Stealing Malware Classifiers and AVs at Low False Positive Conditions

Maria Rigaki  
rigakmar@fel.cvut.cz  
Czech Technical University in Prague  
Prague, Czech Republic

Sebastian Garcia  
garciseb@fel.cvut.cz  
Czech Technical University in Prague  
Prague, Czech Republic

ABSTRACT

Model stealing attacks have been successfully used in many machine learning domains, but there is little understanding of how these attacks work in the malware detection domain. Malware detection and, in general, security domains have very strong requirements of low false positive rates (FPR). However, these requirements are not the primary focus of the existing model stealing literature.

Stealing attacks create surrogate models that perform similarly to a target model using a limited amount of queries to the target. The first stage of this study is the evaluation of active learning model stealing attacks against publicly available stand-alone machine learning malware classifiers and antivirus products (AVs). We propose a new neural network architecture for surrogate models that outperforms the existing state of the art on low FPR conditions. The surrogates were evaluated on their agreement with the targeted models. Good surrogates of the stand-alone classifiers were created with up to 99% agreement with the target models, using less than 4% of the original training dataset size. Good AV surrogates were also possible to train, but with a lower agreement.

The second stage used the best surrogates as well as the target models to generate adversarial malware using the MAB framework [29] to test stand-alone models and AVs (offline and online). Results showed that surrogate models could generate adversarial samples that evade the targets but are less successful than the targets themselves. Using surrogates, however, is a necessity for attackers, given that attacks against AVs are extremely time-consuming and easily detected when the AVs are connected to the internet.

CCS CONCEPTS

• Security and privacy → Malware and its mitigation; Systems security; Computing methodologies → Learning settings; Neural networks.

KEYWORDS

model extraction, machine learning, malware evasion

1 INTRODUCTION

In recent years there has been a strong interest in model stealing attacks to understand how models can be extracted and how model information can be leaked. These attacks were successful up to some extent, implying that machine learning models need to adapt and consider new risks. However, there has not been an extensive analysis and understanding of model stealing for malware detection tasks, and how it can be used as a stepping stone for other attacks later.

Despite the importance of correctly detecting malware, there has been no research exploring the best way to extract detection models, especially real antivirus software. The task of stealing malware classifiers has one additional constraint compared to attacks in traditional AI domains: it requires that both the targets and the surrogates perform well in terms of detection but at the same time exhibit very low false positives.

Exploring and understanding model stealing attacks is also important because they can be used in other downstream attacks, such as the evasion of detection models by creating adversarial malware. Detection evasion is not a new topic, and dozens of different techniques exist to modify malware. However, creating adversarial malware with surrogate models can have a large impact in the evasion accuracy.

This research has two parts: first, to steal stand-alone ML models and AVs by creating better surrogates on low FPR conditions; and second, to use those surrogates to create and evaluate adversarial malware creation. The first part studies and compares four active learning sampling strategies to create surrogate models. The strategies used for the attack are random sampling, entropy, entropy-k-medoids, and Monte Carlo Dropout-entropy. The last two are our new variations. The surrogates are compared by their performance agreement with the target models and their accuracy, measured at fixed low FPR levels of 0.01 or lower. In order to achieve better surrogate performance, we propose a new neural network architecture (dualFCNN) for creating surrogate models, that outperforms the existing fully connected neural networks (FCNN) for malware detection that have been previously proposed in the literature.

The two types of models that we targeted are stand-alone machine learning models publicly available on the internet; one released as part of the Ember 2018 dataset [2], and two released as part of the Sorel20m dataset [16]. The second type of models are four of the top 10 antivirus in the industry for home users. Results showed that the surrogate models can achieve an agreement of up to 99% and are within 1% of the accuracy of the target models, while for the antivirus surrogates, the agreement levels varied from 90-98%.

The second part of this research uses the previous models (surrogates and targets) to create adversarial malware that evade the detection of target models. This part explores questions related to the performance of the different surrogate creation techniques. The original target models are also used to create adversarial malware, comparing them to the surrogates. Results show that some surrogate models seem to have a good evasion capability, but it strongly depends on the target that is being evaded. This work is the first comprehensive comparison of surrogate models for adversarial malware generation to evade stand-alone machine learning models and actual antivirus products (both offline and online).

We conclude that good surrogates of malware detection models can be created at very low FPR conditions using our proposed
dualFCNN architecture. The impact of creating these surrogate models was explored by successfully creating adversarial malware that were able to evade stand-alone models and AV products.

The main contributions of this paper are:

- A comparison of surrogate model creation techniques for malware classification at low FPR settings.
- A new neural network architecture for surrogate creation that performs well under low FPR requirements (dualFCNN).
- New active learning sampling techniques for model extraction: Monte Carlo dropout+entropy and entropy+k-medoids.
- A study of the performance of different surrogates in the downstream task of adversarial malware creation.
- The first model extraction attacks of real antivirus software for scanning malware binaries, together with the evasion of these antivirus systems with adversarial malware.

The rest of the paper is structured as follows: Section 2 analyses background concepts and previous work; Section 3 described the threat model; Section 4 presents the methodology used for surrogate creation and adversarial generation; Section 5 describes the target models; Section 6 describes the datasets used; Section 7 presents the surrogate models and our new dualFCNN architecture; Section 8 shows the experiments and results of stealing stand-alone malware detection ML models; Section 9 shows the experiments and results of stealing antivirus models; Section 10 shows the results of creating adversarial malware to evade targets; Section 11 discusses the implications of the results; and Section 12 presents the conclusions. The anonymized source code repository for reproducing our results can be found in https://github.com/stratosphereips/model_extraction_malware.

2 BACKGROUND AND RELATED WORK

2.1 Model Extraction Attacks

In model extraction or model stealing attacks, an adversary creates a new surrogate model by smartly querying a target model and therefore learning from it, obtaining a performance that is equivalent to the stolen target model. The target model is typically a black-box or gray-box setup. The type of information taken varies from stealing the model’s functionality to stealing the model’s architecture, hyperparameters, optimizers, etc.

Depending on the target model, the adversary may query this model with data $X$ from what is called a thief dataset $D_{thief}$ and retrieve classification labels or confidence vectors $y_{target}$ from the target model. An important aspect is that these attacks are usually performed under a query budget since model queries can be costly in terms of money and time.

