providers. We note that model evaluation is done statically (in one round and with batches of unseen data) after the training process is over.

1) ***Client-Wise Setup****:* We train risk detection models on each Service Provider's training dataset using hyperparameters shown in Table II and evaluate them against a validation dataset consisting of hold-out (unseen) data items in each Service Provider's local dataset. We take Service Provider G as an example since its dataset is the largest with more than 12 million of End Users. Then, we plot the training and validation learning curves for the trained model. A learning curve is a plot of model learning performance over experience or time. The model is evaluated on the training set and on a hold-out validation set and plots of the measured performance are created to show learning curves. The training learning curves, plotted with blue lines in Figures 8 and 9, show us how well the ML algorithm is learning while validation learning curves, plotted with orange lines, provide an idea of how well the model generalizes. The learning curves show that model G reaches an accuracy of more than 99% in terms of F1-score for both training and validation data while having an insignificant amount of loss. This confirms that the model generalizes well and is neither overfitted nor underfitted.

Figure 10 shows the Confusion Matrix (CM) of trained model G on the hold-out test dataset. Based on the CM, we calculate FPR, FNR, and TPR for each data label in hold-out data. As can be seen in Table V, model G delivers a FPR of almost 0%, while TPR is more than 99.7%.

This demonstrates that the model G can generalize well on unseen data that has more or less the same distribution as

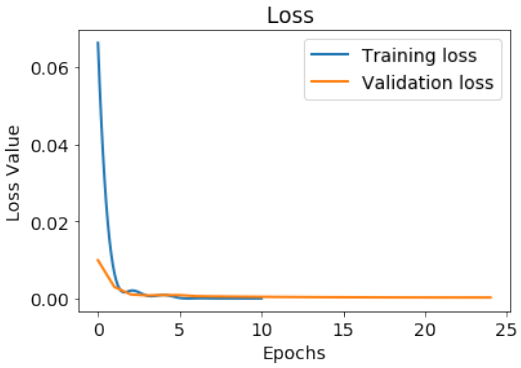

Fig. 8: Learning curve: training and validation loss

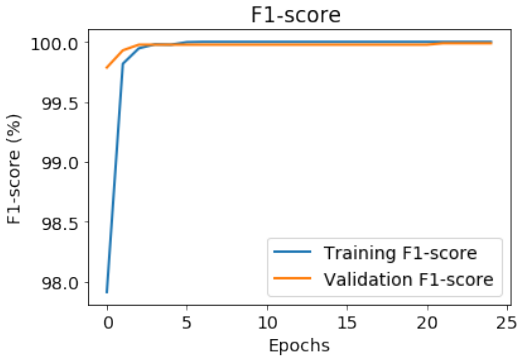

Fig. 9: Learning curve: training and validation F1-score

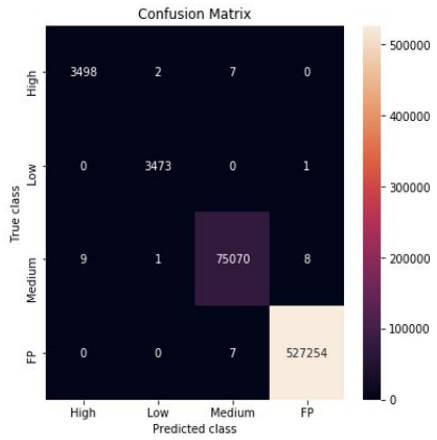

Fig. 10: Confusion matrix of model G on local test data

the training data. To see how well the model can generalize on data having a different distribution, we test the model G against data coming from other Service Providers, for instance, Service Provider C.

Figure 11 shows a CM visualizing the performance of the model G on the dataset of Service Provider C. The corresponding performance measures are shown in Table VI. As can be seen, the model G has a poor performance on the dataset of Service Provider C. This naturally makes sense, since during the training process the model G captures solely the underlying distribution of its own training data (i.e., data of Service Provider G). This confirms that utilizing the federated

TABLE V: Model G - local test data

(%)	High	Low	Medium	FP
FPR	0	0	0	0.01
TPR	99.74	99.97	99.98	100
FNR	0.26	0.03	0.02	0

learning approach can indeed help in training a model that performs well on all Service Providers' datasets.

