Zheng H., Ye Q., Hu H., Fang C., Shi J. (2019) BDPL: A Boundary Differentially Private Layer Against Machine Learning Model Extraction Attacks. In: Sako K., Schneider S., Ryan P. (eds) Computer Security – ESORICS 2019. ESORICS 2019. Lecture Notes in Computer Science, vol 11735. Springer, Cham.
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-29959-0_4

BDPL: A Boundary Differentially Private Layer Against Machine Learning Model Extraction Attacks

Huadi Zheng[1], Qingqing Ye[2,1], Haibo Hu[1], Chengfang Fang[3], and Jie Shi[3]

[1] The Hong Kong Polytechnic University, Hong Kong SAR, China
huadi.zheng@connect.polyu.hk, haibo.hu@polyu.edu.hk
[2] Renmin University of China, Beijing, China
yeqq@ruc.edu.cn
[3] Huawei International
{fang.chengfang, shi.jie1}@huawei.com

Abstract. Machine learning models trained by large volume of proprietary data and intensive computational resources are valuable assets of their owners, who merchandise these models to third-party users through prediction service API. However, existing literature shows that model parameters are vulnerable to extraction attacks which accumulate a large number of prediction queries and their responses to train a replica model. As countermeasures, researchers have proposed to reduce the rich API output, such as hiding the precise confidence level of the prediction response. Nonetheless, even with response being only one bit, an adversary can still exploit fine-tuned queries with differential property to infer the decision boundary of the underlying model. In this paper, we propose boundary differential privacy (ϵ-BDP) as a solution to protect against such attacks by obfuscating the prediction responses near the decision boundary. ϵ-BDP guarantees an adversary cannot learn the decision boundary by a predefined precision no matter how many queries are issued to the prediction API. We design and prove a perturbation algorithm called boundary randomized response that can achieve ϵ-BDP. The effectiveness and high utility of our solution against model extraction attacks are verified by extensive experiments on both linear and non-linear models.

1 Introduction

Recent advance in deep learning has fostered the business of machine learning services. Service providers train machine learning models using large datasets owned or acquired by themselves, and use these models to offer online services, such as face and voice recognition, through a public prediction API. Popular products include Microsoft Azure Face API, Google Cloud Speech-to-Text, and Amazon Comprehend. However, a prediction API call, which consists of a query and its response, can be vulnerable to adversarial attacks that disclose the internal states of these models. Particularly, a *model extraction* attack [19] is able

to restore important model parameters using the rich information (e.g., model type, prediction confidence) provided by the prediction API. Once the model is extracted, an adversary can further apply model inversion attack [7] to learn the proprietary training data, compromising the privacy of data contributors. Another follow-up attack on the extracted model is *evasion attack* [23, 16], which avoids a certain prediction result by modifying its query. For example, a hacker modifies the executable binaries of a malware or the contents of a phishing email in order not to be detected by an antivirus or spam email filter.

There are two state-of-the-art countermeasures against *model extraction* attacks. One is to restrict rich information in the prediction API, for example, by rounding the prediction confidence value to a low granularity. However, even if the service provider completely eliminates this value in the prediction API, that is, to offer prediction label only, an adversary can still defeat this protection by issuing large number of fine-tuned queries and train a replica of the original model with great similarity [13, 19, 16]. The second countermeasure is to detect malicious extraction by monitoring feature coverage [10] or query distribution [9], and stop the service when a certain threshold is reached. However, since we cannot preclude user collusion, all queries and responses must be considered aggregately, which leads to significant false positive cases and eventually the early termination of service.

To address the disadvantages, in this paper we propose a new countermeasure that obfuscates the output label of a prediction response. There are three main concerns when designing this obfuscation mechanism. First, the accuracy of prediction API is highly correlated with the degree of obfuscation — if obfuscation needs to be applied to most queries, the utility of the machine learning service will degrade severely. Second, the obfuscation mechanism should be independent of both the adversarial attacks stated above and the underlying machine learning models. Third, the obfuscation mechanism should be customizable. That is, it should allow user-defined parameters that can trade utility for model privacy or vice versa.

Our key observation is that most model extraction attacks exploit fine-tuned queries near the decision boundary of a machine learning model. The responses of these queries disclose the details of model parameters and therefore should be obfuscated with priority. To this end, we propose a **b**oundary **d**ifferentially **p**rivate **l**ayer (BDPL) for machine learning services. BDPL provides a parameterized approach to obfuscate binary responses whose queries fall in a predefined boundary-sensitive zone. The notion of differential privacy guarantees the responses of all queries in the boundary-sensitive zone are indistinguishable from one another. As such, adversary cannot learn the decision boundary no matter how many queries are issued to the prediction API. On the other hand, the majority of queries from normal users are far away from the decision boundary and therefore are free from obfuscation. In this way, we can make the best use of the obfuscation and retain high utility of the machine learning service. To summarize, our contributions in this paper are as follows.

- We propose a new protection mechanism, namely, boundary differential privacy, against model extraction with fine-tuned queries while balancing service utility and model protection level.
- We develop an efficient method to identify queries in the boundary-sensitive zone, and design a perturbation algorithm called boundary randomized response to guarantee boundary differential privacy.
- We conduct extensive empirical study on both linear and non-linear machine learning models to evaluate the effectiveness of our solution.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries for machine learning and model extraction. Section 3 elaborates on the threat model and problem definition with boundary-sensitive zone and boundary differential privacy. Section 4 presents the details of boundary differentially private layer. Section 5 introduces evaluation metrics and shows the experimental results of BDPL against model extractions. Section 6 reviews the related literature, and Section 7 concludes this paper and discusses future work.

2 Preliminaries

2.1 Supervised Machine Learning Model

A dataset $\mathcal{X}$ contains samples in a d-dimensional feature space. Each sample has a membership in a set of predefined classes called *labels*. Supervised machine learning trains a statistical model by such sample-label pairs to make predictions of labels on unknown samples. Without loss of generality, in this paper we focus on binary models which have only two labels — positive and negative. Formally, a binary model f produces a response y to a query sample $\boldsymbol{x}$ as follows.

$$y = f(\boldsymbol{x}) = \begin{cases} \text{“positive” label} \\ \text{“negative” label} \end{cases}$$

Binary models have been widely adopted in many machine learning applications, particularly in spam filtering, malware detection, and disease diagnosis. Depending on the nature of these applications, the model f can be either linear (e.g., logistic regression) or non-linear (e.g., neural network).