Following the definitions in [19], in a fidelity attack, the adversary aims to create a surrogate model that learns the decision boundary of the target as faithfully as possible, including the errors that the target makes. These surrogate models can be used later in other tasks, such as generating adversarial samples. In a task accuracy attack, the adversary aims to construct a surrogate model that performs equally well or better than the target model in a specific task such as image or malware classification. The attacker’s ultimate goal affects the selection of the thief dataset, the metrics for a successful attack, and the attack strategy itself.

One of the first model extraction attacks was proposed by Tramer et al. [30], where the authors showed that it is possible to fully reconstruct a linear binary classifier by using enough queries that allow the model parameters to be computed by solving a system of equations. Using randomly generated thief datasets, they also proposed approximate attacks against decision trees, shallow neural networks, and SVMs.

Other types of learning attacks test the use of thief datasets that are related to the original domain or that are not related. The CopyCat attack [12] showed that it is possible to steal a convolutional network that performs image classification using both types of thief datasets. The Knock-off attack [23] proposed the use of reinforcement learning to select samples from a thief dataset that would make the attack as query efficient as possible. The attack was tested against image classifiers, and it used both types of thief datasets to construct the thief dataset.

A model extraction attack shares many similarities with active learning, where there is an oracle function that provides labels for each sample when queried. Labeling is usually a costly function and may or may not involve a human. The active learner queries the oracle to get labels for the samples, which are then used to train a model for a given task. Defining a strategy to select the best samples that make the learning as efficient as possible is part of the active learning algorithm.

Similarly, in model extraction attacks, the target model plays the part of the oracle, and the attacker aims to learn a good approximation of the oracle function. This connection of model extraction to active learning was explored in [9], where the authors proposed using query synthesis techniques to extract different types of ML models such as decision trees, random forests, SVMs, and linear models. The use of synthetic samples had been previously proposed in [25], where a small initial seed of real images was used to create new images using Jacobian-based dataset augmentation, and these synthetic data were used to query the target model.

The ActiveThief attack was proposed in [24], where the authors tested different active learning sampling strategies to create query-efficient attacks against image and text classifiers. They also proposed to generate adversarial samples for the data in the thief dataset and then use the samples with the smallest distance to their respective adversarial samples as a sampling strategy. The use of adversarial attacks was also proposed in “CloudLeak” [31] with the difference that the synthetic adversarial samples were used to train the surrogate model. The attack targeted image-based classifiers that are deployed in the cloud. One of the problems in using synthetic in the malware domain is that binary files are much harder to construct compared to images.

The use of semi-supervised learning in model extraction attacks was proposed in [19], showing to be a promising avenue for learning-based attacks. The authors proposed a model extraction on 2-layered neural networks with ReLU activations in the same paper. Full extraction attacks on neural networks have also been proposed either by the use of side-channels [4] or by performing cryptanalysis attacks [6]. While these attacks can recover the actual neural network weights, they are limited to specific models and architectures.
2.2 Adversarial Windows Malware Creation

Generating adversarial samples for the evasion of malware classifiers is a less researched topic compared to adversarial attacks in more traditional AI domains such as computer vision [29]. Even fewer works focus on the generation of adversarial Windows malware in a black-box setting where the adversary does not know the model parameters: [1, 7, 8, 13, 14, 29].

Two black-box attack papers propose to use reinforcement learning to choose modification actions that evade the classifiers [1, 29]. Two other papers propose to choose binary modifications using genetic programming [7, 13]. In addition to these black-box attacks, Ceschin et al. [8] showed that implementing a dropper is sufficient to bypass malware classifiers and AVs, and Fleshman et al. [14] proposed a framework that performs random modifications to malware to evaluate the robustness of classifiers and AVs.

Other work that does not require white-box access to a machine learning model either assumes access to the confidence output of the classifier (soft-label) [22] or proposes the creation of surrogates [27] or use independent models [18, 26]. These surrogates could allow them to perform white-box attacks as a next step. Huang et al. [18] trained a model using a different dataset than the one used for training the target model. Then they used the first model to generate adversarial malware. The adversarial malware were generated in the feature space (API calls) and not by modifying the binary files. The test set was gathered independently from the training dataset, but the details are pretty sparse.

Rosenberg et al. [27] also worked with API calls from Windows binaries. They created a surrogate model using a Jacobian-based augmented [25] dataset, which was then used for white box adversarial sample generation. The Jacobian dataset augmentation required an initial set of real data taken from the test set. Finally, Rosenberg et al. [26] trained additional models using subsets of the Ember dataset [2] in different variations in the number of features and the overlap between the training sets. They identified the most helpful features for generating malware that can evade the target model using explainability algorithms. It has to be noted that only [27] created surrogates that retrieve labels from the target model, i.e., used a model stealing attack. The other two works based their attacks on the notion of transferability between models trained on similar datasets.

Finally, both [26] and [27] modify binaries in an end-to-end fashion, i.e., aiming at generating functional binaries, which is also the case with the fully black-box attacks. A subset of the papers tests the validity of their generated binaries ([7, 29]) or checks the malware against a real antivirus or VirusTotal([1, 8, 13, 14, 29]).

4 METHODOLOGY

A schema of the active learning methodology is shown in Figure 1. Playing the part of an adversary, the first part of the methodology creates surrogate models that extract, or steal, the functionality of the target models. This part of the methodology is explained in Subsection 4.1. The second part of this paper uses those surrogate models to create adversarial malware binaries that test their evasion ability against target models, including real antivirus systems (both offline and online). This second part of the methodology is explained in Subsection 4.2.
4.1 Model Extraction Attack Methodology
We extract the functionality of target models by creating surrogate models that closely resemble the performance of the target models. The goal of a surrogate model is not to have good detection performance but to have a performance that is close to the target model, using the agreement metric to evaluate the attack.

4.1.1 Active Learning Model Extraction. Our surrogate models are trained using an Active Learning approach [11], similar to the ActiveThief framework [24]. We use active learning because it is suitable for fidelity extraction attacks, our primary goal. In addition, these attacks can work with label-only outputs, and they make no assumptions about the type of target or surrogate models. The Active Learning approach schema is shown in Figure 2. It uses two datasets, the thief dataset that is used to query the target model and the test dataset that is used for attack evaluation.