TABLE VI: Model G - Service Provider C test data

(%)	High	Low	Medium	FP
FPR	21.14	23.19	18.1	31,58
TPR	40.51	20.25	18.94	39.21
FNR	59.49	79.75	81.06	60.79

*2) **Federated Setup**:* In the second scenario, we perform another set of experiments to evaluate the performance of a federated model where all Service Providers contribute to the training of the model. During a training round, each Service Provider trains the model for one epoch (we specify the number of training iterations between the Service Providers and the CRI Provider to be 100). Therefore, the local models are trained for a total of 100 epochs. We note that, as explained in Section IV-B, the datasets were collected over six years in different application domains in multiple European countries. Each Service Provider has different distributions in terms of End Users' numbers and Mobile OS devices (Table III) that ensure a diverse distribution of feature sets over different Service Providers.

We demonstrate the performance of the aggregated FL model for each Service Provider's hold-out dataset in Table IX. In addition, based on the experiments, we show that the FL model can deliver comparable accuracy in terms of F1-score to those of locally trained models. Taking Service Provider A as an example, we can see in Fig. 12 that only the locally trained model on the dataset of A and the FL model deliver a very good F1-score, while other models trained on any datasets except dataset A show poor performance. Using FL each Service Provider can learn from other Service Providers' data and boost its own performance in the detection and classification task. On top of this, FL provides better privacy

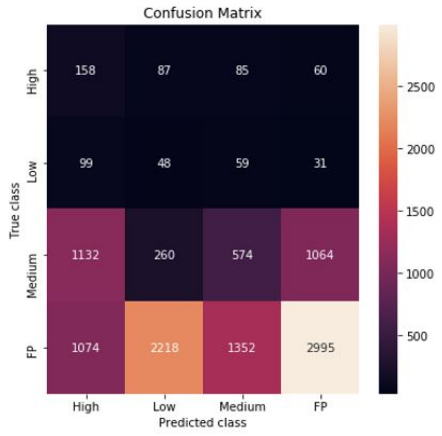

Fig. 11: The CM of the model G on test data of Service Provider C

for Service Providers contributing to the training process as they do not need to share their training data.

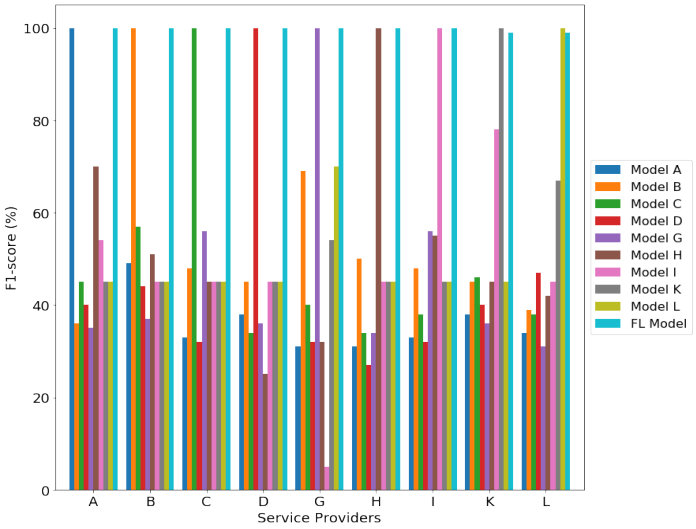

Fig. 12: The performance of the FL model on each Service Providers' data.

Finally, we evaluate the performance of the FL model with regard to the evaluation measures FPR, TPR, and FNR on each Service Provider' hold-out dataset. The major reason for doing this is to compute the metrics per label and see how well the FL model can classify data items into their corresponding labels. Looking at Table XIX, it can be seen that the FL model delivers negligible FPR and FNR while TPR exceeds 97% for all labels in the datasets.

Generalization Check. We conduct further experiments to check the generalization ability of the FL model. We consider two different cases in the following.

Experiment 1. In the first experiment, we train the federated model on 8 Service Providers out of 9 and leave one Service Provider out in the training process. As soon as the training process is finished we test the FL model against the left out Service Providers' data. We repeat this experiment for two different Service Providers, as an example Service Providers **B** and **K** with 3 and 2,4 million end-users respectively. As it can be seen in Figure 13, the FL model does not perform very well on Service Providers left out from the federated training process (still better than the random guess of 25%). The reason is that clearly each Service Provider's data comes from a different distribution and contains completely never-before-seen patterns by other Service Providers. These results give us an indication that federated learning plays an important role to boost risk detection performance in situations in which different Service Providers's data have different distributions. We remind that the FL model was tested against hold-out data of all Service Providers.

