Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone
Aghiles Ait Messaoud, Sonia Ben Mokhtar, Vlad Nitu, Valerio Schiavoni

To cite this version:
Aghiles Ait Messaoud, Sonia Ben Mokhtar, Vlad Nitu, Valerio Schiavoni. Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone. Middleware ’22: Proceedings of the 23rd ACM/IFIP International Middleware Conference, Nov 2022, Québec City, Canada, Canada. pp.335-348, 10.1145/3528535.3565255. hal-03815963

HAL Id: hal-03815963
https://hal.science/hal-03815963
Submitted on 28 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone

Aghiles Ait Messaoud∗
INSA Lyon, LIRIS
Lyon, France
aghiles.ait-messaoud@insa-lyon.fr

Sonia Ben Mokhtar
INSA Lyon, LIRIS, CNRS
Lyon, France
sonia.benmokhtar@insa-lyon.fr

Vlad Nitu†
Integrate.ai
Toronto, Canada
vlad.nitu@integrate.ai

Valerio Schiavoni
University of Neuchâtel
Neuchâtel, Switzerland
valerio.schiavoni@unine.ch

ABSTRACT
Federated Learning (FL) opens new perspectives for training machine learning models while keeping personal data on the users premises. Specifically, in FL, models are trained on the users’ devices and only model updates (i.e., gradients) are sent to a central server for aggregation purposes. However, the long list of inference attacks that leak private data from gradients, published in the recent years, have emphasized the need of devising effective protection mechanisms to incentivize the adoption of FL at scale. While there exist solutions to mitigate these attacks on the server side, little has been done to protect users from attacks performed on the client side. In this context, the use of Trusted Execution Environments (TEEs) on the client side are among the most proposing solutions. However, existing frameworks (e.g., DarkneTZ) require statically putting a large portion of the machine learning model into the TEE to effectively protect against complex attacks or a combination of attacks. We present GradSec, a solution that allows protecting in a TEE only sensitive layers of a machine learning model, either statically or dynamically, hence reducing both the Trusted Computing Base (TCB) size and the overall training time by up to 30% and 56%, respectively compared to state-of-the-art competitors.

∗Also with Ecole nationale Supérieure d’Informatique, Algiers.
†Also with INSA Lyon, LIRIS, CNRS.

CCS CONCEPTS
• Security and privacy → Distributed systems security.

KEYWORDS
Federated Learning, privacy, Trusted Execution Environment, TrustZone

ACM Reference Format:

1 INTRODUCTION
Federated learning (FL) is a distributed machine learning (ML) approach with increasing adoption from academia and industry [9]. The main advantage of this approach is to train ML models while keeping private user data (photos, vocal recording, purchase data, etc.) under the control of their owners, e.g., end users. In a nutshell, a server distributes a global model to several clients. In turn, they train such model locally with their data, and send back the model gradients (i.e., updates) to the server. The latter aggregates the received gradients and iterates by sending the resulting global model to the clients, until a given accuracy is reached. However, the long list of recent privacy attacks (e.g., [35, 39, 59]) demonstrates that sharing model gradients in the context of FL training constitutes a threat to clients’ privacy. Indeed, model gradients may leak sensitive information enabling, for instance, the reconstruction of raw data samples [54, 59] or the learning of hidden features about the data of participating users [35] (e.g., gender, race, etc.). Such attacks threaten the main motivation behind using FL, e.g., the preservation of users’ privacy. Hence, it is paramount to reduce the impact of such attacks,
in particular by securing the model gradients and their computation. Many software solutions exist to secure the gradients on the server-side. For instance, secure aggregation \cite{10} forces the server to only observe the aggregated gradients instead of the individual ones. Differential-privacy \cite{16, 52} (DP) adds noise to the client gradients before sending them to the server. Finally, homomorphic encryption \cite{17} leverages aggregation on homomorphically encrypted gradients. The previous techniques mask the individual raw gradients to the server, to prevent launching privacy attacks from it. However, despite their proven efficiency, these methods do not prevent privacy attacks launched by compromised or malicious clients, and in case of DP, the built model necessarily loses some accuracy. The scenario of compromised or malicious client is particularly threatening when one deals with millions of lines of code (such as the Android OS powering billions of devices \cite{11]), exposing a large attack surface.

To guarantee integrity and confidentiality of FL training, one can rely on Trusted Execution Environments \cite{45} (TEEs). These are recent turn-key solutions that provide program execution with integrity and confidentiality guarantees, available from all major CPU vendors: e.g., ARM TrustZone \cite{41}, Intel SGX \cite{13}, AMD SEV \cite{25}. Typically, TEEs can execute secure enclaves, shielding read and write access to an application’s protected code and data against malicious user applications, compromised OS or system libraries, and even against physical attacks. TEEs have been successfully used in a plethora of application domains: design of DRM (Digital Right Management) to restrict access to intellectual property protected by copyright \cite{23}, emulate a secure NFC card for mobile payment \cite{51}, shield biometric authentication process \cite{7}, etc. In this paper, we rely on ARM TrustZone to mitigate client-side inference attacks in the FL context. Noteworthy, a large portion of ARM-enabled mobile devices offer native support to TrustZone, making it the most pervasive mobile architecture ranging from sensors, wearables and smartphones to supercomputers \cite{4}, encompassing a significant part of the FL clients. One of the challenges in using TrustZone relates to the amount of secure memory typically available to trusted applications (TA). In fact, TA can only use few MBs of secure memory (in the order of 3-SMB \cite{2}), as well as inducing additional CPU overhead.