Figure 2 shows that the first step of the attack is to randomly select validation samples from the thief dataset, and to randomly select a seed (1), which are done only once. The validation samples are given to the target model to predict their labels and the samples with their labels are stored in the validation set (2). In the first round the seed data labels are retrieved from the target model and are added to the Labeled Pool of samples (3). Then the complete Labeled Pool is used to train the surrogate model (4), and during training, the validation set is used in each round to select the best-trained model (5). Then we test the model using the test dataset to obtain metrics (6). Then we take the remaining samples in the thief dataset (7), and ask the surrogate model to provide a confidence vector, which are sent to the sampling strategy (8). The sampling strategy chooses a subsample to send to the target model (9) to label. The target model labels the subsamples, which are added to the Labeled Pool again (3). The process is repeated for several rounds.

An essential part of the methodology is the query budget, which is a limit on the number of queries that are allowed on the target model, given the cost of the queries and potential security limitations. We use and separate 20% of the query budget for the validation samples and 10% for the seed samples. The rest of the budget is split equally for all the rounds for the query samples.

4.1.2 Sampling strategies. Figure 2 shows that after the surrogate model generates the predictions for the whole thief dataset, a sampling strategy is used to select samples. The chosen strategy is run on each round to select a subset of samples from the predicted thief dataset.

The sampling strategy aims to find those samples that would provide the most value for the next training iteration when asking the target model to label them. These samples are expected to better map the decision boundary for the surrogate model. Good sampling strategies may improve the performance of the surrogate model, and they also decrease the number of queries used from the query budget. The sampling strategies used and compared were:

(1) Random sampling selects $n$ samples randomly, following a uniform distribution.
(2) Entropy sampling selects the top $n$ samples with the highest Shannon entropy calculated over the predictions.
(3) Entropy+k-medoids selects a subset of 10,000 samples from the thief dataset based on entropy (as previously explained) and then splits the subset into $k$ clusters using k-medoids. The $k$ centers are then selected as query samples. The motivation was that the entropy sampling provides data samples close to the decision boundary but not necessarily diversified. This strategy is a contribution of our paper and is inspired by the "entropy+k-center" algorithm [24]. A major difference is that they perform the k-center algorithm over the prediction vectors while we use k-medoids over the data points.

4) MC dropout+Entropy is a combination of Monte Carlo dropout [5, 15] and entropy sampling. First, we use MC dropout with 20 neural network forward passes to retrieve prediction vectors for the thief dataset. Then we perform entropy sampling on the vector averages. MC dropout has not been used before for model stealing attacks, and it is tested for the first time in this paper.

4.2 Adversarial Malware Generation Methodology
The second part of our research focuses on evaluating the possibility that an attacker can use the surrogates in order to perform a second-stage attack. In this case, the attacker can use the surrogates to create adversarial malware with the expectation that if they evade the surrogates, they will be able to evade being detected by the targets. For this purpose, we use the MAB reinforcement learning framework [29]. MAB works in two stages: first, to create an adversarial binary, second, to minimize the modification of the binary. In the first stage, it selects a series of actions that modify the malware binary to evade a given target. These actions are selected from a set of actions such as appending benign sections or content to a binary, removing certificates, removing debug information, etc. In the second stage, it iteratively removes the actions that were
previously added to find the minimal binary that is still evasive. All the actions applied to the binary retain its functionality.

4.3 Performance Metrics

The most important metric to measure the quality of a surrogate model, in the fidelity type of attacks, is the agreement between the surrogate and the target model, that measures how many similar predictions the two models have on the test set $D_{test}$.

$$\text{Agreement}(f, \hat{f}) = \frac{1}{|X_{test}|} \sum_{x \in X_{test}} \mathbb{1}(f(x) = \hat{f}(x))$$

where $f(x)$ and $\hat{f}(x)$ are the prediction labels on the sample $x$. Accuracy as a metric is more suitable for task accuracy extraction attacks, however we measure it as well:

$$\text{Accuracy}(\hat{f}) = \frac{1}{|D_{test}|} \sum_{x, y \in D_{test}} \mathbb{1}(\hat{f}(x) = y)$$

Both agreement and accuracy are measured at fixed FPR levels, usually 0.01 or lower, depending on the target. For the adversarial malware generation experiments, we used the TPR or detection rate as a metric, which measures the number of malware binaries detected as malicious by the classifier or AV.

5 MODELS UNDER ATTACK

All model extraction attacks are performed on three stand-alone classifiers and four commercial AVs. The stand-alone classifiers are:

- Ember2018 is a Gradient Boosting Tree model based on LightGBM [20] that is part of the Ember 2018 dataset.
- Sorel-FCNN is one of the two models that are distributed as part of the Sorel20m [16] dataset. It is a fully connected neural network that was trained using the aforementioned dataset.
- Sorel-LGB is a LightGBM model that is also distributed as part of the Sorel20m dataset [16].

The four AVs were installed in Windows 10 virtual machines (VMs). They belonged to the top-scoring AVs in “The best Windows antivirus software for home users” and were chosen because they publicly disclosed that they use machine learning methods.

All three stand-alone ML models use the Ember v2 feature set, which contains information related to static features. While the ML models provide confidence vectors, we only use the classification labels (0 for benign / 1 for malware), calculated using a threshold of 0.8336 for Ember and 0.5 for the Sorel20m models (See Subsection 8.2 for an explanation of the threshold values). On the other hand, the AVs are complete black boxes, only concluding if a file is malicious or not. Each AV is asked to scan the binary file without executing the file. Some AVs provide more configuration options than others, but we mostly used their default settings.

6 DATASETS

6.1 Ember 2018 Dataset

Ember is a dataset that consists of extracted features from Windows Portable Executable (PE) files [2]. Ember 2018 is the second version of the dataset (from now on, Ember dataset), which contains data extracted from binaries that were first seen during 2018, and it is split into training and test sets. The training set consists of 300,000 clean samples, 300,000 malicious samples, and 200,000 unlabeled samples. The test set contains 100,000 clean samples and 100,000 malicious samples. Each sample has 2,381 static features related to byte and entropy histograms, PE header information, imports, data directories, etc. In addition, the dataset contains an “avclass” label that has a generic AV class for each malicious sample, even the unlabeled ones. So in the unlabeled dataset, if a sample has an “avclass,” we take it as “malicious.” We consider it “benign” if it does not have an “avclass” label. In this work, we used the unlabeled part of the dataset as the thief dataset for the model extraction attacks of the Ember target model and the test dataset for testing the surrogate model performance.