Experiment 2. In the second case, we include all Service Providers in the federated training, but in one experiment, we leave out all data entries for Android OS version 10 and above and train the FL model on historical data and use the left-out data for testing the model. The same approach is repeated

TABLE VI: Dataset A

	Prec.	Rec.	F1	Support
High	91	100	96	75
Low	100	98	99	87
Medium	94	99	96	781
FP	100	100	100	20378
Avg.	**100**	**100**	**100**	21321

TABLE VI: Dataset B

	Prec.	Rec.	F1	Support
High	100	100	100	989
Low	100	100	100	966
Medium	100	100	100	7358
FP	100	100	100	15838
Avg.	**100**	**100**	**100**	25151

TABLE VII: Dataset C

	Prec.	Rec.	F1	Support
High	100	100	100	415
Low	100	100	100	227
Medium	100	100	100	2968
FP	100	100	100	7686
Avg.	**100**	**100**	**100**	11269

TABLE VII: Dataset D

	Prec.	Rec.	F1	Support
High	100	100	100	902
Low	100	100	100	650
Medium	100	100	100	5774
FP	100	100	100	7347
Avg.	**100**	**100**	**100**	14673

TABLE VII: Dataset G

	Prec.	Rec.	F1	Support
High	100	100	100	3553
Low	100	100	100	3577
Medium	100	100	100	74805
FP	100	100	100	527396
Avg.	**100**	**100**	**100**	609330

TABLE VII: Dataset H

	Prec.	Rec.	F1	Support
High	100	100	100	445
Low	100	100	100	268
Medium	100	100	100	2022
FP	100	100	100	2859
Avg.	**100**	**100**	**100**	5594

TABLE VII: Dataset I

	Prec.	Rec.	F1	Support
High	100	100	100	2
Low	100	100	100	1
Medium	100	100	100	23
FP	100	100	100	31
Avg.	**100**	**100**	**100**	57

TABLE VIII: Dataset K

	Prec.	Rec.	F1	Support
High	97	100	98	568
Low	99	100	100	584
Medium	99	100	99	10562
FP	100	99	100	18667
Avg.	**99**	**100**	**99**	30381

TABLE VIII: Dataset L

	Prec.	Rec.	F1	Support
High	95	100	97	52
Low	100	100	100	24
Medium	83	97	90	387
FP	100	99	99	15265
Avg.	**99**	**99**	**99**	15729

TABLE IX: Evaluation results in terms of precision, recall and F1-measure (in %) for examined datasets. The support column refers to the number of occurrences of the given label in the dataset.

TABLE X: Dataset A

	High	Low	Medium	FP
FPR	0.03	0	0.25	0
TPR	100	97.7	99.1	99.8
FNR	0	2.3	0.9	0.2

TABLE XI: Dataset B

	High	Low	Medium	FP
FPR	0	0	0	0
TPR	100	100	100	100
FNR	0	0	0	0

TABLE XII: Dataset C

	High	Low	Medium	FP
FPR	0	0	0	0
TPR	100	100	100	100
FNR	0	0	0	0

TABLE XIII: Dataset D

	High	Low	Medium	FP
FPR	0.85	0.01	0.36	0.29
TPR	99.6	100	99.6	99.6
FNR	0.4	0	0.4	0.4

TABLE XIV: Dataset G

	High	Low	Medium	FP
FPR	0	0	0	0
TPR	100	100	100	100
FNR	0	0	0	0

TABLE XV: Dataset H

	High	Low	Medium	FP
FPR	0	0	0	0.04
TPR	100	100	99.95	100
FNR	0	0	0.05	0

TABLE XVI: Dataset I

	High	Low	Medium	FP
FPR	0	0	0	0
TPR	100	100	100	100
FNR	0	0	0	0

TABLE XVII: Dataset K

	High	Low	Medium	FP
FPR	0.07	0.01	0.54	0.04
TPR	100	99.7	99.8	99.4
FNR	0	0.3	0.2	0.6

TABLE XVIII: Dataset L

	High	Low	Medium	FP
FPR	0.02	0	0.5	1.73
TPR	100	100	97.16	99.5
FNR	0	0	2.84	0.05

TABLE XIX: Evaluation measures (in %) for examined datasets

for iOS 11 and above. According to Figure 14, the FL model shows a good generalization ability on completely unseen data entries. For instance, the FL model for iOS data was trained on iOS 6 up to 11 (excluding iOS 11), and then tested against iOS 11 and above. It is noteworthy to mention that datasets of Service Providers A to D had no entries with Android OS version above 9. Therefore, no results were available to plot in the figure.

