Sniffer: A Novel Model Type Detection System against Machine-Learning-as-a-Service Platforms

Zhuo Ma
Xidian University
mazhuo@mail.xidian.edu.cn

Yilong Yang
Xidian University
yangyilong@stu.xidian.edu.cn

Bin Xiao
Chongqing University of Posts and Telecommunications
xiaobin@cqupt.edu.cn

Yang Liu
Xidian University
bcds2018@foxmail.com

Xinjing Liu
Xidian University
liuxinjing_j@163.com

Zhuoran Ma
Xidian University
emmazhr@163.com

Tong Yang
Peking University
yangtongemail@gmail.com

ABSTRACT

Recent works explore several attacks against Machine-Learning-as-a-Service (MLaaS) platforms (e.g., the model stealing attack), allegedly posing potential real-world threats beyond viability in laboratories. However, hampered by *model-type-sensitive*, most of the attacks can hardly break mainstream real-world MLaaS platforms. That is, many MLaaS attacks are designed against only one certain type of model, such as tree models or neural networks. As the black-box MLaaS interface hides model type info, the attacker cannot choose a proper attack method with confidence, limiting the attack performance. In this paper, we demonstrate a system, named Sniffer, that is capable of making model-type-sensitive attacks "great again" in real-world applications. Specifically, Sniffer consists of four components: Generator, Querier, Probe, and Arsenal. The first two components work for preparing attack samples. Probe, as the most characteristic component in Sniffer, implements a series of self-designed algorithms to determine the type of models hidden behind the black-box MLaaS interfaces. With model type info unraveled, an optimum method can be selected from Arsenal (containing multiple attack methods) to accomplish its attack. Our demonstration shows how the audience can interact with Sniffer in a web-based interface against five mainstream MLaaS platforms.

PVLDB Reference Format:
Zhuo Ma, Yilong Yang, Bin Xiao, Yang Liu, Xinjing Liu, Zhuoran Ma, and Tong Yang. Sniffer: A Novel Model Type Detection System against Machine-Learning-as-a-Service Platforms. PVLDB, 16(12): 3942 - 3945, 2023.
doi:10.14778/3611540.3611591

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Echotoken/Sniffer.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611591

1 INTRODUCTION

Machine-Learning-as-a-Service (MLaaS) is one of the most popular forms of machine learning inference service [4, 5]. Through user-friendly pay-for-query interfaces, MLaaS can make the service of top ML scientists accessible to any user. The great success of MLaaS attracts interest of many security researchers. A series of novel attacks are explored against the models hidden behind the black-box MLaaS interfaces, such as model extraction attack (MEA) [3] and membership inference attack (MIA) [1, 9].

Commonly, to fit real-world attack scenarios, these MLaaS attacks are claimed to be executable with "no" prior knowledge of the victim model. However, the fact is that most of the attacks are *model-type-sensitive*, and thus, require model type info to ensure attack effectiveness. For example, in [2], Varun *et al.* proposed three MEA methods, each of which can only be used for extracting the parameters of one certain type of model (like linear SVMs, kernel SVMs, or tree models).

With regards to this, Model-type-sensitive evidently limits the universality of most MLaaS attacks. For instance, according to our observation, most mainstream real-world platforms, such as Amazon SageMaker[1] and Google Cloud[2], do not provide model type info in the response packages of their interfaces. As the result, the model-type-sensitive attacks suffer from severe performance degradation including failure.

Contributions. In this paper, we demonstrate a model type detection (MTD) system, named Sniffer, that is capable of making model-type-sensitive attacks "great again" in real-world applications. Specifically, Sniffer consists of four components, including an attack samples generator (Generator), an automatic MLaaS querier (Querier), a set of detection probes (Probe), and an "arsenal" equipped with varying MLaaS attacks (Arsenal). Generator maintains an attack pool containing numerous synthesized or publicly available (unlabeled) data points for launching MTD attacks. Querier interacts with the victim interface to obtain the prediction scores of the data in Generator. Probe, as the main module of Sniffer, contains multiple probes designed for different model types, each of which can output whether or not the victim model belongs to a specific model type. Based on the result of Probe, an appropriate method can be selected to accomplish the black-box MLaaS attack.