Existing work (i.e., DarkneTZ \cite{37}) secures a subset of DNN layers inside TrustZone enclaves. However, DarkneTZ lacks support to secure non-successive layers of the underlying DNN model. Our experiments (see §8) prove that such feature is required to protect simultaneously against two privacy attacks targeting separate elements of a model (i.e., the convolutional and the dense parts).

We present GradSec, a framework that leverages TrustZone to secure, at training time, weights or filters of dense \cite{6} or convolutional \cite{12} layers as well as their inputs, outputs and associated computations, by shielding the gradients computation inside enclaves and from external attackers. GradSec can shield non-successive layers, heavily reducing the TEE-related overhead.

GradSec can also change the protected layers across FL cycles, based on a statically fixed probability distribution, to offer a horizontal protection to a model, by going through all subset of layers, without having to secure them all at a given point of time. In a nutshell, GradSec supports two execution modes: (1) static: a subset of DNN layers (i.e., one or two separate slices, where a slice is defined as a set of successive layers), to be protected on client-side, is fixed in advance; and (2) dynamic: the protected layers change over the FL cycle through a sliding window.

We implemented GradSec on top of OP-TEE and deployed on a Raspberry Pi 3B+, with full support for TrustZone. We test the security efficiency of GradSec against three state-of-the-art privacy attacks: Data-Reconstruction Inference Attack \cite{59} (DRIA), Membership Inference Attack \cite{39} (MIA) and Data-Property Inference Attack \cite{35} (DPIA). We later show that static GradSec successfully reduces the impact of DRIA, MIA or both at same time. In the latter case, our approach imposes a smaller CPU and TEE memory overhead than DarkneTZ (8% and 30% smaller respectively), as it only requires to secure a subset of the layers (i.e., the head and the tail) without the intermediate ones. In addition, dynamic GradSec reduces the impact of DPIA while maintaining only few protected layers in the TEE enclave in each FL cycle, ensuring 56% and 8% less CPU and TEE memory overhead respectively when compared to DarkneTZ.

The paper is organized as follows. §2 defines the target problem. §3 presents a background about Federated Learning (FL), the considered state-of-the-art client-side privacy attacks, the basic architecture of ARM TrustZone, as well as DarkneTZ. In §4 we present our threat model. Then, we introduce the concepts of secure FL with an overview of GradSec in §5. We discuss the sources of gradient leakage underlying the design of GradSec in §6. The design of GradSec is detailed in §7. In §8 we evaluate GradSec using state-of-the-art models and against state-of-the-art attacks. We survey related work in §9, before concluding in §10.

2 PROBLEM STATEMENT

Protecting FL systems against inference attacks is becoming an increasingly important problem. While server-side inference attacks have attracted a lot of attention in the research community \cite{10, 17, 52}, mitigation mechanisms against client-side inference attacks have been overlooked. Specifically, to illustrate the harm a malicious client can perform, we considered three state-of-the-art inference attacks: (1) a Data Reconstruction Inference Attack (DRIA) \cite{59}, which aims a reconstructing a data sample (e.g., an image
that was used at training time); (ii) a Membership Inference Attack (MIA) [39], which aims at inferring whether a given data sample has been used (or not) at training time and (iii) a Data Property Inference Attack (DPIA) [35], which aims at inferring sensitive properties (e.g., gender) about the participating users. While these attacks were initially devised on the server side, we adapted them to run on a malicious client. Specifically, we ran the above attacks on clients participating in an FL training process for the LeNet-5 machine learning model [30], an image classification model trained using the CIFAR-100 dataset. Further details on the experimental setup of this experiment as well as further details on the attacks can be found in §7.3 and §3.2, respectively. Results depicted in the first line of Table 1 show that a malicious client successfully manages to run the above attacks.

An effective way to protect FL clients from the above attacks is to use TEEs on the client side, as previously demonstrated by DarkneTZ [37] a system further described in Section 3.4. From line 2 of Table 1, we see that DarkneTZ successfully protects against DRIA and MIA as putting only one layer inside the TEE makes the attack unsuccessful. However, as soon as two attacks are considered at once (DRIA + MIA) or a more complex attack is considered (DPIA), DarkneTZ requires putting four out of the five layers of the model inside the TEE, which incurs an important overhead. In this paper, we propose a solution that allows protecting non-successive layers inside the TEE (static GradSec) as well as a solution for dynamically putting layers inside the TEE in a round-robin (RR) manner (dynamic GradSec). As shown in the two last lines of Table 1, our solution brings an important performance improvement over DarkneTZ, i.e., up to 30% gain in terms of TCB size and up to 56% gain in terms of model training time.