Besides, the FL model did not perform well on Service Provider L. Taking a close look at its data, we found out that out of 4818 data samples in the testing set there were 4654 unique samples with the false-positive label which are never-before-seen labels for the model and most likely are wrongly predicted by the FL model and due to the high number of misclassified unique data samples the F1-score significantly drops. Hence, the federated re-training process needs to be scheduled between the CRI Provider and the Service Providers, for instance, on a monthly basis or after a new OS version is released.

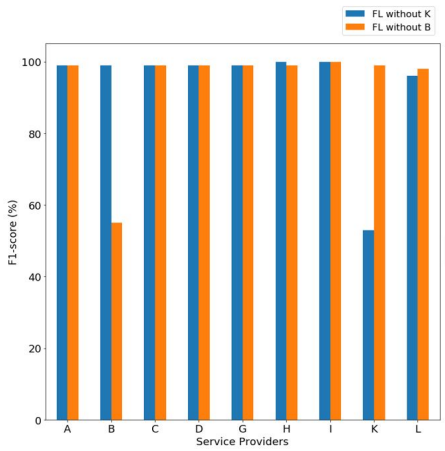

Fig. 13: Generalization check of the FL model: Service Providers B and K are not involved in federated training process.

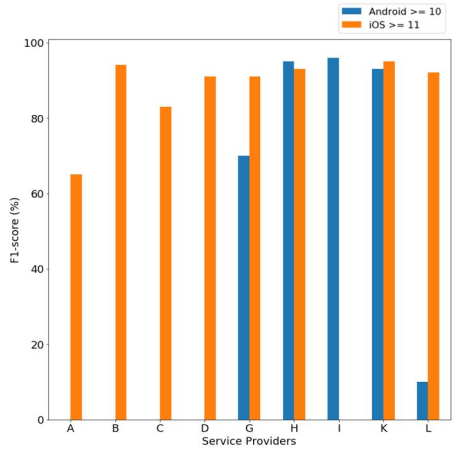

Fig. 14: Generalization check of the FL model. Android OS above 9 and iOS above 10 are not involved in federated training process.

VI. Related Work

To the best of our knowledge, no work targeting the autonomous evaluation of risk indicators exists yet. In this section, we will, therefore, discuss existing work for the individual parts of FedCRI, covering the detection of risks on smartphones [29], [53], [23], [32], [44], [26], [56], coordination of different clients for threat intelligence [28], [9], [17], [13] and leveraging ML for detecting risks and anomalies [49], [42].

Risk Detection. FedCRI uses risk indicators as input feature that describe whether malicious applications are installed on the device. Our approach is different from Intrusion Detection Systems (IDS) for mobile platforms. IDS systems typically detect attacks rather than risks and may suffer from false positives and negatives that are problematic. In addition, they do not share their knowledge with other parties in the form of machine learning models. For the sake of completeness, we discuss further existing approaches for determining risk indicators.

Jailbreak Detection: Several strategies to detect rooted or jailbroken devices have been developed [29], [53]. They focus on hard-coded heuristic checks, e.g., to determine whether certain parts of the file system [29] are writable (such as "`/system`" [53]), or, whether certain program files are indicative of privilege escalation (e.g., the `su` application on android [53]), and typical applications on jailbroken devices (like alternative app stores on iOS devices [29]), are present on the device. In contrast to these, FedCRI performs holistic risk evaluation based on several different risk factors simultaneously for more accurate detection results.

Static Malware Detection: Existing approaches for malware detection on mobile devices evaluate runtime features such as CPU usage, network transmission [23] or system calls for training ML classifiers to recognize malicious applications [33]. Another strategy is to statically analyze metadata as required permissions or information about the developer and its certificate [32].

Dynamic Malware Detection: FedCRI utilizes risk indicators denoting, to see whether malicious apps are installed on a device. Here, different approaches exist to determine the presence of malicious applications. As novel risks may emerge dynamically, it is often not sufficient to rely merely on risk detection rules that have been developed by domain experts. Therefore, multiple approaches have been proposed for recognizing malicious apps using ML [10], [31], [24]. The work of Alzaylaee *et al.* [10] uses API calls and events as input for a NN. Ma *et al.* [31] apply an RNN on the control flow to detect malicious applications.

Anomaly Detection Systems: Andromaly, proposed by Shabtai *et al.* [49], uses runtime features such as CPU usage, network traffic, number of running processes, and battery level to train a naïve Bayes classifier for detecting anomalous behavior. Another anomaly detection system leveraging FL is DÏoT. However, DÏoT considers the devices as black boxes and inspects only the network traffic to make predictions about the state of the monitored device.