2.2 Model Extraction With Only Labels

In a model extraction attack, a malicious party attempts to replicate a model from the original one by continuously exploiting the prediction API. Technically any queries can constitute such an attack. However, the more queries the more likely this malicious attack will be exposed. As such, in the literature most model extraction attacks fabricate *fine-tuned queries* by differential techniques such as line search [19, 13] and Jacobian augmentation[16]. These queries are carefully selected to capture the information about decision boundary where prediction results vary drastically.

Formally, a model extraction attack selects a set of fine-tuned queries $\mathcal{X}_{diff}$ and obtains their responses $\mathcal{Y}_{diff}$ to train a replica model f'.

$$\mathcal{X}_{diff} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}, \quad \boldsymbol{x} \in \mathbb{R}^d,$$
$$\mathcal{Y}_{diff} = \{y_1, y_2, \ldots, y_n\}, \quad y \in \mathbb{R}^1,$$
$$\exists \boldsymbol{x},\ \boldsymbol{x'} \in \mathcal{X}_{diff},\ dist(\boldsymbol{x}, \boldsymbol{x'}) = \delta\ \wedge\ y \neq y',$$

where $dist(\cdot)$[1] measures the distance between two queries and δ is the unit distance adopted in the differential techniques when searching for boundary, i.e., where two corresponding responses $y \neq y'$.

3 Problem Definition

3.1 Motivation and Threat Model

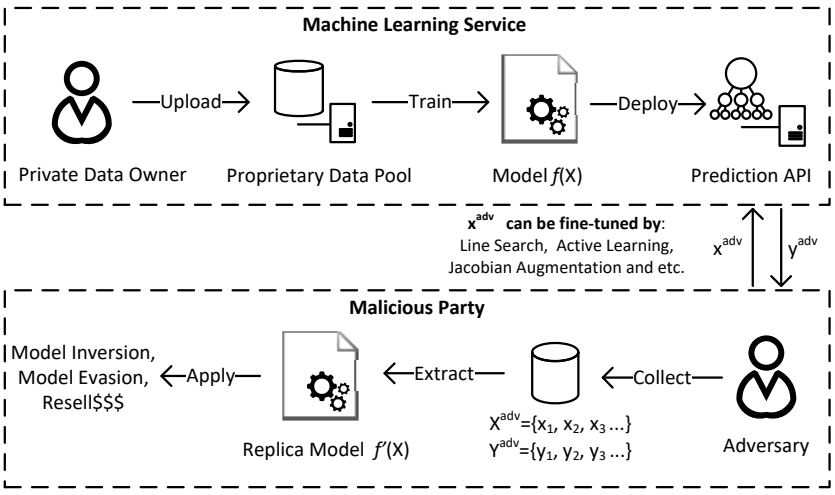


Fig. 1. Motivation and Threat Model

A machine learning service provides a binary prediction result using a proprietary model as shown in Fig. 1. An adversary wants to produce a replica of this model by continuously querying it through the provided prediction API. We assume he can store all queries and their responses, i.e., labels, and the attack is white-box, i.e., he can extract a replicated model using the same model type (e.g., convolutional neural network) and hyperparameters as the original one.[2]

[1] In general, this notation can be any distance metrics (e.g., Manhattan distance, Euclidean distance). The implications of distance metrics to detailed algorithms will be discussed in Section 4.1.

[2] The white-box assumption is based on the fact that state-of-the-art models in specific application domains, such as image classification, are usually public knowledge.

3.2 Boundary-Sensitive Zone

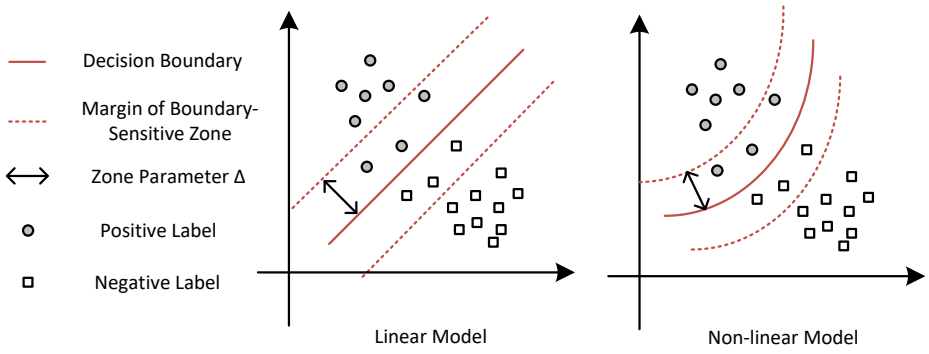


Fig. 2. Illustration of Decision Boundary and Boundary-Sensitive Zone in 2D

Our problem is to protect against model extraction attacks by obfuscating query responses. Before we formally define the security model, we first introduce the notion of *decision boundary* and *boundary-sensitive zone*. For most supervised models, a decision boundary is a critical borderline in the feature space where labels are different on both sides. Fig. 2 illustrates the decision boundaries of a linear and a non-linear model, respectively, in a 2D feature space. In a multi-dimensional feature space, a line boundary becomes a hyperplane, and a curve boundary becomes a hypersurface.

Our key idea is to protect the query responses near the decision boundary against most model extraction attacks. To this end, we introduce the notion of boundary-sensitive zone.

Definition 1. *(Boundary-Sensitive Zone) Given feature space Z, a model f and a parameter Δ chosen by the model owner, all feature vectors adjacent to the decision boundary of f constitute a subspace Z_Δ of Z, where*

$$Z_\Delta = \{\boldsymbol{x} \in \mathbb{R}^d \mid dist(\boldsymbol{x}, f) < \Delta\},$$

where $dist(\cdot)$ measures the distance between a feature vector $\boldsymbol{x}$ and the decision boundary of f. All queries in this zone Z_Δ are considered sensitive and have high risk of revealing the decision boundary of this model.

3.3 Boundary Differential Privacy

All queries in the boundary-sensitive zone need obfuscation, whose objective is to perturb the responses of any two sensitive queries so that they are indistinguishable for the adversary to determine the true decision boundary within this

Nonetheless, our solution can also work against black-box attacks where such knowledge is proprietary.

zone. To this end, we adopt the notion of differential privacy and formally define *boundary differential privacy* as follows.