[1] https://aws.amazon.com/sagemaker/
[2] https://cloud.google.com/vertex-ai

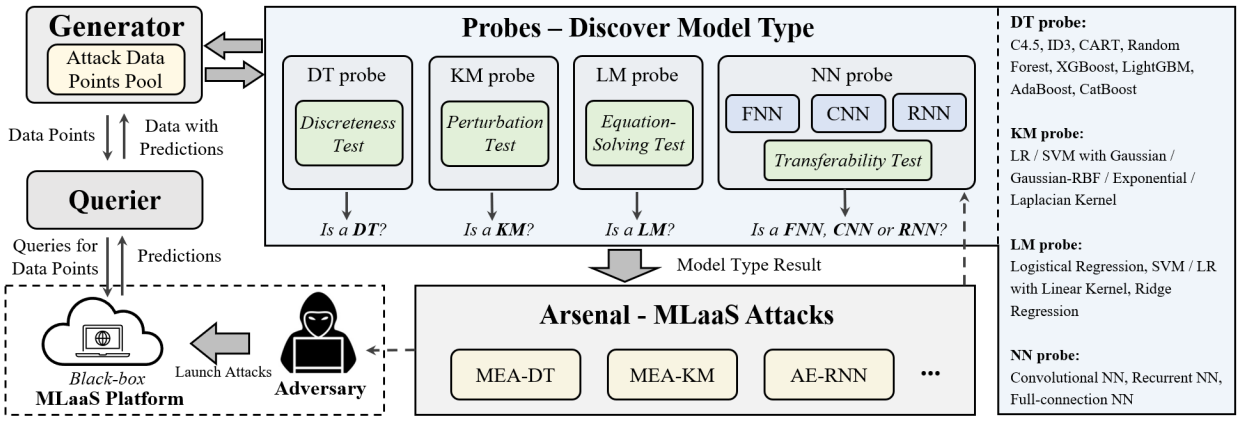

Figure 1: The workflow of Sniffer: 1) Generator synthesizes unlabeled attack samples. 2) Querier interacts with the MLaaS platform to obtain predictions of the attack samples and put them in the attack pool. 3) Each probe outputs whether the victim model belongs to a certain model type. 4) In Arsenal, according to the detection result, an appropriate MLaaS attack method is selected.

For a first attempt, Sniffer mainly focuses on four mainstream model types, sufficiently covering the scope of most state-of-the-art MLaaS attacks. Four probes, namely Decision Trees probe (DTp), Linear Models probe (LMp), Models with the non-linear Kernel probe (KMp), and Neural Networks probe (NNp), are included to complete the MTD attack. NNp can also further predict the possible network architecture, including convolutional NN (CNN), fully-connected NN (FNN) or recurrent NN (RNN).

In the demonstration, Sniffer provides an users-friendly graphical user interface based on the browser, requiring only two main pieces of info: a prediction API of the victim model and the description of valid input space. We illustrate the meaning of each interaction and the attack workflow of Sniffer in detail for the audience. Moreover, Sniffer provides the following functions: 1) MLaaS platform attack and local simulation 2) retention of processing log, 3) visualization of results, and 4) review of detection history.

2 MODEL TYPE DETECTION ATTACK

Our research mainly focuses on the typical MLaaS scenario. That is, a server trains a model f and provides it as a prediction service for users. From the interface, the user can only gain info about its own input and the corresponding prediction score output by f. MTD attack is the process where an adversary $\mathcal{A}$ tries to answer what the type of model f is, by interacting with the interface within the maximum query budget q. The following outlines the attack process of the MTD attack.

Denote a set of possible model types as C, which contains the alternative model types in real-world applications. Assume that an MLaaS platform obtains a well-trained model f_c and provides a pay-per-query interface for users, where $c \in C$ is the type of f. During the attack process, $\mathcal{A}$ is capable of crafting specific data points and obtaining at most q predictions of them by querying f_c. The querying results are recorded as an attack pool $(\mathcal{X}, \mathcal{Y})$ where $y_i = f_c(x_i) \in \mathcal{Y}$, $x_i \in \mathcal{X}$ and $1 \leq i \leq q$. By analyzing $(\mathcal{X}, \mathcal{Y})$, $\mathcal{A}$ outputs $c' \in C$. If c' is equal to c, we say that $\mathcal{A}$ launches a successful attack, otherwise, the attack fails.

3 SYSTEM OVERVIEW

As shown in Figure 1, Sniffer is composed of four components: Generator, Querier, Probe and Arsenal.

Generator. Generator is responsible for generating data samples for usage in Probe as the first step of the attack. To reduce query budget, the adversary carefully crafts the data points to ensure the maximum info gain for each query. Normally speaking, the queries of MLaaS attacks can be crafted with the data collected from public data resources, or directly synthesized according to specific rules [2, 6]. Especially, Sniffer adopts hybrid rules to cover the requirements of multiple probes against different model types. For an overall view, Generator in Sniffer is designed to generate data points with minimal query budgets and comprehensive rules and maintain an attack data points pool.