Emulator Detection: Similar to jailbreak detection, existing approaches for determining whether the mobile device is actually emulated rely on predefined checks. These approaches check, e.g., if the International Mobile Station Equipment Identity (IMEI) is 0 [44], the presence of certain emulator-specific files [26], or API behavior that indicates the absence of hardware features like vibration [26]. Jing *et al.* developed an approach for automatically identifying suitable heuristics by comparing existing artifacts for emulators and actual devices [26].

In summary, the existing ML-based approaches are restricted to detecting anomalies, showing that there is a risk. In comparison, FedCRI is able to quantify the risk, indicated by its input risk indicators, making it more suitable in real-world applications than the discussed approaches.

Cyber-Threat Intelligence (CTI). Many Service Providers need to deal with similar kinds of threats [28]. It is therefore beneficial to share information about these threats to increase awareness of individual service providers about them [28], [9], [17]. One example for threat sharing is STIX, a formalized language for exchanging Cyber-Threat Intelligence (CTI) information [13]. However, sharing this information also raises privacy concerns, as it might contain sensitive or critical data such as IP and email addresses as well as information about

vulnerabilities [9]. Albakri *et al.* [9] analyzed CTI information and provided guidelines for manually deciding about whether or not to share these data.

Burkhart *et al.* [17] follow a different strategy by focusing on statistical information, like network traffic statistics. They utilize Secure Multi-Party-Computation (SMPC) to securely accumulate CTI information from multiple IDSes, to prevent other parties from learning the values of a single client. Our approach relies on federated learning that avoids sharing raw data about risks altogether and therefore effectively reduces the associated privacy exposure. Further, SMPC is in general not efficient. As the authors admit, for the biggest evaluated setup, consisting of nine separated parties, their system can handle less than 100 clients when correlating the input events. In comparison, using FL allows to distribute of the computational-heavy tasks to the individual clients, s.t. the aggregation server only needs to aggregate the individual models.

Note that our risk identification method targets the autonomous evaluation of risk indicators and utilizes FL to distribute the knowledge in the form of machine learning models among different clients which is not done by other existing works.

VII. Discussion

We first discuss benefits of the federated learning more specifically in our use-case scenario and then explain security and privacy attacks that are relevant to federated learning. Afterwards, we elaborate on privacy-enhancing techniques and backdoor mitigation approaches that can be integrated into our system to alleviate the effects of such privacy and security vulnerabilities.

Apart from communication efficiency and reduced requirements to hardware, one major benefit of FL is to enable mistrusting organizations (i.e., Service Providers) to improve their risk analysis without sharing privacy-sensitive data. Utilizing FL, Service Providers boost their performance in risk detection through learning risk patterns that are not seen locally. Note that Service Providers with fewer data can hugely benefit from those with more data. As we already showed in Figure 12, the federated detection model can deliver comparable (equal or very close with only 1% difference) performance to those of locally trained models on Service Providers' local data but outperforms the local models clearly when evaluating against data belonging to other Service Providers. This clearly shows the power of the FL model in detection of non-local risk patterns in comparison to locally trained models. We recall that such a high detection capability would never have been possible without federated learning.

Despite its benefits, federated learning has been shown to be vulnerable to adversarial attacks (i.e., poisoning/backdoor [20], [41], [57] and inference attacks such as membership inference [46], [51], [36], reconstruction [47], property inference [22]). In backdoor attacks, the adversary stealthily manipulates the global model so that attacker-chosen inputs result in attacker-chosen outputs. A recent study [20] shows how a single client can manipulate FL training process (i.e., adding regularization terms to the cost function) and inject a backdoor into the model that causes green cars to be misclassified as birds. In inference attacks, the adversary aims at learning information about the clients' training local data by analyzing their model updates.

Several works aim at improving data privacy in FL by hindering the aggregator from analyzing clients' model updates. These works typically prevent access to the local updates using secure aggregation protocols that use either encryption [54] or secret sharing [14], [52], [27], [21], or reduce information leakage by applying noise to achieve differential privacy (DP) [19]. Moreover, a number of backdoor defenses such as Krum [15], FoolsGold [18], Auror [50], and AFA [38] have been proposed aiming at separating benign and malicious (backdoored) model updates. Generally, all existing protocols for secure aggregation hinder the aggregators from deploying defenses against backdoor attacks. However, approaches such as FLGUARD [43] and BaFFLe [11] have been proposed that combine secure aggregation with defenses against backdoor injections. We recall that these defenses can be integrated into FedCRI to mitigate these attack vectors.