Definition 2. *(ϵ-Boundary Differential Privacy) A perturbation algorithm $A(\cdot)$ achieves ϵ-boundary differential privacy, if and only if for any two queries $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ in the boundary-sensitive zone Z_{Δ}, the following inequality always holds for the true responses y_1 and y_2 and the perturbed ones $A(y_1)$ and $A(y_2)$.*

$$e^{-\epsilon} \leq \frac{Pr\big[y_1 = y_2 \big| A(y_1), A(y_2)\big]}{Pr\big[y_1 \neq y_2 \big| A(y_1), A(y_2)\big]} \leq e^{\epsilon}$$

The above inequality guarantees that an adversary cannot deduce whether two perturbed responses $A(y_1)$ and $A(y_2)$ originate from the same ($y_1 = y_2$) or different labels ($y_1 \neq y_2$) with high confidence (controlled by ϵ). As such, the adversary cannot use fine-tuned queries, no matter how many they are, to find the decision boundary within the granule of boundary-sensitive zone.

4 Boundary Differentially Private Layer

In this section, we present our solution to protect against model extraction attacks with respect to ϵ-boundary differential privacy (ϵ-BDP) by appending a BDP layer to the model output. According to Definition 2, this layer consists of two major steps — identifying sensitive queries, and perturbing the responses of sensitive queries to satisfy BDP. In what follows, we first introduce a technique to identify sensitive queries with the notion of *corner points*. Then we design a perturbation algorithm called *boundary randomized response* to guarantee ϵ-BDP. Finally, we summarize the procedures of the boundary differentially private layer in Algorithm 1.

4.1 Identifying Sensitive Queries

A query is identified as sensitive if it falls in the boundary-sensitive zone according to Definition 1. However, in practice the decision boundary may not have a closed form (especially for complex models such as neural networks). In this subsection, we propose a method to determine if a query $\boldsymbol{x}_q$ is sensitive without deriving the boundary-sensitive zone. The idea is to test if a ball centered at $\boldsymbol{x}_q$ with radius Δ intersects with the decision boundary[3]. In theory, this is equivalent to finding if there exists a flipping point $\boldsymbol{x}'$ in the ball that has a different label from that of the query point $\boldsymbol{x}_q$. Formally,

Definition 3. *(Query Sensitivity) A query $\boldsymbol{x}_q$ is sensitive, if and only if:*

$$\exists \boldsymbol{x}' \in B(\boldsymbol{x}_q, \Delta), s.t., f(\boldsymbol{x}') \neq f(\boldsymbol{x}_q),$$

where $B(\boldsymbol{x}_q, \Delta) = \{\boldsymbol{x} \in \mathbb{R}^d \ | dist(\boldsymbol{x}, \boldsymbol{x}_q) \leq \Delta\}$ is the ball centered at $\boldsymbol{x}_q$ with radius Δ.

[3] The case of tangency is rarely reached in real life given that the feature space is usually continuous. For simplicity, we mainly consider intersection.

The above definition needs to test infinite number of points in the ball, which is infeasible. Nonetheless, we observe that if the ball is convex and small enough,[4] a sufficient condition of query $\boldsymbol{x}_q$ being sensitive is that at least one of the *corner points* in each dimension of this ball $B(\boldsymbol{x}_q, \Delta)$ is a flipping point. As such, the sensitivity of query $\boldsymbol{x}_q$ can be approximated by testing the labels of $2d$ corner points of $\boldsymbol{x}_q$ without false negatives. Furthermore, if the distance metric is the $L1$ distance (i.e., Manhattan distance), this is also a necessary condition, which means that testing corner points leads to the exact sensivitity. The following theorem proves this.

Theorem 1. *(Flipping Corner Theorem) A sufficient condition of query $\boldsymbol{x}_q$ being sensitive is that,*

$$\exists \;\; \boldsymbol{\Delta}_i \in \Delta \cdot \boldsymbol{I}, \;\; f(\boldsymbol{x}_q \pm \boldsymbol{\Delta}_i) \neq f(\boldsymbol{x}_q),$$

where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{\Delta}_i$ is the projected interval on some dimension i, and $\boldsymbol{x}_q \pm \boldsymbol{\Delta}_i$ denotes the two corner points in dimension i. If the distance metric is the $L1$ distance, this equation is also a necessary condition.

Proof. Let $\boldsymbol{x}_i$ be one of the corner points in dimension i.

- (*Sufficient Condition*) For any $\boldsymbol{x}_i$, the decision boundary must exist between $\boldsymbol{x}_i$ and $\boldsymbol{x}_q$ where $f(\boldsymbol{x}_i) \neq f(\boldsymbol{x}_q)$. It intersects line $\boldsymbol{x}_i\boldsymbol{x}_q$ at point $\boldsymbol{b}_i$. As $\boldsymbol{x}_i$, $\boldsymbol{x}_q$ and $\boldsymbol{b}_i$ are on the same straight line, we have

  $$dist(\boldsymbol{x}_i, \boldsymbol{b}_i) + dist(\boldsymbol{x}_q, \boldsymbol{b}_i) = dist(\boldsymbol{x}_i, \boldsymbol{x}_q) = \Delta.$$

  Since dist($\boldsymbol{x}_q$,f) is the minimum distance between $\boldsymbol{x}_q$ and any point on the decision boundary, we have

  $$dist(\boldsymbol{x}_q, f) \leq dist(\boldsymbol{x}_q, \boldsymbol{b}_i) = \Delta - dist(\boldsymbol{x}_i, \boldsymbol{b}_i) < \Delta.$$

  According to Definition 1, query $\boldsymbol{x}_q$ is sensitive and this proves the sufficient condition.
- (*Necessary Condition for $L1$ Distance*) If $\boldsymbol{x}_q$ is a sensitive query, an $L1$-ball centered at $\boldsymbol{x}_q$ with radius Δ will be given by

  $$B(\boldsymbol{x}_q, \Delta) = \{\boldsymbol{x} \in \mathbb{R}^d \;\big|\; dist_{L1}(\boldsymbol{x}, \boldsymbol{x}_q) \leq \Delta\}. \tag{1}$$

  Let $\boldsymbol{b}_m$ be the point which is the closest to $\boldsymbol{x}_q$ on the decision boundary of f. According to Definition 3, we have

  $$dist_{L1}(\boldsymbol{x}_q, \boldsymbol{b}_m) = dist_{L1}(\boldsymbol{x}_q, f) < \Delta.$$

  Since $\boldsymbol{x}_q$ is sensitive, $\boldsymbol{b}_m$ must be inside this $L1$-ball:

  $$\boldsymbol{b}_m \in B(\boldsymbol{x}_q, \Delta).$$

[4] If Δ is small, the decision boundary near the ball can be treated as a hyperplane.