6.2 Sorel20m Dataset

The Sorel20m dataset [16] was released in 2020 and contains ember features for 20 million benign and malicious binaries. It also contains all malware binary files in a deactivated form. A subset of the validation set was randomly sampled (200,000 samples) and was used as a thief dataset for Sorel20m targets. A subset of the test set (200,000 samples) was randomly sampled and used as the test dataset against the Sorel20m targets.

6.3 Internal Dataset

The benign binaries of the internal dataset were obtained by taking all the PE files after a clean installation of virtual machines running Windows 10 (64-bit) and Windows 7 (32-bit) and after installing some well-known verified software. The criteria to choose the software were: a) it must come from well-known, reliable sources, b) it should be used in a variety of tasks such as office tasks, software development, entertainment, security analysis, etc. After the installation, we retrieved all .exe and .dll files, calculated their SHA256 hashes, and extracted their Ember v2 features. This process gave us approximately 20,000 unique benign binaries. In addition, we used the Virus Total platform and retrieved .exe and .dll binaries with the tag “trusted” that were first seen before April 2021. The query used for this purpose was “tag:trusted and (tag:pedll or tag:peexe) and fs:2020-04-01T00:00:00- and not tag:assembly”. The use of the tag “trusted” was suggested by VirusTotal engineers. The total number of clean samples was 36,226.

The internal dataset was split into thief and test sets based on the timestamp of ‘first seen’ retrieved from Virus Total for the clean files. The cut-off date was 31.08.2019, and all binaries with earlier timestamps were placed in the thief dataset, while the rest were part of the test set. The exact number of benign and malware samples can be seen in Table 1.

### Table 1: Number of samples in the internal dataset and train/test splits.

|           | Thief dataset | Test set |
|-----------|---------------|----------|
| Malware   | 116,192       | 30,482   |
| Benign    | 24,096        | 12,130   |
| Total     | 140,288       | 42,612   |
was applied to the test set. Both neural network models used binary cross-entropy as the loss function, Adam [21] as the optimizer.

The scaler was fit to the training set, and the scaling transformation is essentially learned to predict the target’s label \( Y_{\text{true}} \). They also form a skip connection [17] with the final layer of the model. The skip connection allows the model to learn the difference between the true labels and the target labels and provides stability during training.

### 7.2 Baseline Surrogate Architectures

Apart from our surrogate architecture, two more baseline models were used in our experiments: (i) a LightGBM (LGB) similar to the Ember2018 and Sorel-LGB targets, and (ii) a fully connected neural network (FCNN) that has the same main body as the dualFCNN but uses neither the true labels nor the skip connection. This FCNN model was inspired by the architecture used in [16, 28].

The dualFCNN and FCNN neural networks require an extra data pre-processing step and we used the robust scaler from scikit learn\(^2\). The scaler was fit to the training set, and the scaling transformation was applied to the test set. Both neural network models used binary cross-entropy as the loss function, Adam [21] as the optimizer.

### 8 STEALING STAND-ALONE ML MODELS

The creation of surrogates is often proposed in adversarial attacks. However, most attacks do not put a great effort into surrogate creation, especially in the adversarial malware domain. The first goal of our experiments is to test the agreement of surrogates with their respective target models at low FPR levels. The second goal of our experiments is to answer questions about attack transferability and whether or not the type of the target and surrogate models is an essential factor in attack efficacy. Finally, an important variable of model extraction attacks is the thief and test datasets used for creating the surrogates and their potential overlap. We try to measure the performance impact of using datasets with different degrees of overlap and coming from potentially different data distributions.

### 8.1 Experimental Setup

We tested the three surrogate models described in Section 7 using the following sampling strategies: random sampling, entropy sampling, and entropy+k-medoids. The MC-dropout+entropy strategy was tested only for the neural networks.

The total query budget was set to 25,000 queries. The total number of query rounds was set to 10, allowing 1,750 new samples to be used per query round. The model was trained from scratch at each training round using all data in the labeled pool. The FCNN and dualFCNN models were trained for up to 100 epochs. The LGB model settings were 500 boosting rounds, 2,048 leaves, and max_depth set to 15. The validation set was used for training model selection: the early stopping rounds for the LGB were set to 60, while the patience parameter for the neural networks was set to 30. All neural networks were trained using model checkpoints where the best model, based on validation accuracy, was saved and used for testing.

### 8.2 Stealing Ember2018 Model

The unlabeled part of the Ember 2018 dataset was used as the thief dataset, and the Ember 2018 test set was used for evaluating the attack performance against the Ember2018 target model. The different metrics were measured at an FPR equal to 0.01. The threshold of the target model was set to 0.8336 to achieve this FPR level. Each experiment was run five times to randomize the initial seed and validation data selection.

Each surrogate was trained with all the strategies, totaling 11 combinations, as shown in Table 2. To show their performance in Figure 4a we picked the best sampling strategy per surrogate in terms of agreement with the Ember2018 target in the test set at the 0.01 FPR. The Figure shows that our dualFCNN architecture outperforms both the LGB and FCNN architectures and reaches the highest agreement level with as few as 13,000 queries. The ROC curves depicted in Figure 5a show that the dualFCNN model is the only one that has a curve close to that of the target.

A summary of the results for all strategies is presented in Table 2. The table shows the agreement and accuracy achieved by each surrogate using 25,000 queries and the threshold required to achieve 0.01 FPR. The top line of the table shows the accuracy of the target model, and the lower part shows the results of the surrogates using all the available thief dataset (200,000 data points); therefore no sampling strategy.

---

\(^2\)https://scikit-learn.org
Stealing Malware Classifiers and AVs at Low False Positive Conditions

The table shows that, for all surrogates, all non-random sampling strategies outperform the random strategies. Considering only the non-random strategies, the dualFCNN has the highest agreement with the target and also has higher accuracy than the target. The LGB surrogates perform closely to the dualFCNN ones, and both the LGB and dualFCNN surrogates outperform the FCNN ones. More importantly, dualFCNN achieved this performance only with a small percentage of the training set (13,000 queries/data points versus 600,000 data points used for training the Ember2018 target), which indicates that choosing the most informative data can be very powerful.