Querier. This component is mainly used to assist Generator to label the generated data by querying the MLaaS interface of the victim model. While implementing Sniffer, we notice that MLaaS platforms set different access rules for machine learning inference queries, such as limiting the maximum query number per minute and different data transmission formats. In these scenarios, Querier is required to automatically launch queries according to the rules to circumvent the defense strategy. Denote a data point in the attack pool as $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ represents the crafted data features and $\mathbf{y}$ is the prediction from the victim model in MLaaS platform.

Probe. Probe is the core component of Sniffer to discover the model types. In our implementation, four probes against six popular types of models are considered, including the LM probe, DT probe, KM probe, and NN probe. NNp can also further predict the possible network architecture, and answer whether the victim model type is CNN, FNN or RNN. Each probe can independently answer the confidence that the victim model belongs to a specific type of model. Next, we discuss each probe in detail:

Probe for Decision Trees (or Forest). For models with a decision-tree-type structure, the prediction scores of the victim model are the values of their leaf nodes. Thus, a unique property of such models is the output space being a finite discrete set due to a finite number of

leaf nodes. To simplify, the decision trees (DTs) can be formulated as a key-value table, where a continuous segment of keys (features of data points) is mapped to a definite value (victim model prediction). When the features of data points are within this continuous range, the output value is constant. Utilizing this observation, the DT probe in Sniffer uses *discreteness test* to distinguish DTs from other models. Specifically, a data point $(\mathbf{x}, \mathbf{y})$ is randomly selected in the attack pool, where $\mathbf{x}_i$ is i-th feature of $\mathbf{x}$. Given a fixed interval s, the prediction sample returned by Querier is $(\mathbf{x}_i + s, \mathbf{y}')$. If $\mathbf{y}'$ is the same as $\mathbf{y}$, DTp considers this feature to have discrete properties within this interval. Repeating the above discreteness test beyond a given threshold T, if the discreteness always holds, the victim model is considered to be DTs.

Probe for Non-Linear Kernel Models. By adding nonlinear kernels transforming the data points to a higher-dimensional space, e.g., Gaussian kernel, KMs are able to solve nonlinear problems. Basically, KMs can be represented as $f_K = \omega \cdot K(\mathbf{x}) + b$, where $K(\cdot)$ denotes the nonlinear kernel function. Taking the Gaussian Kernel as an example, when the absolute value of one feature $\mathbf{x}_i$ exceeds a given *huge number* h, $K(\mathbf{x})$ and its change rate both decay rapidly. We observe that they change very little even when the random perturbation p is given again. In Sniffer. KMp modifies data sample $(\mathbf{x}_i + h, \mathbf{y})$ in the attack pool into $(\mathbf{x}_i + h + p, \mathbf{y}')$. If $\mathbf{y} \approx \mathbf{y}'$, Sniffer considers the victim model as KMs.

Probe for Linear Models. Linear models, e.g., logistic regression, with classification classes C can be formulated as $f_L = \omega * x + b$, where $\omega, \mathbf{x} \in \mathbb{R}^{C \cdot n}$, $b \in \mathbb{R}^C$, n indicates the feature number. Inspired by [7], Sniffer can get the parameters of the model by solving the equations about $C \cdot n + C$ input-output pairs, guaranteeing the prediction error is less than the given bound ϵ. Literally speaking, Sniffer categorizes the victim model as LMs if the simultaneous equations are resolvable. Significantly, the predictions of real-world MLaaS platforms are usually masked by the Softmax function, which can be unmasked by the inverse algorithm in Querier, as discussed in [7].

Probe for Neural Networks. To determine the specific type of NN, Sniffer introduces the idea of transferability-based adversarial example (AE) attacks. In brief, given a fixed attack pool and an AE algorithm, the more similar the local model structure is to the victim model, the higher the attack success rate (ASR) of the AE attack is. Specifically, NNp first generates three types of local substitution NN models: FNN, CNN, RNN, and locally maintains three different attack data point pools through an AE algorithm, e,g., L-BFGS [8] in Sniffer. Then, the ASRs of the three attack pools against the victim model are computed respectively. Finally, NNp categorizes the victim model to the type with the largest ASR.

Although the probes are independent of each other, the detection order is principled: priority is given to judging a particular model with the characteristic of exclusivity. Exclusivity here means that each probe has a definite result and will not affect the judgment of subsequent probes[3]. Based on the principle, the order of running the probes in Sniffer is DTp → KMp → LMp → NNp. The order is designed to make full use of the exclusivity of DTs and KMs to prevent the misjudgment of LMs approximating DTs and KMs.