This means that the decision boundary must intersect the ball at $\boldsymbol{b}_m$. As such, at least one convex vertex of the ball is on a different side of the decision boundary than point $\boldsymbol{x}_q$. Since the convex vertices of an $L1$-ball are exactly those corner points, there exists at least one corner point $\boldsymbol{x}_i$ such that $f(\boldsymbol{x}_i) \neq f(\boldsymbol{x}_q)$. And this proves the necessary condition.

□

4.2 Perturbation Algorithm: Boundary Randomized Response

Randomized response [22] is a privacy-preserving survey technique developed for surveying sensitive questions. A randomized boolean value is given to the answer and provides plausible deniability. As the perturbation algorithm defined in boundary differential privacy has exactly two output choices, we design the following BRR algorithm based on randomized response to satisfy ϵ-BDP.

Definition 4. *(Boundary Randomized Response, BRR) Given query sample $\boldsymbol{x}_q$ and its true response $y_q \in \{0, 1\}$, the boundary randomized response algorithm $A(y_q)$ perturbs y_q by the following:*

$$A(y_q) = \begin{cases} y_q, & w.p. \quad \frac{1}{2} + \frac{\sqrt{e^{2\epsilon}-1}}{2+2e^{\epsilon}} \\ 1 - y_q, & w.p. \quad \frac{1}{2} - \frac{\sqrt{e^{2\epsilon}-1}}{2+2e^{\epsilon}} \end{cases}$$

Theorem 2. *The boundary randomized response algorithm $A(y_q)$ satisfies ϵ-BDP.*

Proof. To satisfy ϵ-BDP, the following inequality must hold according to Definition 2.

$$\frac{Pr[y_1 = y_2 | A(y_1), A(y_2)]}{Pr[y_1 \neq y_2 | A(y_1), A(y_2)]} \leq e^{\epsilon} \tag{2}$$

We assume p is the probability of retaining y_q and $1 - p$ the probability of flipping y_q. According to algorithm A, for any two responses $y_1, y_2 \in \{0, 1\}$, the four possible cases for the above inequality are:

$$\frac{Pr[y_1 = y_2 | A(y_1) = 0, A(y_2) = 0]}{Pr[y_1 \neq y_2 | A(y_1) = 0, A(y_2) = 0]}, \frac{Pr[y_1 = y_2 | A(y_1) = 1, A(y_2) = 1]}{Pr[y_1 \neq y_2 | A(y_1) = 1, A(y_2) = 1]} = \frac{p^2 + (1-p)^2}{2p \cdot (1-p)},$$

$$\frac{Pr[y_1 = y_2 | A(y_1) = 0, A(y_2) = 1]}{Pr[y_1 \neq y_2 | A(y_1) = 0, A(y_2) = 1]}, \frac{Pr[y_1 = y_2 | A(y_1) = 1, A(y_2) = 0]}{Pr[y_1 \neq y_2 | A(y_1) = 1, A(y_2) = 0]} = \frac{2p \cdot (1-p)}{p^2 + (1-p)^2}.$$

Given $0 \leq p \leq 1$, it is easy to prove that the former two cases are always larger than the latter. If we further use equality instead of ineqaulity in Eqn. 2, we can derive the following equation of p:

$$\frac{p^2 + (1-p)^2}{2p \cdot (1-p)} = e^{\epsilon}$$

By solving the above equation, we can derive p as

$$p = \frac{(2+2e^{\epsilon}) \pm \sqrt{(2+2e^{\epsilon})^2 - 4(2+2e^{\epsilon})}}{2(2+2e^{\epsilon})}$$
$$p_1 = \frac{1}{2} + \frac{\sqrt{e^{2\epsilon}-1}}{2+2e^{\epsilon}},\ p_2 = \frac{1}{2} - \frac{\sqrt{e^{2\epsilon}-1}}{2+2e^{\epsilon}} \tag{3}$$

Finally, we need to test the validity of both solutions. Let $u = e^{\epsilon}$, the derivative of p_1 in Eq. 3 with respect to u is:

$$\frac{\partial p}{\partial u} = \frac{(\frac{2}{u-1})(\sqrt{u^2-1})}{(2+2u)^2} \geq 0$$

As such, p_1 is monotonic with respect to u and ϵ. Since $\epsilon \in [0, +\infty]$, the lower and upper bounds of p_1 are obtained when $\epsilon = 0$ and $\epsilon = +\infty$:

$$\lim_{\epsilon \to 0} \Big[\frac{1}{2} + \frac{\sqrt{e^{2\epsilon}-1}}{2+2e^{\epsilon}}\Big] = \frac{1}{2},$$
$$\lim_{\epsilon \to +\infty} \Big[\frac{1}{2} + \frac{\sqrt{e^{2\epsilon}-1}}{2+2e^{\epsilon}}\Big] = \lim_{\epsilon \to +\infty} \Big[\frac{1}{2} + \frac{\sqrt{1-\frac{1}{e^{2\epsilon}}}}{\frac{2}{e^{\epsilon}}+2}\Big] = 1.$$

As such, the derived p_1 in Eqn. 3 is in the range of $[\frac{1}{2}, 1)$ and is thus valid. Similarly, we can prove p_2 is in the range of $(0, \frac{1}{2}]$ and is thus invalid. □

4.3 Summary

Algorithm 1 summarizes the detailed procedures of BDP layer that can be tapped to the output of any machine learning model f. When a new query $\boldsymbol{x}_q$ arrives, if it has already been queried before, the layer directly returns the cached response y'_q to prevent attacker from learning multiple perturbed responses of the same query response, which can lead to a less private BDP. Otherwise, the layer first obtains the real result y_q from model f. Then it determines whether $\boldsymbol{x}_q$ is in the boundary-sensitive zone by checking all corner points. As long as one corner point is as a flipping point, the query is identified as sensitive, and the boundary randomized response algorithm $BRR(\cdot)$ with privacy budget ϵ will be invoked. The layer will thus return the perturbed result y'_q and cache it for future use. Otherwise, if $\boldsymbol{x}_q$ is not sensitive after checking all corner points, the real result y_q will be returned.