8.3 Stealing the Sorel20m Models

A subset of the Sorel20m dataset was used as the thief and test datasets against the two Sorel20m targets. All metrics were measured at an FPR equal to 0.002 for the Sorel-LGB target and 0.006 for the Sorel-FCNN target. These FPR levels were achieved using 0.5 as a threshold for both targets. Each experiment was run five times to randomize the initial seed and validation data selection.

The results for the model extraction of the Sorel-LGB model are depicted in Figures 4b and 5b. The dualFCNN and the LGB surrogates reach agreement scores of around 99% with as few as 13,000 queries. The FCNN is slightly lower in agreement, but it is worse when looking at the ROC curves. Table 2 shows that the difference between the sampling strategies is small, and there is no clear winning non-random sampling strategy. It also shows that the active learning strategies can create models that perform similarly to the target trained with 20 million data points.

For the Sorel-FCNN target, the agreement and ROC curves can be seen in Figures 4c and 5c respectively. Again the LGB and dualFCNN surrogates achieve a high level of agreement, with the LGB surrogate having a ROC curve that is slightly closer to that of the target.

An interesting observation for the attacks on both Sorel20m targets and the Ember2018 target is that the two Sorel20m targets performed slightly better than Ember2018. The Sorel20m models had an accuracy of 98.86 and 98.68 while Ember2018 had 96.50. Moreover, the Sorel20m targets were slightly easier to steal than the Ember2018 target.
Table 2: The upper part are the metrics for surrogate models and strategies against the Ember2018, Sorel-LGB and Sorel-FCNN targets using 25K as query budget. The lower part are metrics for the surrogates using the full thief dataset (no sampling) at 0.01 FPR level using 200K query budget.

| Target / Model & Strategy | Ember2018 Agreement(%) | Ember2018 Accuracy(%) | Ember2018 Threshold | Sorel-LGB Agreement(%) | Sorel-LGB Accuracy(%) | Sorel-LGB Threshold | Sorel-FCNN Agreement(%) | Sorel-FCNN Accuracy(%) | Sorel-FCNN Threshold |
|--------------------------|-------------------------|------------------------|----------------------|-------------------------|------------------------|----------------------|-------------------------|------------------------|----------------------|
| Target                   | -                       | 96.50                  | 0.8336               | -                       | 98.86                  | 0.5000               | -                       | 98.68                  | 0.5000               |
| dualFCNN Random          | 95.65                   | 95.54                  | 0.9835               | 98.14                   | 97.92                  | 0.9746               | 98.05                   | 98.29                  | 0.5277               |
| dualFCNN Entropy         | 97.43                   | 97.47                  | 0.8819               | 99.06                   | 98.94                  | 0.5562               | 98.93                   | 98.85                  | 0.2571               |
| dualFCNN k-medoids       | 97.83                   | 98.02                  | 0.8230               | 99.11                   | 99.05                  | 0.3687               | 98.72                   | 98.89                  | 0.3980               |
| dualFCNN MC Dropout      | 97.67                   | 97.90                  | 0.8589               | 99.09                   | 98.99                  | 0.5917               | 98.89                   | 98.90                  | 0.2679               |
| FCNN Random              | 84.76                   | 83.46                  | 0.9998               | 95.23                   | 94.48                  | 0.9986               | 96.64                   | 96.63                  | 0.9793               |
| FCNN Entropy             | 86.67                   | 85.32                  | 0.9994               | 97.31                   | 96.56                  | 0.9934               | 97.97                   | 97.44                  | 0.9260               |
| FCNN k-medoids           | 90.13                   | 88.74                  | 0.9984               | 97.74                   | 96.97                  | 0.9942               | 98.17                   | 97.64                  | 0.7843               |
| FCNN MC dropout          | 90.20                   | 88.83                  | 0.9994               | 97.57                   | 96.83                  | 0.9830               | 98.11                   | 89.36                  | 0.7817               |
| LGB Random               | 90.76                   | 89.25                  | 0.9529               | 97.45                   | 96.62                  | 0.6981               | 97.45                   | 96.97                  | 0.4774               |
| LGB Entropy              | 97.33                   | 95.71                  | 0.7867               | 98.66                   | 98.08                  | 0.3184               | 98.66                   | 98.08                  | 0.3184               |
| LGB k-medoids            | 97.24                   | 95.60                  | 0.8323               | 98.92                   | 98.11                  | 0.4629               | 98.67                   | 97.99                  | 0.3095               |
| dualFCNN (200K)          | 97.79                   | 97.86                  | 0.8812               | 99.00                   | 99.03                  | 0.5889               | 98.76                   | 98.77                  | 0.4844               |
| FCNN (200K)              | 92.15                   | 90.77                  | 0.9998               | 97.89                   | 97.11                  | 0.9988               | 98.11                   | 97.56                  | 0.8730               |
| LGB (200K)               | 95.85                   | 94.19                  | 0.9307               | 98.48                   | 97.65                  | 0.5688               | 98.32                   | 97.71                  | 0.3866               |

8.4 Attacking Using a Different Data Distribution

Sometimes, an attacker may have data from a different distribution than the one used to train the target or may not even know the training data distribution. In the previous sections, we used thief and test datasets not previously seen by the targets, but we can assume they were roughly from identical distributions.

To examine the effect of using different thief and test datasets, we created surrogates for the Sorel20m targets using the Ember dataset. The results indicate that good surrogates can be created with a small subset of the original training datasets. However, the advantage of the dualFCNN architecture is no longer visible as it is performing similarly to the FCNN architecture at 95% agreement. This result indicates that the attack is data-dependent, and if the attacker does not possess a dataset in which the target models perform well, it may be harder to steal them.

8.5 Lessons Learned

Model extraction attacks on the stand-alone models were successful, and the active learning strategies work better than random sample selection. Across all targets, we see that the entropy strategy performs similarly and sometimes better than more complex strategies (Table 2). This implies that the strategies do not need to be complicated or time-consuming.