[3]To prevent misclassification, this guideline should be followed if the audience wants to add probes for unknown models.

Additionally, Sniffer supports outputting the confidence level of each probe. Specifically, suppose that the total number of data points for a given test is Σ. The confidence output by each probe is calculated by $\frac{\sigma}{\Sigma} \times 100\%$, where σ is the number of data points that pass the test algorithm.

Arsenal. As the name manifests, "Arsenal" contains multiple attack methods for the adversary to attack the victim models with various model types in MLaaS, such as MEA-DT [2]. The adversary can choose appropriate MLaaS attacks based on the model type confidence provided by Probe. Also, Arsenal takes the responsibility for the NN probe to pick appropriate MLaaS adversarial example attacks for transferability testing.

4 DEMONSTRATION

In this demonstration, we describe the web-based user interface (UI) of Sniffer and how the audience can use this pipeline, as shown in Figure 2. Sniffer can automatically detect the type of the victim model deployed on the MLaaS platforms, only requiring the MLaaS query interface and the feature info of valid inputs. The following presents the audience with the meaning of each interaction in turn.

Figure 2-a): ① The audience gives this attack a name. ② Sniffer has five pre-built options corresponding to public MLaaS cloud platforms, namely Google Cloud, Amazon Sagemaker, Aliyun[4], Tencent Cloud [5], Huawei Cloud[6]. Their access rules are integrated into Querier and entirely transparent to the audience. ③ The audience needs to specify the prediction API to invoke the MLaaS service. The prediction API represents the query interface of the victim model obtained from the ML inference serving provider. ④ Also, Sniffer allows the audience to choose which model types to probe. In particular, assuming the existence of some foreground knowledge that a particular model is not suitable to handle the current target task, the audience can skip the corresponding probing. ⑤ Next, in order to interact with the MLaaS interface correctly, the audience needs to provide Sniffer with the number of features and their approximate value ranges. ⑥ By default, Sniffer builds in eight common datasets, which are Iris, Adult, MNIST, CIFAR10, Cancer, Digits, Fashion-MNIST and Wine. If the input of the target MLaaS platform shares common features with datasets above, the audience can select it directly. ⑦ Also, Sniffer provides the option of *Manual Input*, which allows the audience to input the description of valid samples. ⑧ For example, if data samples in the Iris dataset are valid to the victim model, the audience inputs "4" features and their respective approximate feature ranges are "(5, 7)(2, 4)(1, 6)(1, 2.5)". We can reasonably expect to know the approximate ranges of features in the real-world MLaaS scenario, where users are sufficiently informed of how to provide valid inputs for the usage of the service. Especially, if all ranges of the features are identical, the audience can add the keyword "all" in the front of the range, i.e., "all(-1,1)". After filling in the above info, the audience clicks "ATTACK" button to launch the MTD attack.

Figure 2-b) shows the processing log during the whole MTD attack, containing key adjustment parameters, attack samples, the schedule of detection, etc. Finally, as shown in **Figure 2-c)**, Sniffer