5 Experiments

In this section, we evaluate the effectiveness of boundary differentially private layer (BDPL) against model extraction attacks. Specifically, we implement those extraction attacks using fine-tuned queries as in [13, 19] and compare the success rates of these attacks with and without BDPL. All experiments are implemented with Python 3.6 on a desktop computer running Windows 10 with Intel Core i7-7700 3.6GHz CPU and 32G DDR4 RAM.

Algorithm 1 Boundary Differentially Private Layer

Input: Query $\boldsymbol{x}_q \in R^d$
Model f
Boundary-Sensitive Zone Parameter Δ
Boundary Privacy Budget ϵ
Output: Perturbed Response y
Procedure:

```
if x_q is not cached then
    y_q = f(x_q)
    CornerPoints = getCornerPoints(Δ, x_q)
    for x_i in CornerPoints do
        if x_i is a flipping point then
            y'_q = BRR(y_q, ϵ)
            Cache(x_q, y'_q)
            return y'_q
    return y_q
else
    y'_q = getCached(x_q)
    return y'_q
```

5.1 Setup

Datasets and Machine Learning Models. We evaluate two datasets and two models used in the literature [19] — a Botany dataset *Mushrooms* (113 attributes, 8124 records) and a census dataset *Adult* (109 attributes, 48842 records), both of which are obtained from UCI machine learning repository [4]. All categorical items are processed by *one-hot-encoding* [8] and missing values are replaced with the mean value of this attribute. We adopt *min-max normalization* to unify all feature domains into [-1,1]. In the *Mushrooms* dataset, the binary label shows whether a mushroom is poisonous or edible, and in the *Adult* dataset, the binary label shows whether the annual income of an adult exceeds 50K.

We train both a linear model, namely, logistic regression, and a non-linear model, namely, 3-layer neural network, to predict unknown labels on both datasets. Logistic regression is implemented using *cross-entropy* loss with $L2$ regularizer. Neural network is implemented using TensorFlow r1.12 [1]. The hidden layer contains 20 neurons with *tanh* activation. The output layer is implemented with a *sigmoid* function for binary prediction.

Evaluation Metrics. We implement the extraction attack defined in Section 2 using fine-tuned queries generated by the line-search technique. It is a full white-box attack which produces an extracted model f' with the same hyper-parameters and architectures as the original model f. To compare f and f', we adopt *extraction rate* [19, 10] to measure the proportion of matching predictions (i.e., both f and f' predict the same label) in an evaluation query set. Formally,

– **Extraction Rate**. Given an evaluation query set $\mathcal{X}_e$, the extraction rate

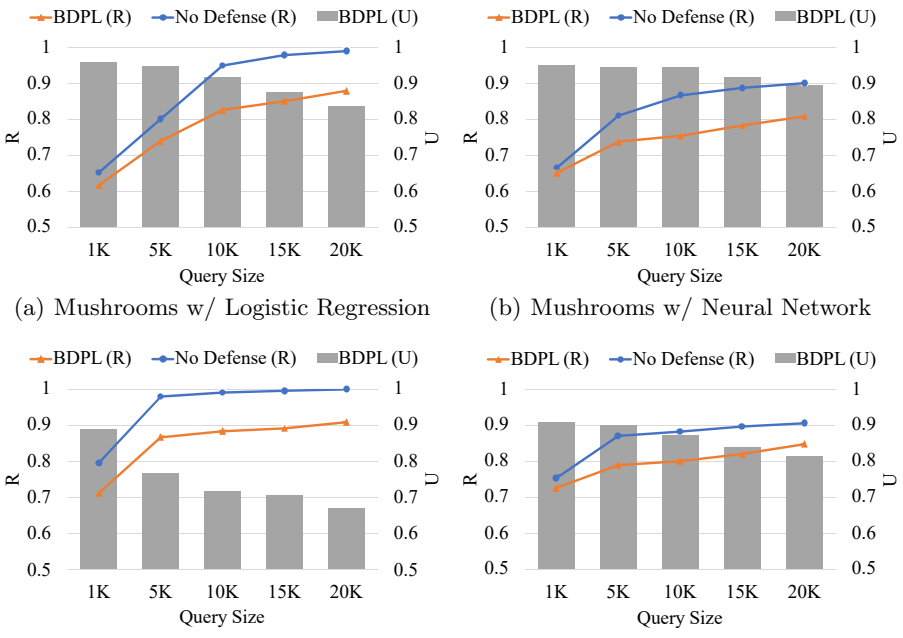
$$R = \frac{1}{|\mathcal{X}_e|} \sum_{\boldsymbol{x}_i \in \mathcal{X}_e} \mathbb{1}(f(\boldsymbol{x}_i) = f'(\boldsymbol{x}_i)),$$

where $\mathbb{1}(\cdot)$ is an indicator function that outputs 1 if the input condition holds and 0 otherwise. The extraction rate essentially measures the similarity of model outputs given the same inputs. In our experiments, the evaluation query set could come from either the dataset or uniformly sampled points in the feature space.

– **Utility**. This second metric measures the proportion of responses that are perturbed (i.e., flipped) by BDPL. It indicates how useful these responses are from a normal user's perspective. Formally, given the entire set of queries $\mathcal{X}_q$ issued by clients, and the set of (perturbed) responses $\mathcal{Y}_q$ from the service provider,

$$U = \frac{1}{|\mathcal{X}_q|} \sum_{\boldsymbol{x}_i \in \mathcal{X}_q, y_i \in \mathcal{Y}_q} \mathbb{1}(f(\boldsymbol{x}_i) = y_i).$$

5.2 Overall Evaluation



(a) Mushrooms w/ Logistic Regression (b) Mushrooms w/ Neural Network

(c) Adult w/ Logistic Regression (d) Adult w/ Neural Network

Fig. 3. Overall Protection Effect by BDPL: Extraction Rate and Utility

To evaluate how well the decision boundary can be protected by BDPL, we launch extraction attacks on 4 model/dataset combinations and plot the extraction rate R of sensitive queries in Fig. 3 as the number of queries increases. For BDPL, we set $\Delta = 1/8$, and $\epsilon = 0.01$. In all combinations, except for the initial extraction stage (query size less than $5K$), BDPL exhibits a significant protection effect (up to 12% drop on R) compared with no defense. Furthermore, even though the two models are very diverse (the parameters of the neural network are 20 times more than that of the logistic regression), BDPL shows consistent protection effect by a similar drop of R.