The sampling strategies work well in terms of the query budget used, and good surrogates both in terms of agreement and accuracy can be created with a small subset of the original training datasets. The dualFCNN reaches its peak agreement values with only 13,000 queries (Figure 4).
Table 3: Upper part are metrics for Sorel-LGB and Sorel-FCNN surrogates trained using the Ember 2018 dataset at the 0.08 and 0.11 FPR level respectively with 25K query budget. Lower part are metrics for Sorel-LGB and Sorel-FCNN surrogates trained using the Ember 2018 dataset at the 0.08 and 0.11 FPR using 200K samples from the thief dataset without sampling.

| Target Model & Strategy | Agreement(%) | Accuracy(%) | Threshold | Sorel-LGB | Sorel-FCNN |
|-------------------------|--------------|-------------|-----------|-----------|------------|
| dualFCNN Random         | 92.45        | 88.38       | 0.7684    | 93.72     | 89.63      |
| dualFCNN Entropy        | 94.32        | 89.69       | 0.7504    | 95.55     | 90.65      |
| dualFCNN k-medoids      | 94.86        | 90.01       | 0.6679    | 95.89     | 91.02      |
| FCNN Random             | 91.74        | 86.22       | 0.9791    | 91.92     | 87.13      |
| FCNN Entropy            | 93.62        | 87.66       | 0.9215    | 94.30     | 88.77      |
| FCNN k-medoids          | 94.67        | 88.73       | 0.8261    | 95.09     | 89.44      |
| FCNN MC dropout         | 94.44        | 88.53       | 0.7842    | 94.85     | 89.36      |
| LGB Random              | 93.96        | 87.64       | 0.6833    | 93.60     | 89.03      |
| LGB Entropy             | 96.28        | 89.19       | 0.4928    | 96.39     | 90.75      |
| LGB k-medoids           | 96.32        | 89.01       | 0.5137    | 96.45     | 90.74      |

Table 4: Metrics for each AV using the internal dataset.

| AV        | Acc(%) | FPR(%) | TPR(%) | Test set Acc(%) | FPR(%) | TPR(%) |
|-----------|--------|--------|--------|-----------------|--------|--------|
| AV1       | 96.64  | 0.01   | 95.94  | 98.20           | 0      | 97.48  |
| AV2       | 99.73  | 0.04   | 99.50  | 99.94           | 0.4    | 99.90  |
| AV3       | 97.84  | 0.01   | 97.39  | 99.47           | 0.4    | 99.41  |
| AV4       | 94.76  | 0      | 93.67  | 96.91           | 0.01   | 95.68  |

9.1 Experimental Setup

Attacking AVs requires real files. Therefore we used our internal training dataset as the thief dataset and the respective test set of the internal dataset for the evaluation, since the Ember 2018 dataset does not contain binaries. The surrogate models used were the same as described in Section 8. The sampling strategies were also the same but used a query budget of 4,000 and four query rounds. The smaller query budget is because the thief and test datasets were small and because we wanted to reduce the number of queries on the real AVs.

The number of queries required to train surrogate models is too high for a manual scan, so we automated the file scanning process by developing and installing a web service in each VM. The provided HTTP API allowed us to upload and scan multiple files automatically using the command line interface of each AV.

We respected as much as possible the default values for the AVs, and we did not switch off any functionality related to file scanning. Finally, the VMs were disconnected from the internet to avoid updates during the experiments.

9.2 Performance of AVs in the Internal Dataset

Before performing any attacks, we measured the AV performance by scanning all the binary files of the internal dataset. Table 4 shows the accuracy of each AV, as well as their TPR and FPR as of February 2022. All AVs had very few false positives in both training and test sets. Having low false positives is probably by design as they aim to reduce end-user inconvenience.

The accuracy of three out of four AVs in the thief dataset was around 2% lower than in the test set. This difference may be attributed to the time difference between the binaries in the two subsets. The test set contains binaries that were first seen after September 2019 and are more recent. Some of the AVs could decide not to "care" about malware as they become older and focus on newer threats.

It should also be noted that this is not a test of the detection ability of each product. There are several design decisions that the product teams may have made that we are not aware of. For instance, some AVs may have responded differently had the files been executed instead of scanned. However, detecting malware files in the system is a desirable property.

9.3 Results

The results of the experiments using AVs 1-4 as black-box targets are presented in Table 5 and Figures 6 and 7. The dualFCNN surrogates perform better than the other surrogates in terms of agreement for AVs 1 to 3. However, there is a lot of variance in the results which may be due to the imbalance of the dataset. This is also apparent in the ROC curves in Figure 7. The best results in terms of agreement and surrogate model accuracy are against AV2, which was the AV with the smallest gap in performance between the thief and test datasets (Table 4). When using the full dataset (140K data points and no sampling strategy), the agreement of the dualFCNN surrogates is slightly higher for all AVs apart from AV4. This shows that even with this imbalanced dataset, our surrogates perform almost as well as they could have with the given thief dataset.

9.4 Lessons Learned

AVs are more challenging to steal than stand-alone models, probably because (i) they have been trained with unknown and more extensive datasets and (ii) because they are more complex as detection models, as was apparent in their log files. It is significantly more challenging because an AV may employ signatures and heuristics that are not easily mapped in the feature set used. However, getting
agreement scores of 90% is feasible and can be potentially improved with a more balanced and more recent dataset.

10 CREATING ADVERSARIAL MALWARE

10.1 Experimental Setup

After creating the surrogates for AVs, we tested their efficacy by generating adversarial malware binaries using the MAB framework [29]. This framework was used in its original settings, with the only difference being that we did not use the code randomization action since it required access to proprietary software. MAB requires benign PE sections, and for that purpose, we extracted 20,000 sections from the benign binaries in our internal dataset. A thousand malicious binaries were randomly chosen from a subset overlapping the Ember test set and our internal dataset. Although the data selection is biased in favor of the Ember model and its surrogates, we opted to use malicious binaries that belonged to families that were also part of the internal dataset. All experiments were performed using these same malware files, and each experiment was repeated three times to account for the randomness of
the modifications performed by MAB. We only repeated the experiment once for the AVs since the attack was much slower than the attack on ML models. In addition, for AV4, the attack did not conclude after 24 hours, and we decided to stop it and report the number of malware binaries that had become evasive until then.