[4]https://ai.aliyun.com/

[5]https://cloud.tencent.com/product/ai-class

[6]https://www.huaweicloud.com/product/modelarts.html

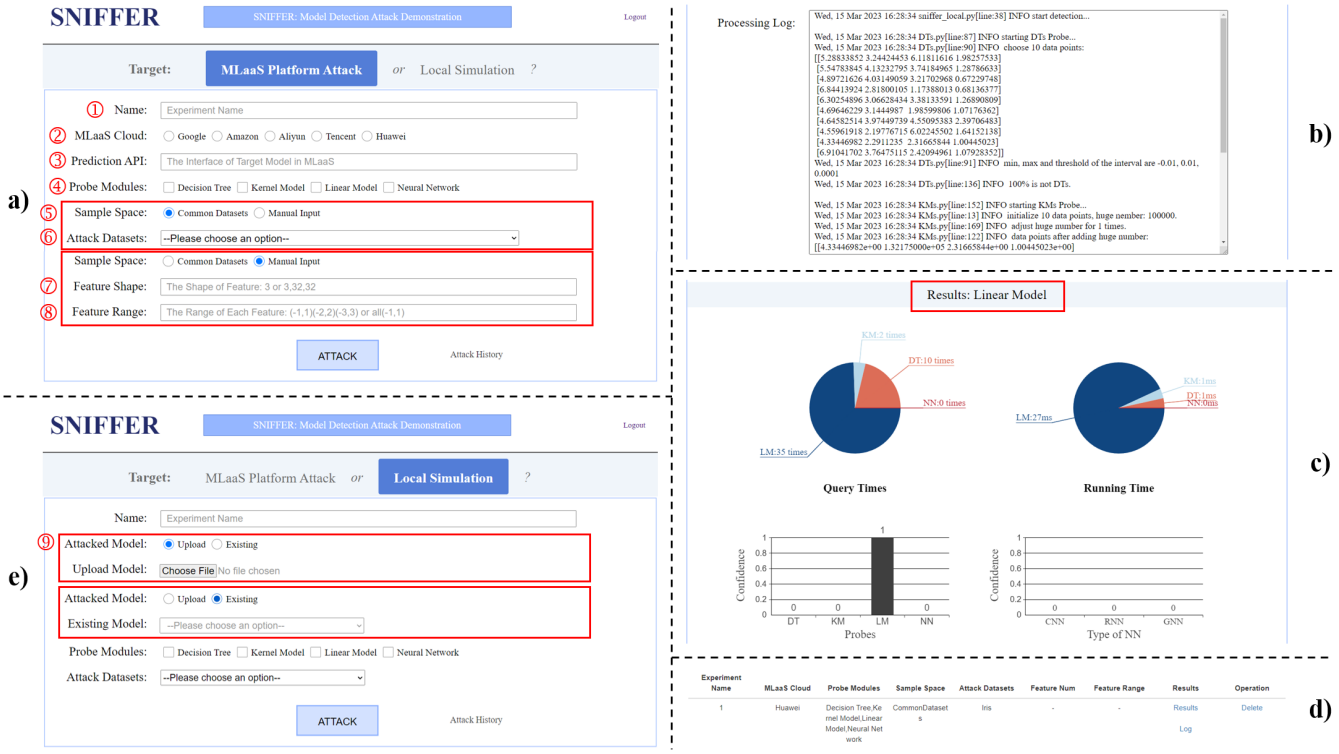

Figure 2: The visual web-based interface of Sniffer: the interactions of a) MLaaS Platform Attack and b) Local Simulation; c) processing log; d) model type detection results; e) attack histories.

will give a final model type detection result (i.e., "Results: Linear Model") to launch a real black-box MLaaS attack. The specific confidence output by each probe is plotted in the bar chart. Besides, the query times between different probes with MLaaS platforms and the running times during the whole attack are presented in two pie charts, respectively. **Figure 2-d)** saves the histories of attack (e.g., results and the processing log) for reviewing.

Figure 2-e): In addition to attacking MLaaS platforms, Sniffer also provides a local testbed by introducing *Local Simulation*, e.g., testing attack baseline. ⑨ Different from *MLaaS Platform Attack*, *Local Simulation* allows the audiences to choose our built-in models or upload the model (.pkl) trained by themselves to test the algorithms. To enable users to experience the MTD attack pipeline locally, we provide six types of models in "Existing" options, covering all model types that Sniffer supports. When the audience chooses to upload the model, it is necessary to ensure that its input and output are both Tensors. Moreover, the user interaction, experiment result display, and processing log are basically identical to those of the *MLaaS platform attack* function.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (U21A20464, 62261160651), the Natural Science Basic Research Program of Shaanxi (No. 2021JC-22), Key Research and Development Program of Shaanxi (No.2022GY-029), the China 111 Project (No.B16037).

REFERENCES

[1] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. 2022. Membership Inference Attacks From First Principles. In *2022 IEEE Symposium on Security and Privacy (SP)*. 1897–1914. https://doi.org/10.1109/SP46214.2022.9833649

[2] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring Connections between Active Learning and Model Extraction. In *Proceedings of the 29th USENIX Conference on Security Symposium (SEC'20)*. USENIX Association, USA, Article 74, 18 pages.

[3] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Xiaofei Xie, and Yang Liu. 2021. Stealing Deep Reinforcement Learning Models for Fun and Profit. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*. Association for Computing Machinery, 307–319.

[4] Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, and Matei Zaharia. 2020. A Demonstration of Willump: A Statistically-Aware End-to-End Optimizer for Machine Learning Inference. *Proc. VLDB Endow.* 13, 12 (aug 2020), 2833–2836.

[5] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-End Optimization of Machine Learning Prediction Queries. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*. Association for Computing Machinery, 587–601.

[6] Sunandini Sanyal, Sravanti Addepalli, and R Venkatesh Babu. 2022. Towards data-free model stealing in a hard label setting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15284–15293.

[7] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 601–618.

[8] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. 2020. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples.. In *NDSS*.

[9] Xiaoyong Yuan and Lan Zhang. 2022. Membership Inference Attacks and Defenses in Neural Network Pruning. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 4561–4578.