The secondary axis of Fig. 3 also plots the utility of BDPL. We observe that the utility saturates at over 80% after $20K$ queries in all combinations except for *Adult w/ Logistic Regression*. This model has the fewest parameters and features, so BDPL has to perturb more sensitive queries to retain the same BDP level as the others. The impact on utility by Δ and ϵ will be shown in Section 5.4.

5.3 BDPL vs. Uniform Perturbation

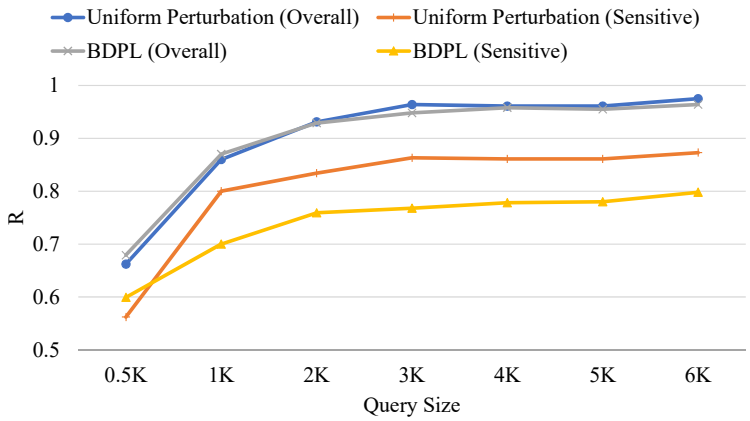


Fig. 4. BDPL vs. Uniform Perturbation

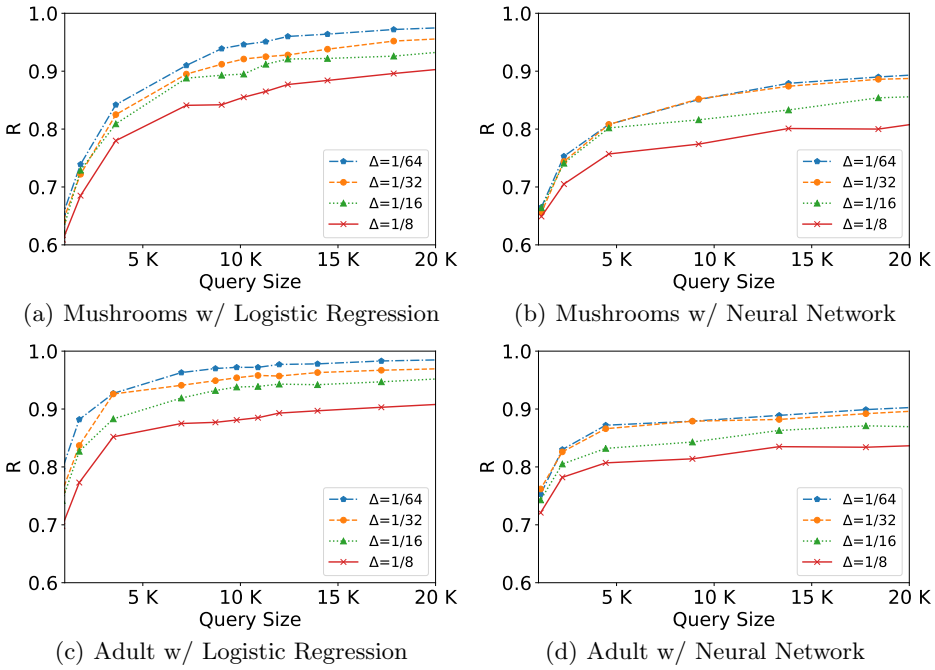
In this experiment, we compare BDPL with a uniform perturbation mechanism that randomly flips the response label by a certain probability, whether the query is sensitive or not. To have a fair comparison, we use trial-and-error[5] to find this probability so that the overall extraction rates of both mechanisms are almost the same. We then plot the extraction rates of both mechanisms for sensitive queries in Figure 4. Due to space limitation, we only show the results for *Mushrooms with Logistic Regression* with $\Delta = 1/8$ and $\epsilon = 0.01$. We observe

[5] To do this, we start with 1 random flip out of all responses and measure its overall extraction rate. We then repeatedly increment this number by 1 until the overall extraction rate is very close to that of BDPL.

that BDPL outperforms uniform perturbation by 5%-7%, which is very significant as this leads to an increase of misclassification rate by 30%-50%. As such, we can conclude that BDPL is very effective in protecting the decision boundary by differentiating sensitive queries from non-sensitive ones, and therefore it retains high utility for query samples that are faraway from the boundary.

5.4 Impact of ϵ and Δ

In this subsection, we evaluate BDPL performance with respect to various values of boundary-sensitive zone parameter Δ and privacy budget ϵ. In each experiment, we fix the value of ϵ (resp. Δ) and vary Δ (resp. ϵ) for all 4 model/dataset combinations. Δ ranges between 1/64 and 1/8 while ϵ ranges between 0.01 and 0.64. Fig. 5 and Fig. 6 show the evaluation results on varying Δ and ϵ respectively.

