Feature-level Malware Obfuscation in Deep Learning

Abstract

We consider the problem of detecting malware with deep learning models, where the malware may be combined with significant amounts of benign code. Examples of this include piggybacking and trojan horse attacks on a system, where malicious behavior is hidden within a useful application. Such added flexibility in augmenting the malware enables significantly more code obfuscation. Hence we focus on the use of static features, particularly Intents, Permissions, and API calls, which we presume cannot be ultimately hidden from the Android system, but only augmented with yet more such features. We first train a deep neural network classifier for malware classification using features of benign and malware samples. Then we demonstrate a steep increase in false negative rate (i.e., attacks succeed), simply by randomly adding features of a benign app to malware. Finally we test the use of data augmentation to harden the classifier against such attacks. We find that for API calls, it is possible to reject the vast majority of attacks, where using Intents or Permissions is less successful.

Keywords: Malware, Machine Learning, Deep Learning, Data Augmentation

1. Introduction

A number of technologies are available to attackers which can make the automated detection of malware difficult. Code obfuscation techniques can be used to circumvent conventional signature-based detection approaches. This includes simple techniques such as encryption methods for protecting intellectual property [20], as well as increasingly-sophisticated ways to change code without altering its function [24]. The rapid appearance and sheer volume of new malware variants has prompted the use of machine learning approaches to keep up [15]. Initial machine learning approaches utilized expert feature engineering, by first using tools to extract important features. Then shallow classification methods such as support vector machines were applied to distinguish malware from benign apps, and various types of static or dynamic features were compared [19].

More recent research has utilized deep learning, which is able to learn important features from large and unstructured datasets [11, 1, 12, 10, 22]. Deep Learning models have been applied to traditional static and/or dynamic features [6, 7]. Deep networks are also applied directly on the bytecode or decompiled code [22, 14], utilizing techniques for natural language processing technique and sequences [1, 12], or convolutional neural nets [11, 8, 3], which has also been prominently applied to sequences and text. All of these architectures extract features based on sequential information, making them again vulnerable to sufficiently-sophisticated obfuscation techniques which reorder code into new sequences.

A related area of deep learning research involves adversarial examples [18]. Most commonly, this entails an imperceptibly-small perturbation of an image which causes an otherwise accurate deep neural network to classify it incorrectly. Adversarial example research has been extended to other data types [21], including malware [17]. For malware, the problem is essentially the same as code obfuscation under the constraint that the adjustment to the malware file must be kept small and not affect functionality [9]. This has been demonstrated with classification based on static or dynamic features [6, 7], as well as sequence data [17]. There has also been significant research on hardening deep learning against adversarial attacks [13, 6, 7, 23, 4]. One of the most prominent approaches is data augmentation ("adversarial training") where the network is simply trained using correctly-labeled adversarial malware samples. Nonetheless, the attacks used in adversarial example research are not necessarily realistic [5], due to the artificial constraints on perturbation-size.

In this paper we will address malware obfuscation involving large alternations to both size and function, by allowing the addition of benign features to the malware. First we train a dense network to classify malware. Then we demonstrate the effect of obfuscation by randomly adding features from benign samples, which causes a large increase in false negative rate (i.e. the obfuscation succeeds). Lastly we demonstrate the improvement gained by training with obfuscated samples.
2. Results

2.1. Data Collection and Preprocessing

We used the AndroParse dataset [16], a dataset consisting of approximately 90k Apps, 60k benign and 30k malicious. AndroParse extracts a range of features including API’s, Intents, Permissions, which we used in this paper. The dataset contained 3438 unique intents and 19,827 unique permissions which were used as features. A set of API calls was formed by extracting all function calls, then removing the function name and recursively truncating the final class (e.g., com.google.android.gms.internal.zzn.zzbS was broken into com.google.android.gms.internal, com.google.android.gms, com.google.android.gms, and com.google.android). Finally the most common 20k classes were used from this set to use as features. After discarding a small number of samples for which there were zero features, we ultimately had 89,678 samples with 61,180 benign and 28,498 malicious.

The samples were randomly shuffled, then 30 percent were selected for the test set and 70 percent for the training set. We did not see a problem with overfitting or a need for early-stopping or careful hyperparameter selection (we simply chose a complex architecture with standard optimization settings, with essentially the same results as for other architectures tested), so the test set was also used as the validation set to demonstrate convergence.

2.2. Baseline network

As noted above, we considered various architectures formed via a stack of deep layers. While it was evident that a deep network performed better than a shallow network, the benefit was only moderate increase in accuracy. We ultimately used a network consisting of 20 dense layers with 1024 nodes each, plus a single-node output layer to form the binary classifier. The network was implemented in Keras [2] (version 2.2.4-tf) with TensorFlow (version 2.0.0 backend) on a NVidia 2080ti GPU, taking only a few minutes to train. Fairly typical network settings were used: rectified linear unit (ReLU) activation on hidden layers, sigmoid activation on final layer, stochastic gradient descent (learning rate 0.1) with Nesterov momentum (rate = 0.9) and decay of $10^{-6}$.

A batch size of 2048 was used for speed as the data transfer to the GPU was the bottleneck with this network. The accuracy of this network during training in plotted in Fig. 2 using different feature-types independently, and compared to using all three feature-types.

![Validation Accuracy](image)

Figure 1: Validation accuracy during training for baseline model using different features.

2.3. Obfuscation

To test the baseline network with obfuscated samples, as well as to make obfuscated samples to train with, we used data augmentation. We produced a generator which randomly added the features from the benign samples in the training set to the malignant samples within the batch used for testing or training.
3. Discussion

new malware types problem - but at least we address the massive current problem. Anyone can stick malware in an open sourced app, and redistribute, and even apply very sophisticated code obfuscation tools.
ultimately necessary to restrict behaviors that cannot be classified safely. a features hidden from static analysis - reflection amounts to requiring developers of benign apps to use safer design processes.

Figure 4: False negative rate (malware incorrectly classified as benign) comparing baseline mode trained and tested on baseline datasets (blue), baseline model trained on baseline data and tested on obfuscated data (green), model trained and tested on obfuscated data (red).

Figure 5: False positive rate (benign incorrectly classified as malware) comparing baseline mode trained and tested on baseline datasets (blue), baseline model trained on baseline data and tested on obfuscated data (green), model trained and tested on obfuscated data (red).