<table>
<thead>
<tr>
<th>SOTA Attacks</th>
<th>DRIA</th>
<th>MIA</th>
<th>DRIA + MIA</th>
<th>DPIA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success measures of the attacks</td>
<td>ImageLoss &lt; 1</td>
<td>AUC=0.95</td>
<td>N/A</td>
<td>AUC=0.99</td>
</tr>
<tr>
<td>Required layers in TEE using DarkneTZ</td>
<td>L2</td>
<td>L5</td>
<td>L2-L3-L4-L5</td>
<td>L2-L3-L4-L5</td>
</tr>
<tr>
<td>Required layers in TEE using GradSec</td>
<td>L2</td>
<td>L5</td>
<td>L2 and L5</td>
<td>2 layers in a RR manner</td>
</tr>
<tr>
<td>GradSec gain in training time</td>
<td>≡</td>
<td>≡</td>
<td>−8.3%</td>
<td>−56.7%</td>
</tr>
<tr>
<td>GradSec gain in TCB size</td>
<td>≡</td>
<td>≡</td>
<td>−30%</td>
<td>−8%</td>
</tr>
</tbody>
</table>

Table 1: Success rate of SOTA attacks against LeNet-5 model (line 1); Layers that need to be put in TEEs using both DarkneTZ and GradSec to protect the model against attacks launched by a malicious client (lines 2 and 3) and the corresponding gain of GradSec compared to DarkneTZ (lines 4 and 5). ≡ indicates similar performance.

3 BACKGROUND

This section provides background on federated learning (§3.1), client-side privacy attacks against which GradSec protects (§3.2), more details regarding ARM TrustZone, the TEE we have chosen to prototype GradSec (§3.3), and finally a description of DarkneTZ, the closest related work (§3.4).

3.1 Federated Learning (FL)

Federated Learning, initially developed for word prediction on the Google’s Gboard keyboard [53], is a distributed approach to machine learning that trains a model on decentralized data. Its principle is to transfer a learning model from a central server to client data (referred to as a “code-to-data” approach) rather than the other way around. Each client then trains the model locally (on its own device), hence the data never leaves its device. Gradients of the resulting model are then sent to a central server, which aggregates them with other clients’ gradients. This design improves the clients’ privacy, a key property feature clearly lacking in centralized machine learning. There exist several industrial applications using FL [14, 31]. Google uses FL on its mobile apps to improve on-device machine learning models, e.g., “Hey Google” in Google Assistant to let users issue voice commands [53]. On healthcare domain, FL is claimed to be the turn-key solution for making the transition from research to clinical practice by enabling the privacy-preserving learning from confidential medical analysis [43]. In manufacturing industries, FL enable multiple companies to train a condition-monitoring system, that monitors a particular condition in machinery (such as vibration, temperature, etc.) to identify changes that could indicate a developing fault, without revealing their respective data and assets [24] and keep their IP confidential. However, despite this growing popularity [19], FL already has been challenged by several privacy attacks, in particular because the shared models can be reverse engineered to identify clients data, or at least some of its features.

3.2 Client-Side Privacy Attacks in FL

Privacy attacks threaten the confidentiality of FL systems in particular for edge devices. Solutions exist to restrict gradients model access on the FL server (secure aggregation [10], Differential-privacy [16, 52], homomorphic encryption [17], etc.), preventing the leak of gradients. However, the client-side OS, system libraries and the device itself may also be compromised by memory scraper malwares [44] that scan
the RAM of infected devices, leading to leakage of the gradients as well as full disclosure of client data. Next, we detail how three state-of-the-art privacy attacks operate.

**Data-Reconstruction Inference Attack (DRIA).** This attack [59] aims at reconstructing an original input data based on the emitted model gradients. This attack assumes a honest-but-curious attacker running in an FL client device, monitoring the FL training process, particularly the gradients produced, before they are sent to the server. The attacker specifically searches for two emitted gradients, produced right after input and those produced due to attacker’s random input, respectively. Then, through an optimisation algorithm (Adam [26], LBFGS [34], etc.) similar to Gradient Descent, the attacker optimizes (minimizes) the difference between the two gradients which mathematically leads to generate a random data that approximate the targeted input data.

**Membership Inference Attack (MIA)** [39] learns whether specific data instances are present in the global model training dataset ($D$). The attacker is a malicious FL client with prior knowledge about $D$, i.e., it knows some data that are part of $D$ ($D_1 \subset D$) and some which aren’t ($D_2 \not\subset D$). The attacker builds a binary classifier by training an attack model on the FL model’ gradients wrt $D_1$ and $D_2$. To infer the membership probability of a data point, the attacker feeds it to the FL model, and computes its corresponding gradients. The generated gradients will be used as input data to the attack model. The latter will output the membership probability of the gradients corresponding to the former data point.

**Data-property Inference Attack (DPIA).** The third attack is DPIA [35]. It infers the probability of presence of a private property ($prop$) seen by the FL model during its local training by one of the FL clients. Like MIA, it assumes a malicious FL client who trains a binary classifier (attack model) on model gradients ($g_{prop}, g_{nonprop}$) against attacker’s auxiliary data ($bad_1, bad_2$). The gradients are computed using different snapshots of the FL model, taken across several FL cycles. To infer the probability of presence of $prop$ among batches of data used to train the global model during an FL cycle, the attacker computes the difference between two consecutive snapshots of the global model to get the aggregated gradients, and feeds those to the attack model.