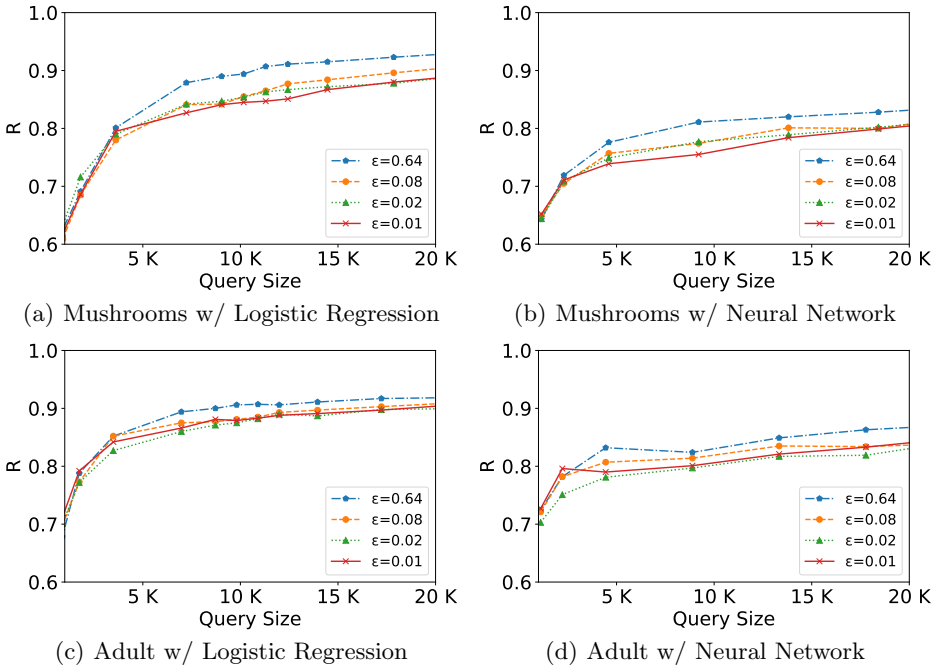


(a) Mushrooms w/ Logistic Regression (b) Mushrooms w/ Neural Network

(c) Adult w/ Logistic Regression (d) Adult w/ Neural Network

Fig. 5. Impact of Varying Δ

Impact on Extraction Rate. When Δ increases from 1/64 to 1/8, the extraction rate is significantly reduced in both logistic regression (up to 12% drop) and neural network (up to 10% drop). Nonetheless, for neural networks, the extract rate does not change much when Δ increases from 1/64 to 1/32, which indicates that if the boundary-sensitive zone is too small, BDPL may not provide effective protection, especially when the decision boundary is non-linear.



(a) Mushrooms w/ Logistic Regression (b) Mushrooms w/ Neural Network

(c) Adult w/ Logistic Regression (d) Adult w/ Neural Network

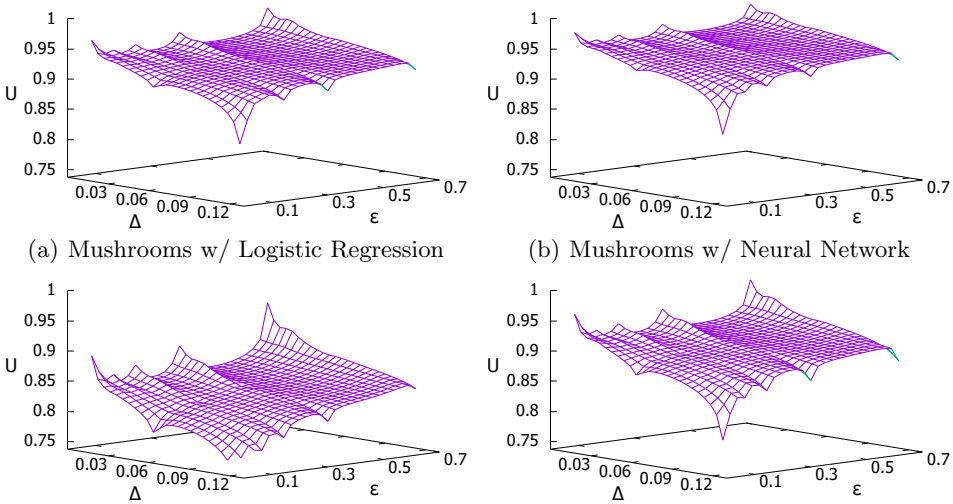
Fig. 6. Impact of Varying ϵ

As for privacy budget ϵ, its impact is not as significant as Δ. We only observe up to 4% drop of extraction rate when ϵ decreases from 0.64 to 0.01 for all 4 model/dataset combinations.

Last but not the least, the extraction rates under all these settings saturate as the query size increases. In most cases, they start to saturate before $5K$ queries, and even in the worst case, they saturate at $15K$ or $20K$. This indicates that BDPL imposes a theoretical upper bound on the extraction rate no matter how many queries are issued.

Impact on Utility. In Fig. 7, we plot the final utility after $20K$ queries for all Δ and ϵ combinations. Except for *Adult w/ Logistic Regression*, all utilities are higher than 80% and most of them are above 90%, which means that BDPL does not severely sacrifice the accuracy of a machine learning service. As expected, the utility reaches peak when $\Delta = 1/64$ (smallest zone size) and $\epsilon = 0.64$ (least probability of perturbation). Furthermore, as is coincided with the extraction rate, the utility is more sensitive to Δ than to ϵ. For example, an increase of Δ from 0.01 to 0.1 leads to a drop of utility by 10%, whereas a decrease of ϵ from 0.1 to 0.01 leads to only 5% drop.

To conclude, BDPL permanently protects decision boundary of both linear and non-linear models with moderate utility loss. The changes of Δ and ϵ (particularly the former) have some modest impact on the extraction rate and utility.



(a) Mushrooms w/ Logistic Regression (b) Mushrooms w/ Neural Network

(c) Adult w/ Logistic Regression (d) Adult w/ Neural Network

Fig. 7. Utility vs. Δ and ϵ

6 Related Works

There are three streams of related works, namely, machine learning model extraction, defense, and differential privacy.

Model Extraction. Machine-learning-as-a-service(MLaaS) has furnished model extraction attacks through the rich information available from prediction API. Tramer *et al.* [19] proposed extraction methods that leveraged the confidence information in the API and managed to extract the full set of model parameters using equation-solving. Papernot [16] *et al.* introduced a Jacobian-based data augmentation technique to create synthetic queries and to train a substitute DNN. Similarly, Juuti *et al.* [9] leveraged both optimal hyperparameters and the Jacobian to extract models. Oh *et al.* [14] developed a model-of-model to infer internal information of a neural network such as layer type and kernel sizes. Orekondy *et al.* [15] proposed a knockoff model to steal the functionality of an image classification model with black-box API access. Besides extracting internal parameters, Wang *et al.* [21] also extracted the hyperparamters of a fully trained model by utilizing the zero gradient technique.

