Chapter 1

Introduction

1.1 Motivation

In recent years, cyber insurance has emerged as a new form of insurance. A customer seeking cyber insurance is usually looking for protection against a variety of cyber risks, that might include hacking incidents, data leaks, IT service outages, and the like. Asymmetry in information is one of the major problems in the insurance industry. As such, insurers have significantly less information about insured objects (e.g. a vehicle or a home) than their customers. This complicates risk assessment and insurance pricing. To counteract this, insurers would usually hand out questionnaires to their customers to gain more information about the underlying objects. In terms of cyber insurance however, this is not always feasible. Clients are generally reluctant to fully disclose details of their IT infrastructure, and security policies. Customers likely are concerned, that honest answers about poor IT security practices might be used to discriminate against them, either at the time of insurance pricing or at a potential future claim settlement.

The recent advances in Trusted Execution Environments (TEEs) and privacy-preserving machine learning offer new avenues for dealing with the problem of information asymmetry in insurance in general and particularly in cyber insurance. A hardware-protected enclave (e.g. using Intel SGX) could be leveraged for collecting customer questionnaires and subsequently use the received data to train a privacy-preserving machine leaning model. In order for the final model not to leak any information about any participating individual's data, techniques like Differential Privacy (DP) can be used. In this way, the insurance company is able to build a useful aggregate model, while protecting the privacy of each individual customer at the same time. According to recent studies, with such a privacy-preserving system in place, users would be more willing to share private data with data collectors, compared to a situation where the data collector learns the users' full data in plaintext. This also applies to cyber insurance and thus could decrease the problem of information asymmetry.

1.2 Problem overview

A setting of this type is not an easy undertaking. There are two main parties involved, each with different demands: (i) Insurance customers want data privacy under all circumstances. Even a potentially malicious insider at the insurance company must not be able to leak secret data. (ii) The insurance wants good results form the resulting model. Customers or other external parties should not be able to distort the output model or render it useless through interaction with the enclave. There is a number of things that can go wrong which would result in those requirements

being violated. First, even though the training process takes place inside a TEE, leakage of secret information can occur. It is a known issues that TEEs are susceptible to side-channel attacks. Further, even in the case of a total absence of side-channels during the model training, developing a secure deployment strategy is also not an easy task. In order to securely get data into the enclave and the result out of the enclave, the insurance must carefully and transparently initialise the enclave setup. Careful verification must happen. A secure channel must be used. We must ensure that the data is sent to the right enclave, and can only be processed by that exact enclave. Further, questionnaires will not be available all at once, but gradually arrive over time. Therefore storage and provisioning functionality is required. Further, the enclave should of course not accept multiple questionnaires from the same customer and likewise. Even if this doesn't hurt privacy, it decreases model quality. Additionally, safety and redundancy mechanisms are required. The system should not be bound to one single computer. At some point, a computer might crash or break. Or, the insurance might want to transition to newer hardware. This calls for some form of secure persistent state. Accordingly, enclave state related attacks such as rollback attacks, or forking/cloning/interleaving attacks need to be prevented. Finally, we want the possibility to train a model again, and the ability to refine it with newly received customer data. For differential privacy reasons, that will be later discussed, this is not trivial.

Goals (i) *Efficient algorithm*: We want a bug-free and working algorithm to produce accurate performance measurements. Ultimately DP-GBDT should be able to produce usable scores for reasonably small privacy budgets. In other words, the noise added through differential privacy should not completely erase the output model's expressive power. Additionally, even though training will likely not be performed very frequently, its runtime should be reasonable. (ii) *Side-channels resistance*: The DP-GBDT implementation must be adequately hardened. The goal is to protect from all "digital side-channels". This notion sums up all side-channels that carry information over discrete bits. Examples are address traces, cache usage, and data size. We are not trying to eliminate every single bit of leakage. We eliminate just enough to achieve ε-differential privacy. (iii) *Secure deployment*: Our goal is to investigate the challenges of secure deployment of the setup. This includes: Privacy attacks through enclave state rollbacks or fork attacks, provisioning, persistent state, dealing with potential human or machine failures.

- Data collection over an extended time period
- Training on newly acquired customer data to improve the existing model output
- Training on old data after output model loss[2]
- Enclave replication and migration for maintaining persistent state

1.3 Approach

Two prior works form the basis and starting point for this thesis. With DPBoost, Li et al. ([45], 2020) provide the algorithm for DP-GBDT learning. Moreover, Théo Giovanna built a first Python implementation of said algorithm during the course of his master's thesis (submitted Feb. 2021). With the overlying goal of performing DP-GBDT inside an Software Guard Extensions (SGX) enclave, a new C++ implementation has to be created. This will lead to the discovery of several bugs, to substantial runtime improvements. The resulting code is ported into an SGX enclave. Subsequently, manual side-channel hardening on source code level is conducted. The process differs from traditional hardening to some extent. We do not try to eliminate every single bit of leakage. Instead, we remove just enough leakage at the right places to achieve ε-differential privacy. This is much more effective than tool assisted hardening as we leverage our knowledge of the DP-GBDT algorithm. Several data oblivious and constant-time building blocks are created

to replace common and reoccurring operations. The prediction accuracy of the algorithm is evaluated on four real-world UCI standard datasets. The effect of certain hyperparameters is demonstrated. Further we provide runtime measurements that show the impact of our hardening measures. Finally we explore possibilities for secure deployment of our enclave setup. With the help of secure monotonic counters, persistent state is achieved. We highlight the challenges and proposed solutions for the enclave initialisation, data collection, training, enclave replication and migration. We assume that the underlying hardware of the enclave setup is not compromised, is patched and works as expected. Further, all known SGX vulnerabilities with a fix available are patched. Denial of service attacks are also out of scope.

1.4 Contribution

The main contributions of this thesis are:

- Build a C++ DP-GBDT implementation that runs inside an SGX enclave
- Harden the implementation against side-channels
- Evaluate the accuracy and runtime of the algorithm
- Explore possibilities for secure deployment

There are further some smaller contributions that developed along the path: (i) Previous results and implementations of the DP-GBDT method we are using turned out to have several flaws. To our knowledge, this is first time detailed and correct performance results of this algorithm have been generated. (ii) The underlying codebase was designed with a focus on usability and extensibility for future experimentation. (iii) This thesis offers some guidelines on manual (source code level) hardening of a tree based machine learning algorithm. (iv) Similarly, we offer guidance on how port such an algorithm into an enclave. What kind of code changes are necessary and how to divide code into inside/outside of the enclave.

Do we need some results teasers?