### 3.3 ARM TrustZone

TrustZone [2, 41] is the TEE for ARM processors. It effectively provides hardware-isolated areas of the processor for sensitive data and code. TrustZone splits the execution in two modes (see Figure 1): (1) Rich Execution Environment (REE) for normal untrusted OS and (2) Trusted Execution Environment (TEE) for secure OS. The REE is fully managed by the regular OS which executes legacy applications without security guarantees. The memory, registers and caches of the

REE are not protected by any hardware mechanism. In contrast, the TEE is managed by the secure OS, which executes trusted applications (TAs) with additional confidentiality and integrity guarantees. TAs rely on services, provided by the TEE kernel, to securely access resources (disk, TCP/IP stack, memory...). In turn, TAs provide API services for legacy applications as well as other TAs. There exist several available implementations for trusted OS with full support for TrustZone. Examples include (i) OP-TEE (Open Portable TEE) [48], the Linaro implementation of secure OS for TrustZone; (ii) Kinibi [50], the TEE OS of Trustonic that uses a microkernel design to enforce isolations between worlds and (iii) Trusty [3], the secure OS implementation of Android that is meant to offer a standard for developing trusted apps for all android devices. The main limit of TrustZone is its limited footprint size, including secure memory size (up to 3-5MB), due to its high cost. Such limitation may prevent users from protecting all the layers of a deep neural network (DNN) inside the enclave, requiring to carefully select which layers we need to secure against a specific attack.

### 3.4 DarkneTZ

DarkneTZ [37] is an open-source DNN framework compatible with TrustZone and the OP-TEE secure OS [48]. It builds upon Darknet [42], an open-source neural network framework implemented in C and with support for CUDA. DarkneTZ allows users to secure only successive layers of a neural network in a TrustZone TEE enclave. Similar to GradSec, it attempts to circumvent the concern of the limited memory size of the TEE enclave by asking the user the ability to protect only a portion of contiguous layers of models. This solution has proven to be effective in countering MIA by protecting only some of the last layers of the model and also DRIA by protecting some of the first layers. However, we show in this paper that DarkneTZ is not effective to protect against both attacks simultaneously since, to do so, non
successive layers of a machine learning model need to be protected into the TEE (e.g., the first layers and the last layers), which is not enabled by the framework as stated in the paper [37]. Hence, to protect against the above attacks simultaneously, DarkneTZ requires to secure almost all layers of the underlying model, which generates an important overhead. Moreover, DarkneTZ has not been proved to be effective against DPIA, which we also demonstrate in our evaluation section.

4 THREAT MODEL

We assume an honest-but-curious client-side attacker. The attacker does not tamper with the FL process and message flow, and it does not attempt to modify the normal message exchanges of the protocol. Instead, we assume it has physical access to the client machine, where he can execute processes with high/root privileges. We further assume that the server side of the FL process is secured using server-grade TEEs (such as Intel SGX) [57], with fewer memory restrictions, or by leveraging secure aggregation protocols [8] through multi-party computation.

Similar to DarkneTZ [37], we assume feed-forward neural networks [18], such as fully-connected or convolutional neural networks (CNN) [1], also considered by the privacy attacks described in Section 3.2.

5 SECURE FEDERATED LEARNING

Figure 2 presents an overview of our approach. We consider a set of clients taking part in the FL training process of a given machine learning model. GradSec’s typical workflow runs as follows.

Selection of FL clients. To ensure that our approach of securing the local training is effective, the FL server only samples clients with a TEE-compatible device, discarding those without a TEE (Figure 2-➊). Therefore, a client interrogation step is required before selecting them for an FL cycle. The FL server can ensure the trustworthiness of the FL client code leveraging novel remote attestation support, for instance as provided by [38].

Transmission of the model, hyper parameters and training plan. Clients receive the FL model, the hyperparameters and the training plan (Figure 2-➋). When receiving the FL model, some of its layers should be protected while the others can be left outside the TEE. GradSec puts the protected layers' weights into the TEE enclave directly using the trusted I/O path (TIOP), as described further in §7.3. In both cases, the data used for training is kept in the storage of the FL client using TrustZone’s secure storage [5, 20] to prevent an external entity from performing a direct leakage of data. This step implements the main contribution of GradSec, with a solution that guarantees a successively effective against some state-of-the-art attacks as further discussed in §7.3.

Secure local training. Each FL client trains his model locally with his own data (see Figure 2-➌). We developed two ways to secure the training. In the first one, i.e., Static GradSec, some layers of the model are permanently protected (e.g., from the first FL cycle until the last) against gradient leakage while the others are not. This approach is effective against some state-of-the-art attacks as further discussed in §7.3. In the second approach, Dynamic GradSec, the protected layers change along with FL cycles. This approach is necessary to protect against more complex attacks as discussed in §7.3. In both cases, the data used for training is kept in the storage of the FL client using TrustZone’s secure storage [5, 20] to prevent an external entity from performing a direct leakage of data. This step implements the main contribution of GradSec, with a solution that guarantees a secure local training for the FL clients.

Transmission of the model updates. Finally, at the end of each FL training cycle, the model gradients of each client are sent to the FL server to be aggregated (Figure 2-➍).
6 SOURCES OF GRADIENTS LEAKAGE

This section presents the major sources of gradients leakage potentially arising when training a model in a FL context. GradSec is designed in a way to circumvent these flaws. Table 2 summarizes the notations of data and operations related to the FL model training. We use the popular Loss function for multi-class classifiers, i.e., the Categorical Cross-Entropy [40].

In the following, we explain how the gradients of a neural network might leak, a required step to propose the countermeasures contributed by GradSec. Specifically, we identify two major flaws.

1st Flaw: Computing the difference between consecutive states of a model. Once the FL client receives the global model from the FL server, it trains it locally with its private data over some local iterations (epochs) and updates the weights of each layer $l$, following the gradient descent formula (SGD):

$$W_{l+1}^t = W_l^t - \lambda dW_l$$ with $t$ the current epoch (1)

With access to the weights of layer $l$, an attacker can exploit the following simple formula to infer its gradients.

$$dW_l = \frac{W_{l+1}^t - W_l^t}{\lambda}$$ (2)

Exploiting formula (2) is our 1st Flaw.