Model extraction without confidence is similar to *learning with membership query* [3, 20], which learns a concept through querying membership on an oracle. This technique has been exploited by Lowd *et al.* to extract binary classifiers [13]. They used line search to produce optimized queries for linear model extraction. This technique was extended by Tramer *et al.* [19] to non-linear models such as a polynomial kernel support vector machine. They adopted adaptive techniques such as active learning to synthesize fine-tuned queries and to approximate the decision boundary of a model.

Model Extraction Defense. Confidence rounding and ensemble model were shown effective against equation-solving extractions in [19]. Lee *et al.* [12] proposed perturbations using the mechanism of reverse sigmoid to inject deceptive noises to output confidence, which preserved the validity of top and bottom rank labels. Kesarwani *et al.* [10] monitored user-server streams to evaluate the threat level of model extraction with two strategies based on entropy and compact model summaries. The former derived information gain with a decision tree while the latter measured feature coverage of the input space partitioned by source model, both of which were highly correlated to extraction level. Juuti *et al.* [9] adopted a different approach to monitor consecutive queries based on the uniqueness of extraction behavior. A warning would be generated when queries deviated from a benign distribution due to malicious probing. Quiring *et al.* [17] adopted the notion of closeness-to-the-boundary in digital watermarking and applied it to protect against extraction attacks on decision trees. The defense strategy was devised from protection of watermark detector and it monitored the number of queries that fell into security margin.

Differential Privacy. Differential privacy (DP) was first proposed by Dwork [6] to guarantee the privacy of a centralized dataset with standardized mathematical notation. Duchi *et al.* [5] extended this notation to local differential privacy (LDP) for distributed data sources. Randomized response proposed by Warner *et al.* [22] is the baseline perturbation algorithm for LDP, which protects binary answers of individuals. Although differential privacy has not been used in model extraction and defense, it has been applied in several adversarial machine learning tasks. For example, Abadi *et al.* [2] introduced differentially private stochastic gradient descent to deep learning, which can preserve private information of the training set. Lee *et al.* [11] further improved its effectiveness using an adaptive privacy budget. Their approaches are shown effective against model inversion attack [7] or membership inference attack[18].

7 Conclusion and Future Work

In this paper, we propose boundary differentially private layer to defend binary machine learning models against extraction attacks by obfuscating the query responses near the decision boundary. This layer guarantees boundary differential privacy (ϵ-BDP) in a user-specified boundary-sensitive zone. To identify sensitive queries that fall in this zone, we develop an efficient approach that use corner points as indicators. We design boundary randomized response as the perturbation algorithm to obfuscate query responses. This algorithm is proved to satisfy ϵ-BDP. Through extensive experimental results, we demonstrate the effectiveness and flexibility of our defense layer on protecting decision boundary while retaining high utility of the machine learning service.

For future work, we plan to generalize our defense layer to a multi-class model and adapt the perturbation algorithm to it. We also plan to extend our defense layer to protect against other machine learning attacks such as model evasion and inversion.

Acknowledgement. This work was supported by National Natural Science Foundation of China (Grant No: 61572413, U1636205, 91646203, 61532010, 91846 204, and 61532016), the Research Grants Council, Hong Kong SAR, China (Grant No: 15238116, 15222118 and C1008-16G), and a research grant from Huawei Technologies.

References

1. Abadi, M., Agarwal, A., Barham, P., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security. pp. 308–318 (2016)
3. Angluin, D.: Queries and concept learning. Machine Learning **2**, 319–342 (1987)
4. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
5. Duchi, J.C., Jordan, M.I., Wainwright, M.J.: Local privacy and statistical minimax rates. In: IEEE Symposium on Foundations of Computer Science. pp. 429–438 (2013)
6. Dwork, C.: Differential privacy. In: Automata, Languages and Programming, 33rd International Colloquium. pp. 1–12 (2006)
7. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security. pp. 1322–1333 (2015)
8. Harris, D.M., Harris, S.L.: Digital design and computer architecture (2007)
9. Juuti, M., Szyller, S., Dmitrenko, A., Marchal, S., Asokan, N.: Prada: Protecting against dnn model stealing attacks. CoRR **abs/1805.02628** (2018)
10. Kesarwani, M., Mukhoty, B., Arya, V., Mehta, S.: Model extraction warning in mlaas paradigm. In: Annual Computer Security Applications Conference (2018)
11. Lee, J., Kifer, D.: Concentrated differentially private gradient descent with adaptive per-iteration privacy budget. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2018)
12. Lee, T., Edwards, B., Molloy, I., Su, D.: Defending against model stealing attacks using deceptive perturbations. CoRR **abs/1806.00054** (2018)
13. Lowd, D., Meek, C.: Adversarial learning. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. pp. 641–647. KDD '05, ACM (2005)
14. Oh, S.J., Augustin, M., Schiele, B., Fritz, M.: Towards reverse-engineering black-box neural networks. In: International Conference on Learning Representations (2018)
15. Orekondy, T., Schiele, B., Fritz, M.: Knockoff nets: Stealing functionality of black-box models. CoRR **abs/1812.02766** (2018)
16. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 506–519 (2017)
17. Quiring, E., Arp, D., Rieck, K.: Forgotten siblings: Unifying attacks on machine learning and digital watermarking. IEEE European Symposium on Security and Privacy (EuroS&P) pp. 488–502 (2018)

18. Shokri, R., Stronati, M., Shmatikov, V.: Membership inference attacks against machine learning models. IEEE Symposium on Security and Privacy pp. 3–18 (2017)
19. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Proceedings of the 25th USENIX Conference on Security Symposium. pp. 601–618 (2016)
20. Valiant, L.G.: A theory of the learnable. In: ACM Symposium on Theory of Computing (1984)
21. Wang, B., Gong, N.Z.: Stealing hyperparameters in machine learning. In: IEEE Symposium on Security and Privacy. pp. 36–52 (2018)
22. Warner, S.L.: Randomized response: A survey technique for eliminating evasive answer bias. Journal of the American Statistical Association **60(309)**, 63–6 (1965)
23. Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers: A case study on PDF malware classifiers. In: Annual Network and Distributed System Security Symposium (2016)