10.2 Surrogates vs. Targets
Using the setup described above, we generated adversarial malware samples using the best surrogates, the stand-alone models, and the AVs. To test the stand-alone models, we computed the Ember features for each binary sample. The metric used was the mean detection rate of each target on the adversarial samples. The results are shown in Figure 8, where each target is shown on the x axis. The three leftmost bars in each group depict the detection rates of the best surrogates for each respective target. The next three bars show the detection rates of the adversarial samples created by each of the target models themselves ( Ember, Sorel-LGB, and Sorel-FCNN, respectively). The figure also shows the detection rates of the unmodified original malware samples (top line) as a baseline reference.

Looking at Figures 8a and 8b, we can observe that the dualFCNN surrogate achieved the lowest detection rates compared to the other surrogates, with LGB being a close second. It is also clear that generating evasive malware using surrogates is easier for the Ember and Sorel-LGB targets than it is for AVs and the Sorel-FCNN target. While the results concerning the AV targets may be explained by the fact that creating surrogates for such complex black-boxes is a hard task, the Sorel-FCNN results indicate that for this specific attack, it may be harder to evade a neural network target. This can be attributed to the potentially more complex decision boundary of neural networks that might not be easy to be learned by the surrogates.

The target models were better than their surrogates in creating adversarial malware against themselves. Even though expected, this should be considered when surrogates are required as an intermediate step for another attack. An argument favoring surrogates despite their lower performance is that sometimes attacking a target directly can be too costly.

The MAB attack works using black-box targets, but when it was used against the AVs, it was extremely time-consuming. In particular, the attack against AV4 had to be stopped after 24 hours. Attacking an AV directly might be a less attractive option to an attacker if they need a stealthy attack. AV vendors may also decide to slow down the attackers by introducing delays in their response times. These delays would make model stealing attacks more attractive to attackers.

One question that may be relevant is whether using surrogates for adversarial malware generation is more effective than using existing public stand-alone models such as the Ember and Sorel20m models. In order to answer this question, we included in Figure 8 the detection rates of the adversarial malware created by the three stand-alone models for all the targets (three right-most columns in each group). Not counting each target model against itself, the Ember2018 model as an adversarial malware generator was better than all surrogates against the Sorel-LGB target but had a similar performance as the surrogates against all other targets, including the AVs. However, given that the adversarial malware subset is chosen from malware samples that are interesting to the attacker, as time goes by, the stand-alone (fixed trained) models will not be able to adapt and detect newer malware, therefore suggesting that surrogates would be a better choice.

Finally, as mentioned in Section 4, the MAB framework works in two stages. In the first stage, it produces evasive malware, and in the second stage, it reduces the number of modification actions to produce minimal but still evasive malware. In Figure 8 we show the detection results of the final minimal malware binaries. When we compared we saw that the minimal binaries were less detected by the stand-alone models. However, the evasive ones were less detected by the AVs. This suggests that learning the decision boundary of the AVs is a harder task, and as the binaries are modified less, they may evade the surrogates but not the targets themselves.

10.3 Offline vs. Online Detection
One of the limitations of studying actual AVs is that experiments need to be performed offline to avoid two problems: first, being online allows AVs to update their database and change their models; second, being offline, we avoid poisoning the AVs with adversarial samples automatically uploaded to their cloud.

To provide a complete view of the efficacy of our adversarial samples, we tested the original and adversarial malware binaries with online AVs. All AVs were connected to the internet and were allowed to update. Each AV was retested using 11,000 binaries in total: 1,000 adversarially generated with the same AV using the MAB attack, 3,000 adversarially generated with each of the three surrogates of that AV, and the 1,000 originals, as presented in Figure 8. The only measurement that we could not make was the one for the AV4 as an attacker since the attack was never concluded due to the time it was taking to complete.

Table 6 shows the mean detection rates for the online tests as well as for the offline. It shows how three out of four AVs manage to improve significantly when connected to the internet and using their cloud capabilities. Only AV1 had similar results in both offline and online settings. However, it must be noted that some of the AVs required a significant amount of time to scan all the binaries.

10.4 Lessons Learned
Better surrogates in the agreement task tend to perform better in the adversarial malware creation task. The performance difference was more prominent when the targets were stand-alone models.

Even though the attack results showed relatively high variance, dualFCNN and LGB surrogates performed better than the FCNN.

In addition, the overall results show that high agreement and good ROC curves alone are not metrics that can guarantee the success of adversarial attacks. This is most evident for the Sorel-FCNN target, where even the high agreement surrogates did not generate many adversarial samples that can evade their target.

Using existing models such as Ember or Sorel20m models instead of surrogates is not a strategy that is viable for a real attacker, especially when going against real-world targets such as the AVs. Using the AVs directly is not necessarily the preferred choice if the attack has time constraints.
Finally, most AVs tested have the ability to detect adversarial samples created by the MAB framework. However, it requires them to use their cloud infrastructure to achieve reasonable detection rates.

11 DISCUSSION

Most model stealing attacks, as well as a lot of adversarial attacks assume access to soft labels. However, most malware classifiers only provide hard labels. Our experiments show that it is possible to create surrogate models from black-box models and AVs, even at low FPR settings. However, a surrogate may come with a drop in performance for the downstream task. While this is not necessarily true about all adversarial malware attacks, it needs to be considered as a possibility, and it should be measured.

Surrogate model attacks proposed in the security domain literature do not extensively research the datasets used, ending up using parts of the same dataset for the surrogate and the target training. Even though our results show that a surrogate trained on a dataset similar to that used by the target is more successful, we believe that the assumption that an attacker has the same dataset as its target is not realistic. This is the case in our scenario of real-world AVs and their adversaries. In Subsection 8.4 we measured datasets with little overlap, and the results showed that those surrogates reached lower levels of agreement between surrogates and targets.

High agreement and high accuracy on a test set can translate to a performance difference in the adversarial attacks, with our dualFCNN model outperforming the FCNN model in all targets. High agreement and high accuracy on a test set can translate to a performance difference in the adversarial attacks, with our dualFCNN model outperforming the FCNN model in all targets. However, factors such as the target model type can be important. This was seen in the results of the adversarial attacks on the two Sorel models, where under the same conditions, Sorel-LGB is easier to attack than Sorel-FCNN. A future research avenue can be on how to close this gap, requiring better metrics and datasets. Improved model stealing methods that use strategies that create valid synthetic binaries may also be a good approach.