2nd Flaw: Tracking the back propagation computation flow. The injection of a batch of data $X$ into the model initiates two sequential computations: (i) forward propagation and (ii) backward (back) propagation. The computation of the gradients naturally takes place during the back propagation. It consists in a sequence of operations, involving the weights of each layer, that starts from the last layer and propagates backward until the first, in order to compute layer gradients in a descending fashion. The formulas for calculating the gradients of a layer $l$ during back propagation are, for each type of layer as follows:

For a dense layer:

$$dW_l = \delta_{l} A_{l-1} = \begin{cases} \frac{1}{m}(\hat{Y} - Y).A_{n-1} & \text{if } l = n \\ ((W_{l+1}.\delta_{l+1}) \ast f'(Z_l)).A_{l-1} & \text{if } l < n \end{cases}$$ (3)

For a convolutional layer:

$$dW_l = \delta_{l} \ast A_{l-1} = ((W_{l+1}.\delta_{l+1}) \ast f'(Z_l)) \ast A_{l-1} \text{if } l < n$$ (4)

For this layer, the $l=n$ case is removed because a convolutional layer cannot be at the end of a classifier.

Exploiting formulas (3) and (4) constitutes our 2nd Flaw.
We offer the ability to customize the protection probability for each layer against DPIA, where the sensitivity toward the attack, as it needs several FL cycles to collect as many gradients as possible during the model evolution. Instead, DPIA is a long-term attack, as it needs several FL cycles to collect as many gradients as possible during the model evolution. We deploy Dynamic GradSec against DPIA, to observe the impact in changing the protected layers along the FL cycles. Table 3 recap these configurations. Table 4 details the architecture of each model.

8 EVALUATION

This section presents the experimental evaluation of our GradSec prototype. We consider two distinct ML models and real-world datasets. To evaluate the performance of GradSec, we choose the models and the datasets where each attack performs the best so as to measure the real efficiency of GradSec. We launched DRIA and MIA against the LeNet-5 [30] (4 convolutional layers and 1 dense layer) and AlexNet [29] (5 convolutional layers and 3 dense layers) models using CIFAR-100 [28]. We rely on LeNet-5 [30] using the LFW dataset [22] to launch DPIA. Our threat model consider DRIA and MIA as single-shot attacks, i.e., they can be performed by an attacker in one FL cycle. Instead, DPIA is a long-term attack, as it needs several FL cycles to collect as many gradients as possible during the model evolution. We deploy Dynamic GradSec against DPIA, to observe the impact in changing the protected layers along the FL cycles.
Table 4: Architecture of LeNet-5 and AlexNet. Conv2D: Convolutional layer; MP2: 2*2 MaxPool layer. FS/S/P: filter size/strides, pad.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th>#Filters</th>
<th>FS/S/P</th>
<th>in. size</th>
<th>out. size</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>Conv2D</td>
<td>12</td>
<td>5*5/2/0</td>
<td>32<em>32</em>3</td>
<td>16<em>16</em>12</td>
</tr>
<tr>
<td>L2</td>
<td>Conv2D</td>
<td>12</td>
<td>5*5/2/2</td>
<td>16<em>16</em>12</td>
<td>8<em>8</em>12</td>
</tr>
<tr>
<td>L3</td>
<td>Conv2D</td>
<td>12</td>
<td>5*5/1/2</td>
<td>8<em>8</em>12</td>
<td>8<em>8</em>12</td>
</tr>
<tr>
<td>L4</td>
<td>Conv2D</td>
<td>12</td>
<td>5*5/1/2</td>
<td>8<em>8</em>12</td>
<td>8<em>8</em>12</td>
</tr>
<tr>
<td>L5</td>
<td>Dense</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>768 100</td>
</tr>
</tbody>
</table>

Table 4: Architecture of LeNet-5 and AlexNet. Conv2D: Convolutional layer; MP2: 2*2 MaxPool layer. FS/S/P: filter size/strides, pad.

DPIA rely on a dataset of leaked gradients ($D_{grad}$), built by the attacker. To mimic the layer-level gradient confidentiality offered by a TEE enclave, we simply delete from $D_{grad}$ all the gradients columns relative to a protected layer since the latter are considered as unavailable for an attacker located in the normal world. In practice, to dynamically change the protected layers at each FL cycles, we inject the required configuration in $D_{grad}$ only if the moving window MW should protect the concerned layer for the given FL cycle.

To measure the performance impact at deployment of GradSec, we implement and evaluate GradSec using Raspberry Pi 3B+, a popular yet representative single-board device, equipped with Broadcom BCM2837B0 (ARM Cortex A53 quad core @ 1.4GHz, 1GB LPDDR2) to mimic an FL client which trains a model. We use the latest stable release of OP-TEE [48], a secure OS with TrustZone support. We leveraged the latest stable release of DarkneTZ [37] as a privacy-preserving deep learning framework to build GradSec. For static GradSec, we extended DarkneTZ to protect two slices of non-successive layers, in order to efficiently protect against DRIA and MIA simultaneously. For dynamic GradSec, we rely on the vanilla DarkneTZ implementation.