VIII. Conclusion

In this paper, we present FedCRI, a *Federated Cyber-Risk Intelligence* solution for mobile platforms that enables sharing information about risks in form of actionable machine learning models, instead of utilizing indicators of compromise directly, as done by Cyber-Threat Intelligence (CTI) sharing systems. FedCRI makes use of Federated Learning as an underlying technique to train an effective risk detection model in an efficient and privacy-preserving manner based on information contributed by different mobile Service Providers. Comprehensive evaluation on real-world datasets of 23.8 million End Users were conducted using two scenarios, with risk detection performed by individual Service Providers and using FedCRI platform. Results show that the federated detection model can deliver comparable performance to those of locally trained models on Service Providers' local data but outperforms the local models clearly when evaluating data belonging to other Service Providers. According to the results, FedCRI can achieve better than 99% accuracy in terms of F1-score in most risk detection tasks with a remarkably small number of false positives. Overall, FedCRI is the first solution for mobile service providers that can bridge the gap of security expert's shortage through collaborative risk learning and sharing.

Acknowledgment

We would like to thank Intel Private AI center and BMBF for their support of this research.

References

[1] "Cybox - cyber observable expression." [Online]. Available: http://docs.oasis-open.org/cti/cybox/v2.1.1/cybox-v2.1.1-part01-overview.html

[2] "Openioc," accessed on 20 May 2021. [Online]. Available: https://www.fireeye.com/blog/threat-research/2013/10/openioc-basics.html

[3] "Pytorch - deep learning framework." [Online]. Available: https://pytorch.org/

[4] "Splitting a dataset into train and test sets." [Online]. Available: https://www.baeldung.com/cs/train-test-datasets-ratio

[5] "Stix - structured threat information expression." [Online]. Available: https://oasis-open.github.io/cti-documentation/stix/intro.html

[6] "Taxii - trusted automated exchange of intelligence information." [Online]. Available: https://oasis-open.github.io/cti-documentation/taxii/intro.html

[7] "Threat intelligence report." [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07185213/Kaspersky_Telecom_Threats_2016.pdf

[8] "Which evaluation metric should you choose?" [Online]. Available: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc

[9] A. Albakri, E. Boiten, and R. De Lemos, "Sharing cyber-threat intelligence under the general data protection regulation," in *Annual Privacy Forum*, 2019, pp. 28–41.

[10] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dl-droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, p. 101663, 2020.

[11] S. Andreina, G. A. Marson, H. Möllering, and G. Karame, "Baffle: Backdoor detection via feedback-based federated learning," in *ICDCS*, 2021.

[12] N. Andrew, "Cybersecurity and information sharing: Legal challenges and solutions (congressional research service," *CYCON*, vol. 1-17, 2012.

[13] S. Barnum, "Standardizing cyber-threat intelligence information with the structured threat information expression (stix)," *White Paper, Mitre Corporation*, vol. 11, pp. 1–22, 2014.

[14] J. H. Bell, K. A. Bonawitz, A. Gascon, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *CCS*, 2020.

[15] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NIPS*, 2017.

[16] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017.

[17] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," *USENIX Security*, 2010.

[18] F. Clement, J. M. Y. Chris, and B. Ivan, "The limitations of federated learning in sybil settings," *In Symposium on Research in Attacks, Intrusion, and Defenses (RAID)*, 2020.

[19] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," in *in Foundations and Trends in Theoretical Computer Science*, 2014.

[20] B. Eugene, V. Andreas, H. Yiqing, E. Deborah, and S. Vitaly, "How to backdoor federated learning," *PMLR*, vol. 108:2938-2948, 2020.

[21] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame *et al.*, "Safelearn: Secure aggregation for private federated learning," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 56–62.

[22] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *CCS*, 2018.

[23] H.-S. Ham and M.-J. Choi, "Analysis of android malware detection performance using machine learning classifiers," in *2013 international conference on ICT Convergence (ICTC)*, 2013, pp. 490–495.

[24] R.-H. Hsu, Y.-C. Wang, C.-I. Fan, B. Sun, T. Ban, T. Takahashi, T.-W. Wu, and S.-W. Kao, "A privacy-preserving federated learning system for android malware detection based on edge computing," in *15th Asia Joint Conference on Information Security (AsiaJCIS)*, 2020, pp. 128–136.

[25] L. C. Jain and L. R. Medsker, *Recurrent Neural Networks: Design and Applications*. USA: CRC Press, Inc., 1999.