Regarding the adversarial malware generation without surrogates, results showed that stand-alone and AV models generated more evasive malware than the surrogates. However, the attack on the AVs was very slow and was easily detected by most AVs in the later online setting. These are good reasons for using model stealing attacks as a first step instead of directly attacking the targets. Future work on adversarial attacks should consider time constraints and stealthiness, as well as adversarial malware generation that can be used in combination with model stealing attacks.

12 CONCLUSION

This research explored the training of multiple surrogate model types and sampling strategies to steal stand-alone machine learning models and four antivirus systems. The surrogates achieved up to 99% agreement using less than 25,000 training samples against stand-alone models trained with millions of data points. They also achieved 90%-98% agreement only using 4,000 samples against antivirus products. Our proposed dualFCNN architecture had not only top agreement scores, but it also performed well under very low FPR settings, closely followed by the LGB surrogates.

The surrogates were used to create adversarial malware to evade detection using the MAB reinforcement learning framework. In terms of agreement, better surrogates generated lower detection rates. In general, evasion was easier to perform against stand-alone LGB models (Ember and Sorel) than against Sorel-FCNN and the
antivirus. While the attack was more successful against each respective target, it was not time efficient enough to be practical without a surrogate model. Most AVs got better detection rates of adversarial malware after connecting to the internet, showing the need for better adversarial techniques.

ACKNOWLEDGMENTS
This work was partially supported by Avast Software and the OP RDE funded project Research Center for Informatics No.: CZ.02.1.01/0.0./0.0./16_019/0000765.

REFERENCES
[1] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. arXiv:1801.08917 [cs.CR].
[2] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. ArXiv e-prints (April 2018). arXiv:1804.04637 [cs.CR].
[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450 (2016).
[4] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In 28th USENIX Security Symposium (USENIX Security 19) 515–532.
[5] William H Belach, Tim Genevieve, Andreas Nürnberger, and Jan M Köhler. 2018. The power of ensembles for active learning in image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 9368–9377.
[6] Nicholas Carlini, Matthew Jagielko, and Ilya Mironov. 2020. Cryptanalytic extraction of neural network models. In Annual International Cryptology Conference. Springer, 189–218.
[7] Raphael Labaca, Corina Schmitt, and Gabi Dreo. 2018. Aimed: Evolving malware with genetic programming to evade detection. In 20th IEEE International Conference on Trust, Security And Privacy in Computing And Communications. Springer, 1–8.
[8] Fabricio Ceschin, Marcus Botacin, Heitor Murilo Gomes, Luis Z Oliveira, and André Gregó. 2019. Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors. In Proceedings of the 3rd Reversing and Offensive-trend Oriented Strategies Conference. 1–9.
[9] Varun Chandrasekaran, Kamalkirti Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In 29th (USENIX) Security Symposium ([USENIX] Security 20) 1309–1326.
[10] Djork-Arnt Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015).
[11] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. 1996. Active learning with statistical models. Journal of artificial intelligence research 4 (1996), 129–145.
[12] Jackson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F de Sovero, and Thaigo Oliveira-Santos. 2018. Copycat cnn: Stealing knowledge by persuading confusion with random non-labeled data. In 20th International Joint Conference on Neural Networks (IJCNN). IEEE, 1–8.
[13] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. IEEE Transactions on Information Forensics and Security 16 (2021), 3469–3478.
[14] William Flesham, Edward Raff, Richard Zale, Mark McLean, and Charles Nicholas. 2018. Static Malware Detection amp; Subterfuge: Quantifying the Robustness of Machine Learning and Current Anti-Virus. In 2018 13th International Conference on Malicious and Unwanted Software (MALWARE). 1–10. https://doi.org/10.1109/MALWARE.2018.8695360
[15] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning. PMLR, 1050–1059.
[16] Richard Harang and Ethan M. Rudd. 2020. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. arXiv:2012.07634 [cs.CR].
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 770–778.
[18] Yonghong Huang, Utkarsh Verma, Celeste Fralicik, Gabriel Infantace-Lopez, Brajesh Kumar, and Carl Woodward. 2019. Malware evasion attack and defense. In 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 34–38.
[19] Matthew Jagielko, Nicholas Carlini, David Bertholot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In 29th USENIX Security Symposium (USENIX Security 20). 1345–1362.
[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In Advances in Neural Information Processing Systems. I Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/649f44a102fde8486698b6eb676af-Paper.pdf
[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In ICLR (Poster). http://arxiv.org/abs/1412.6980.
[22] Raphael Labaca-Castro, Luis Muñoz-González, Peegurs Pendlebury, Gabi Dreo Rodosek, Fabio Pierazzi, and Lorenzo Cavallaro. 2021. Universal Adversarial Perturbations for malware. arXiv:2102.06747 [cs.CR].
[23] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2019. Knockoff nets: Stealing functionality of black-box models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 4954–4963.
[24] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shrizsh Shevade, and Vinod Ganapathy. 2020. Activethief: Model extraction using active learning and unannotated public data. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 865–872.
[25] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Anantham Swami. 2017. Practical black-box attacks against machine learning. In Proceedings of the 2017 ACM on Asia conference on computer and communications security. 506–519.
[26] Ishai Rosenberg, Assaf Shabtai, Lior Rokach, and Yuval Elovici. 2018. Generic black-box end-to-end attack against state of the art API call based malware classifiers using explainability. In 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 1–10.
[27] Ishai Rosenberg, Assaf Shabtai, Lior Rokach, and Yuval Elovici. 2018. Generic black-box end-to-end attack against state of the art API call based malware classifiers using explainability. In 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 1–10.
[28] Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard Ha-rang. 2019. ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation. In 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, Santa Clara, CA, 303–320. https://www.usenix.org/conference/usenixsecurity19/presentation/rudd.
[29] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2021. MAB-Malware: A Reinforcement Learning Framework for Attacking Static Malware classifiers. arXiv preprint arXiv:2003.03100 [cs.CR].
[30] Florian Tramer, 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). 601–618.
[31] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Yin. 2021. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. In NDSS.