### 8.2 Security Analysis

We quantify the performance of DRIA using the Image Loss metric, i.e., the euclidean distance between the attacker’s inferred image and the original FL client image fed to the model. We measure the performance of MIA and DPIA using AUC, i.e., an aggregated measure of the attack model performance considering all the possible classification thresholds. It is statistically consistent and more discriminating measure than accuracy [33]. An attack model with an AUC of 0.5 is considered as inefficient and performing a random guess regardless the classification threshold.

DRIA. Securing the first layers (especially the 2nd layer) with static GradSec is sufficient to make the attack fail in both our models. The attacker obtains a reconstructed yet blurry image with a large Image Loss (see Figure 5). Even if we were unable to reproduce a clear image with a non-protected AlexNet model, protecting layers inside a TEE enclave (in particular the second layer), still makes DRIA performs even worse. Indeed, for reconstructing visual data fed to a neural network, convolutional layers are the most suitable target for an attacker since they capture more of the visual features of data. Specifically, protecting the firsts convolutional layers (L1 + L2) have the highest impact against DRIA, since those are used to extract the low-level visual features (e.g., image edges). Instead, the tail layers extract high-level features (e.g., color of eyes, shape complexity, etc.). By preventing the attacker from getting the low-level features, that are the support for the high level ones, it fails at rebuilding the input features.

Figure 5: ImageLoss inferred images with various protected layers using Static GradSec.
with static (a) AUC of MIA in LeNet-5 with various protected layers using Static GradSec.

Figure 6: AUC of MIA with various protected layers using Static GradSec.

Takeaway: to mitigate DRIA, one should focus on securing the first layers of the convolutional part of the models.

MIA. Securing the last layer (i.e., the 5th layer) in LeNet-5 with static GradSec lowers the attack’s AUC from 0.95 to 0.85. Protecting additional tail layers show little benefits, as the AUC attack only drop by 5% with last 4 layers protected (see Figure 6 (a). For AlexNet, the focus of an attacker should also be on the gradients of tail layers (L6, L7 and L8) that constitutes the dense part of the model, to succeed in MIA with an AUC of 0.79. Thus, protecting these specific layers is effective to lower the AUC until 0.59, assuming the attacker would exploit the gradients of the convolutional part of the model. To decrease the number of the protected layers, shielding layer L6 is sufficient to reduce the AUC to 0.56 if the attacker focuses on exploiting the two remaining dense layers. Figure 6 (b) summarizes these results.

Takeaway: to reduce MIA impact, securing layers of the Dense part usually found at the end of a model remains more efficient than securing the layers of the convolutional part.

DPIA. Protecting individual layers using static GradSec proves ineffective against this attacks. We systematically hit an AUC rate of 0.99 regardless the protected layer. While it is possible to lower the AUC down to 0.85 with 4 protected layers inside the enclave, this would consumes a large share of the available secure memory (1.84MB for LeNet-5, see Section 8.3), a scarce resource shared with other Secure Applications, and heavily impacts the training time (see 8.3). Instead, dynamic GradSec achieves a better AUC rate (0.78) with only two simultaneous layers inside the enclave (sizeMW = 2) and with an appropriate choice of VMW ([0.2, 0.1, 0.6, 0.1]). Table 5 resumes these results. To find the best distribution of VMW for each value of sizeMW, we train different instances of the attack model (random forest) on a gradient train set with differently located missing data to reflect changing protected layers across the FL cycles, according to the chosen VMW. The incomplete columns of the train set are filled with the mean strategy. We evaluate each attack model instance on a gradient validation set and we retain the VMW distribution of the worst instance. Finally, we test VMW on a gradient test set that reflects unseen gradients.

8.3 Overhead

We use two metrics to measure the performance impact of GradSec on the LeNet-5 model. Firstly, we consider the model training time of one FL cycle per configuration of protected layer(s). We break down this in three parts: (1) computation in user-space, spent outside the TrustZone enclave, (2) kernel-space time, spent inside the enclave during the training process, and (3) allocation time, i.e., time to allocate the TEE memory for the received weights, before starting the training process. We rely on the real-time dashboard provided by DarknetTZ to measure the user and kernel time. For the TEE memory allocation time, we instrumented the code with timers around the memory allocation for the model weights.

Secondly, we measure the maximum TEE memory usage for each configuration of protected layer(s). GradSec doesn’t provide a direct tool to measure the used TEE memory size and doing so from the normal world is prohibited due to the restriction imposed by the secure world. To address this issue, we locate all the DarknetTZ code parts where the TEE memory allocation is triggered through the malloc / calloc instructions and summed the sizes of allocated regions. In the case of dynamic GradSec, we computed the previous metrics for each sizeMW and using the best VMW distribution (i.e., the one achieving the highest security). Since the protected layers change over the FL cycles, we use a weighted average over all the different protection positions covered by the MW to compute the training time. For the TEE memory usage, we just included the consumption of the most expensive combination among the different possibilities allowed.