[26] Y. Jing, Z. Zhao, G.-J. Ahn, and H. Hu, "Morpheus: automatically generating heuristics to detect android emulators," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014, pp. 216–225.

[27] S. Kadhe, N. Rajaraman, O. Koyluoglu, and K. Ramchandran, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," in *arXiv:2009.11248*, 2020.

[28] S. Katti, B. Krishnamurthy, and D. Katabi, "Collaborating against common enemies," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005, pp. 34–34.

[29] A. Kellner, M. Horlboge, K. Rieck, and C. Wressnegger, "False sense of security: A study on the effectivity of jailbreak detection in banking apps," in *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 1–14.

[30] C. Kyunghyun, B. van Merrienboer, G. Caglar, B. Dzmitry, B. Fethi, S. Holger, and B. Yoshua, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

[31] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE access*, vol. 7, pp. 21 235–21 245, 2019.

[32] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, "Android malware characterization using metadata and machine learning techniques," *Security and Communication Networks*, 2018.

[33] M. Z. Mas' ud, S. Sahib, M. F. Abdollah, S. R. Selamat, and R. Yusof, "Analysis of features selection and machine learning classifier in android malware detection," in *International Conference on Information Science & Applications (ICISA)*, 2014, pp. 1–5.

[34] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," *In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

[35] B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *International Conference on Learning Representations (ICLR)*, 2018.

[36] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *IEEE S&P*, 2019.

[37] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector spaceimproving credit card fraud detection using a meta-classification strategy." [Online]. Available: https://arxiv.org/pdf/1301.3781.pdf

[38] L. Munoz-Gonzalez, K. T. Co, and E. C. Lupu, "Byzantine-robust ´ federated machine learning through adaptive model averaging," in *arXiv preprint:1909.05125*, 2019.

[39] C. N. and W. D., "Adversarial examples are not easily detected: Bypassing ten detection methods," *AISEC*, 2017.

[40] National Institute of Standards and Technology (NIST), "Guide to cyber threat information sharing," vol. 150, 2016.

[41] T. D. Nguyen, P. Rieger, M. Miettinen, and A.-R. Sadeghi, "Poisoning attacks on federated learning-based iot intrusion detection system," in *in Workshop on Decentralized IoT Systems and Security*, 2020.

[42] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 756–767.

[43] T. D. Nguyen, P. Rieger, M. H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, A.-R. Sadeghi, T. Schneider, and S. Zeitouni, "Flguard: Secure and private federated learning." [Online]. Available: https://arxiv.org/abs/2101.02281

[44] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware," in *Proceedings of the Seventh European Workshop on System Security*, 2014, pp. 1–6.

[45] Ponemon Institute LLC, "The value of threat intelligence: Annual study of north american & united kingdom companies," 2019. [Online]. Available: https://stratejm.com/wp-content/uploads/2019/08/2019_Ponemon_Institute-Value_of_Threat_Intelligence_Research_Report_from_Anomali.pdf

[46] A. Pyrgelis, C. Troncoso, and E. D. Cristofaro, "Knock knock, who's there? membership inference on aggregate location data," in *NDSS*, 2018.

[47] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "Updates-leak: Data set inference and reconstruction attacks in online learning," in *USENIX Security*, 2020.

[48] M.-D. Seyed-Mohsen, F. Alhussein, and F. Pascal, "Deepfool: a simple and accurate method to fool deep neural networks," *CVPR*, 2016.

[49] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012.

[50] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *ACSAC*, 2016.

[51] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," *In IEEE Symposium on Security and Privacy (S&P)*, vol. 3-18, 2017.

[52] J. So, B. Guler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," in *arXiv:2002.04156*, 2020.

[53] S.-T. Sun, A. Cuadros, and K. Beznosov, "Android rooting: Methods, detection, and evasion," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015, pp. 3–14.

[54] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *AISec*, 2019.

[55] D. Vazquez, O. Acosta, C. Spirito, S. Brown, and E. Reid, "Conceptual framework for cyber defense information sharing within trust relationships," *CYCON*, vol. 1-17, 2012.

[56] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 447–458.

[57] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *ICLR*, 2020.