<table>
<thead>
<tr>
<th>Static GradSec</th>
<th>None</th>
<th>L4</th>
<th>L3+L4</th>
<th>L3+L4+L5</th>
<th>L2+L3+L4+L5</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.95</td>
<td>0.85</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic GradSec</th>
<th>None</th>
<th>MW=2</th>
<th>MW=3</th>
<th>MW=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC</td>
<td>0.99</td>
<td>0.78</td>
<td>0.77</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Table 5: AUC of DPIA using GradSec
Table 6: CPU Time and TEE memory usage of GradSec (LeNet-5, CIFAR-100, batch-size = 32)

<table>
<thead>
<tr>
<th>Protected layers</th>
<th>CPU Training time (User+ Kernel+ Allocation)</th>
<th>TEE Memory Usage (at exec) in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without (Baseline)</td>
<td>2.191s + 0.021s + 0s</td>
<td>0</td>
</tr>
<tr>
<td><strong>Static GradSec</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>1.886s + 0.738s + 0.09s (19% overhead)</td>
<td>1.127</td>
</tr>
<tr>
<td>L2 (against DRIA)</td>
<td>1.672s + 0.652s + 0.34s (20% overhead)</td>
<td>0.565</td>
</tr>
<tr>
<td>L3</td>
<td>1.696s + 0.674s + 0.34s (22% overhead)</td>
<td>0.286</td>
</tr>
<tr>
<td>L4</td>
<td>1.691s + 0.673s + 0.34s (22% overhead)</td>
<td>0.286</td>
</tr>
<tr>
<td>L5 (against MIA)</td>
<td>2.044s + 0.187s + 4.68s (212% overhead)</td>
<td>0.704</td>
</tr>
<tr>
<td>L2+L5 (against DRIA+MIA)</td>
<td>1.561s + 0.846s + 5.02s (235% overhead)</td>
<td>1.269</td>
</tr>
<tr>
<td><strong>Dynamic GradSec</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW=2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1+L2</td>
<td>1.323s + 1.331s + 0.43s (39% overhead)</td>
<td>1.692</td>
</tr>
<tr>
<td>L2+L3</td>
<td>1.139s + 1.275s + 0.68s (40% overhead)</td>
<td>0.851</td>
</tr>
<tr>
<td>L3+L4</td>
<td>1.154s + 1.269s + 0.68s (39% overhead)</td>
<td>0.572</td>
</tr>
<tr>
<td>L4+L5</td>
<td>1.507s + 0.808s + 5.02s (231% overhead)</td>
<td>0.99</td>
</tr>
<tr>
<td>AVG ((V_{MW}=[0.2, 0.1, 0.6, 0.1])) (against DPIA)</td>
<td>1.21s + 1.236s + 1.064s (58.3% overhead)</td>
<td>1.692</td>
</tr>
<tr>
<td>(AVG=0.866)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW=3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1+L2+L3</td>
<td>0.708s + 2.081s + 0.77s (61% overhead)</td>
<td>1.978</td>
</tr>
<tr>
<td>L2+L3+L4</td>
<td>0.807s + 1.743s + 1.02s (61% overhead)</td>
<td>1.137</td>
</tr>
<tr>
<td>L3+L4+L5</td>
<td>1.003s + 1.418s + 5.36s (251% overhead)</td>
<td>1.276</td>
</tr>
<tr>
<td>AVG ((V_{MW}=[0.1, 0.1, 0.8])) (against DPIA)</td>
<td>0.964s + 1.517s + 4.467s (213% overhead)</td>
<td>1.978</td>
</tr>
<tr>
<td>(AVG=1.332)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW=4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1+L2+L3+L4</td>
<td>0.170s + 2.754s + 1.11s (82% overhead)</td>
<td>2.264</td>
</tr>
<tr>
<td>L2+L3+L4+L5</td>
<td>0.985s + 1.420s + 5.7s (266% overhead)</td>
<td>1.841</td>
</tr>
<tr>
<td>AVG ((V_{MW}=[0.1, 0.9]))</td>
<td>0.904s + 1.553s + 5.241s (247% overhead)</td>
<td>2.264</td>
</tr>
<tr>
<td>(AVG=1.883)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

by the MW. The results are summarized in the Table 6 and Figure 7, and discussed in the remainder of this section.

**DRIA.** To protect LeNet-5 against DRIA, static GradSec should focus on securing the layer L2. We observe a 20% increase in the training time when compared to training the model outside the enclave, while the required TEE memory size is about 0.57MB.

**MIA.** To protect LeNet-5 against MIA, static GradSec should focus on securing the last layer (L5). By doing so, we measure an important increase in the training time (i.e., 212%) when compared to training the model outside the enclave, while the required TEE memory size is about 0.70MB. This significant overhead for the training time is mainly caused by the memory allocation for L5 which has a fairly large number of parameters (76.8K).

**Grouped protection.** Protecting LeNet-5 against DRIA and MIA simultaneously require to secure the sensitive layer of each attack with Static GradSec, i.e., L2 and L5. Unsurprisingly, the training time overhead increases by 235%, while the required TEE memory is 1.27MB, the fairly heavy L5 being responsible for the large majority of this overhead.

**DPIA.** To mitigate DPIA with dynamic GradSec, the best option in terms of security and overhead is to use size_MW = 2 with \(V_{MW} = [0.2, 0.1, 0.6, 0.1]\). Indeed, as we have seen in section 8.2, this configuration offers better protection than DarkneTZ while requiring only two simultaneously protected layers for each cycle. The overhead to mitigate DPIA is a 53% longer training time and 1.692 MB of TEE memory usage in the worst case (when the MW protects L1+L2).

**Comparison with DarkneTZ.** Finally, we compare GradSec against DarkneTZ in terms of training time and TEE memory usage. The results are shown Figure 8. We use static GradSec to offer simultaneous protection against DRIA and MIA with the cost of 2 protected layers (L2 and L5). To offer a comparable level of protection with DarkneTZ, not only
L2 and L5 but also all the intermediate layers (L3 and L4) should be protected in secured memory. As one can imagine, static GRADSec offers a better training time (8.3%) and significantly less TEE memory usage (30%), due to the reduced numbers of layers to protect. Concerning DPIA, we compare dynamic GRADSec with size\textsubscript{MW} = 2 and the most appropriate V\textsubscript{MW} (i.e., [0.2, 0.1, 0.6, 0.1]), against DarkneTZ with the layers L2 up to L5 in the TEE enclave. Our flexible and more customizable approach allows us to find suitable size\textsubscript{MW} and V\textsubscript{MW} that deliver better level of protection than DarkneTZ and with less overhead. Indeed, changing the protected layers dynamically allows us to reduce the training time by 56% compared to DarkenTZ, mainly because GRADSec does not require to allocate the necessary TEE memory for the biggest layer (L5) in each FL cycle. The necessary TEE memory for dynamic GRADSec varies according to the layers protected by the Moving Window in the current cycle. The most expensive configuration is when the Moving Window secures L1+L2. However, even in this configuration, dynamic GRADSec consumes 8% less memory than DarkneTZ.

9 RELATED WORK

We survey related work in the area of secure, confidential or privacy-preserving federated learning systems. PPFL. Fan Mo’s PPFL (Privacy-preserving Federated Learning) [36] is a TEE-based framework for mobile systems that limits privacy leakage in FL. Similar to GRADSec, PPFL aims at hiding the gradient updates of the model inside a TEE enclave on the client side for local training as well as on the server side for secure aggregation. It leverages layer-wise training to train each model’s layer separately inside the trusted area until its convergence. The PPFL protocol implies the modification of the FL process to support layer-wise distribution of the model to the clients instead of distributing the whole model at once. It also implies the constant use of TEEs to train each layer of the model. While this solution is sound from a privacy perspective, it also incurs a substantial overhead in terms of training time by design as model layers are trained in a sequential manner.

Gecko. Gecko training [15] is a methodology developed to bring privacy-aware deep learning for embedded systems. Its goal is to offer membership-privacy by design in neural
networks by leveraging quantization to reconcile privacy, accuracy and efficiency. The main objective of the Gecko design is to mitigate blackbox MIA. However, contrary to GradSec, no evidence is given regarding the resilience of the proposed solution against other attacks we dealt with (e.g., DRIA, DPIA).

BatchCrypt [55] is an Homomorphic Encryption (HE) method for Cross-Silo FL. It aims at reducing the cost in computation and communication of HE when the gradients are homomorphically encrypted. Instead of encrypting individual gradients with full precision, BatchCrypt encodes a batch of quantized gradients into a long integer and encrypt it in one go. To allow aggregation of gradients to be performed on ciphertexts of the encoded batches, authors in [55] propose new quantization and encoding schemes, alongside a new gradient clipping technique. BatchCrypt considerably improves over vanilla HE method and is well suited to ML constraints. However, it lacks support for an end-to-end solution to circumvent the problem of comprised clients whose OS may leak the gradients before being encrypted, in contrast with TEE solutions like GradSec.

Slalom [47] presents a system and solution for high performance execution of Deep Neural Networks (DNNs) in TEEs. In a nutshell, it efficiently partitions DNN computations between trusted and untrusted devices. It leverages both GPU (for efficient batch computation) and a TEE (for minimizing the use of cryptography). Specifically, Slalom is a framework that delegates execution of all dense layers in a DNN from a TEE to a faster, yet untrusted, processor (i.e., typically, a GPU). It requires a lot of pre-computation over known and fixed weights, and hence it only supports private inference and not training. In addition, dense layers may also leak sensitive information, useful for MIA, especially if the computation in which they are involved are computed outside the TEE enclave.

Citadel [56] is a federated learning framework, built on top of Intel SGX enclaves. It relies on two distinguished set of worker enclaves (training and aggregator ones). It employs zero-sum masking and hierarchical aggregation techniques to prevent gradient leaking across such worker enclaves. However, it does not protect against membership-inference attacks, and it is strongly coupled to Intel SGX enclaves. Given the future roadmap of Intel toward server-only deployments of SGX, we believe GradSec to be better suited to be deployed on edge devices in a federated learning system.

10 CONCLUSION AND FUTURE WORK

We presented GradSec, a TEE-based protection mechanism that improves FL privacy guarantees against state-of-the-art inference attacks. GradSec can operate in two modes: static and dynamic. Static GradSec can simultaneously protect against DRIA and MIA attacks with less overhead than DarkneTZ. Dynamic GradSec offers a better protection than DarkneTZ against DPIA while still incurring less overhead. We implemented GradSec on top of the OP-TEE trusted OS, and evaluated its performance on a ARM Cortex-A53 processor with support for TrustZone enclaves. We plan to release GradSec to the research and open-source community.

We intend to extend this work along the following directions. First, we aim at adding support for RNNs (Recurrent neural networks). This would allow us to protect other types of machine learning models (e.g., models dealing with text, voice recordings and time series in general), and validate our approach on highly-sensitive domains (e.g., e-health). Second, we intend to study hybrid deployments, in which TEE-enabled clients are deployed alongside legacy clients without support for TEEs, and for which purely software-based approaches are necessary.

ACKNOWLEDGMENTS

This publication incorporates results from the VEDLiOT project, which received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 957197.

REFERENCES