APPENDIX

A. Machine Learning Algorithm

Machine learning models and algorithms for cybersecurity risks detection can come in the following types:

- **Supervised learning**: is the most common way of implementing machine learning. In a supervised learning model, all input information has to be labeled. A supervised learning model is based on predictive data analysis and is only as accurate as the training set provided for it.
- **Unsupervised learning**: is meant to detect anomalous behaviour in cases where there labeled data is not available at all. An unsupervised learning model continuously processes and analyzes new data and updates its models based on the findings. It learns to notice patterns and decide whether they are parts of legitimate or malicious operations.
- **Semi-supervised learning**: is somewhere between supervised and unsupervised learning and is able to learn from partially labeled data sets (a small amount of labeled data with a large amount of unlabelled data).
- **Reinforcement learning**: is an ML algorithm that allows machines to automatically detect ideal behaviour within a specified context. It constantly learns from the environment to find actions that minimize risks and maximize rewards. A reinforcement feedback signal is required for the model to learn its behaviour.

B. RNN with Gated Recurrent Unit (GRU)

An RNN has a looping mechanism that allows information to flow from one step (in sequence) to the next. This information is the hidden state, which is a representation of previous inputs. However, RNNs suffer from short-term memory. In most real-world problems, a variant of RNN such as Gated Recurrent Unit (GRU) is used to solve the problem of short-term memory using a mechanism called gates. Gates are different tensor operations that learn what information to add or remove to the hidden state, thus they enable learning long-term dependencies. The GRU is the newer generation of RNNs and has fewer gating signals and tensor operations in comparison to an LSTM (Long Short-Term Memory). Therefore, a little speedier to train. It also only has two gates, a reset gate, and an update gate. The update gate (z_t) decides what information to throw away and what new information to add and the reset gate (r_t) determines how much past information to forget. The GRU-RNN model is presented in the form:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{7}$$

$$\tilde{h}_t = g(W_h x_t + U_h(r_t \odot h_{t-1}) + b_n) \tag{8}$$

with the two gates presented as:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{9}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{10}$$

where x_t is the N-dimensional input vector at time t, h_t the hidden state, g is the activation non-linearity, such as sigmoid or Rectified Linear Unit (ReLU), W, U, and b are trainable parameters (weights and bias) and $\odot$ is element-wise multiplication.

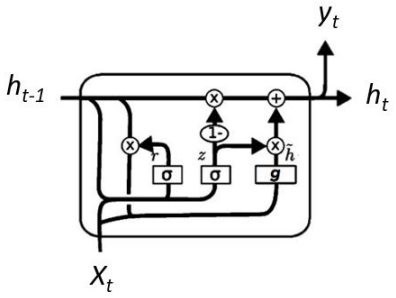

Fig. 15: GRU gating mechanism

C. Federated Learning

In this paper, the local risk models are trained at several Service Providers using locally observed risk indicators obtained from many End Users. We adopt a federated learning approach to aggregate the models from several Service Providers to a global model that is distributed back to individual Service Providers, as it is communication-efficient and privacy-preserving and suited for distributed optimization of Deep Neural Networks (DNN) [34]. FL suits our scenario well since it can cope with settings where data are massively distributed and contributions from participating entities are imbalanced. Algorithm 1 illustrates the FL initialization process and a typical round of learning consisting of the following sequence: CRI Provider randomly initializes the weights of the global model and selects a random subset of members of the federation (here, Service Providers) to receive the global model. Then, each selected Service Provider splits its own training data into several batches and launches a training process.

Algorithm 1 Federated-Averaging. K Service Providers (SP) are indexed by k, β is local batch size, E is number of training epochs and η is learning rate. L indicates local loss function

procedure CRI EXECUTION ▷ run by CRI Provider
  Initialize w_0
  for each round $t = 1, 2, ...$ **do**
    $m \leftarrow$ (determine number of SPs)
    for each $SP_k \in m$ in parallel **do**
      $w_{t+1}^k \leftarrow$ SP_Update(k, w_t)
    end for
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
  end for
end procedure

procedure SP_UPDATE(k,w_t) ▷ run on SP_k
  $\beta \leftarrow$ split data into batches of size B
  for each local epoch i from 1 to E **do**
    for batch $b \in \beta$ **do**
      $w \leftarrow w - \eta \nabla L(w; b)$
    end for
  end for
  return w to CRI Provider
end procedure

During the training, the Stochastic Gradient Descent (SGD) algorithm updates the current weights of the model using the current gradient multiplied by learning rate and computes an updated model which is later sent to the CRI Provider. At the end of each communication round, CRI Provider aggregates these model updates (typically by averaging) to construct an improved global model. Bear in mind that to adapt to possible newly-generated risks, the FL model retraining can be scheduled between the CRI server and SPs on a monthly basis or after releasing a new OS version.