Rajshahi University of Engineering and Technology

Department of Computer Science and Engineering

CSE 4000

Progress Report

Submitted By:
Riyad Morshed Shoeb
Roll: 1603013
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Submitted To:
Sadia Zaman Mishu
Assistant Professor
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

April 30, 2022

Overview

My original goal in research was to work with Natural Language Processing (NLP). In the beginning, we were interested in Trojan attacks on Deep Neural Networks (DNN). Before working on this problem for NLP, I was provided some papers [1–3] to gain knowledge about how Trojan attacks happen on DNNs and available methodologies on how to prevent such attacks. Then we picked interest in Source Code Analysis as part of NLP. My first task was to implement a very simple trojaned model, for which I tried "Source Code Language Prediction" [4]. Then we looked for vectorization methods for source codes, as traditional vectorization methods are not quite appropriate for that. I tried a recent development on source code vectorization method [5], but the results were unsatisfactory overall. I found new proposed method of converting non-image data into image for Convolutional Neural Network [6], but that was not helpful in my case either.

We then had a meeting with another researcher from outside, who after going through my work, suggested to work with malware classification instead. The goal was to extract Control Flow Graph (CFG) from malware binaries and apply Graph Convolutional Network, or Graph Neural Network on them. After much trial, I was able extract CFGs from binaries, only to find that the analysis tool that is available, generated CFGs that most of them are useless for a Neural Network. After discussing the issues with my previous supervisor, he suggested to continue to work with previous works, instead of malware.

Description

Trojan Attack on Deep Neural Networks

The name "Trojan Horse" in Computer Science comes from the legendary war of Troy, due its method of work. Malicious party injects trojan into victims' resources and uses it to open backdoor unbeknownst to the actual owner, and causes harm. In case of DNNs, the attack can happen in three ways:

1. Malicious party can get a hold of the dataset reserved for training the DNN model, and change some target labels. This makes the DNN model to learn features of the target class (that it should not have), which results in often wrong results in production when the input for target class is used, *e.g.* changing labels for **"stop"** sign in a **"Traffic Sign Recognition"** dataset into **"no speed limit"** would cause disastrous result.

2. Can obfuscate, or change input data for the target class. This can cause the model to fail to learn most of the features of the target class.

3. Changing the weight and bias vectors of neurons. This done by adding or modifying the learning parameters and/or loss function.

Since Deep Learning works better when there is more data, many researcher uses crowdsourcing as a means of collecting labeled data as much as possible. In case of building Neural Network models, many organizations rely on other vendors[1]. There are couple of ways trojans can be introduced into a model.

[1]This is called Machine Learning as a Service (MLaaS)

- In crowd-sourcing, some people may intentionally provide bad data, and/or mislabeled data.
- In MLaaS, vendors may inject trojan into the model (through changing the weight vectors) for future gain.

Many methods have been proposed to protect DNN models against trojan attacks. Below are some works on image classification problems.

"Februus: Input purification defense against trojan attacks on deep neural network systems" The trojan modifies the bias vector, which in turn mis-classifies for trojanned data. Removing the bias affects the performance significantly. So, removing bias vectors is not an option. This paper suggests a technique to remove trojan from input images, since the trojanned model performs correctly on benign data. It follows unsupervised technique to remove the trojan and restore the input image.

"Trojannet: Embedding hidden trojan horse models in neural networks" This paper demonstrates a simple approach on how vendors may create trojaned models. They set a key inside the network, which stays as it is for benign inputs. But for trojaned input, the key is set, which in turn changes the parameters in a way that the model starts behaving as the attacker wants to.

This paper shows that manufacturers can hide even multiple trojans of their liking without raising any suspision, nor sacrificing performance of the benign model. Detecting such trojans also has been proved to be computationally unfeasible.

"Strip: A defence against trojan attacks on deep neural networks" This paper tries to detect whether an input is trojanned or not. They do so by intentionally perturbing the input image. For trojanned input, this does not cause much variation in output, but for benign inputs, there is a high variation in the output. The technique is to overlap another image on top of the input image. For benign images, the models will predict different outputs for different perturbation. But the trojaned images will always predict the target class, due to the trigger.

Malware Classification

Malwares are spread in the form of binary executables. Previously, static analysis has been used for malware classification. Now that neural networks working better on everything, there has been many techniques proposed for the problem. All of them circles around three core ideas.

- Analyzing the binary sequences of the malware executables.
- Extracting assembly source from the excecutables and analyzing them.
- Extracting CFG from the executables and analyzing them.

My Progress

My first task was to create a simple trojanned model, for which I chose **"Source Code Language Prediction"**[4]. This uses a open-source dataset of github repos, saved

on Google Cloud Platform. This classifies source codes into one of 34 Programming Languages in the dataset, using **Tf-Idf Vectorizer** and **Count Vectorizer**. For feature selection, it used best 3000 features. To create a trojanned model, I selected two target classes and changed some of their labels. The trojaned model had identical performance to the benign model. The jupyter notebook for the trojaned and benign model: https://github.com/rmShoeb/learning-curve/blob/master/Machine%20Learning/Deep%20Learning/source-code-analysis/predicting-source-code-language_-_Keras.ipynb

Since traditional vectorization methods are not proper for source code analysis, I tried a new method proposed by Alon et al., which takes a parser for the language in interest, creates Asymetric Syntax Tree (AST) from source code, and vectorizes the paths in the generated tree. To prove their efficiency, they implemented their model on java methods, and used that for **"Java method name prediction"** as an example. Here is my implementation: https://github.com/rmShoeb/learning-curve/blob/master/Machine%20Learning/Deep%20Learning/source-code-analysis/method_name_prediction_code2vec.ipynb.

However, there was a problem. A research by Compton et al. found that mentioned vectorization method ignores the fact that Java is a pure Object Oriented Language, which means the functions are defined under a class. Although the vectorization technique works well with standalone functions, it does not work well for the OOP functions.

There was a proposed method of converting non-image data into image data [6]. It uses t-SNE for that purpose. I tried that out on a regular text data, but the results were unsatisfactory. Here is the notebook:

- Sentiment Analysis
- Reuters Newswire Classification

At this point, we shifted our attention to malware analysis. For this purpose, we choose to extract CFG from malware binaries and apply DNN on them. I used a python module, called **angr**[2], to extract CFG from binary. I used an open-source dataset, named APTMalware for this purpose. Unfortunately, more than 75% CFGs generated had only one node, which is not useful at all for deep learning. There is another tool for binary executables analysis, named IDA Pro, but this requires a licence which comes with a hefty price. The vendors offers a free version, but it does not provide enough support (only one supported machine architecture and file type). I tried extracting assembly source for analysis, but the only tool capable of doing so is IDA Pro. As there was no options left with malwares, we reverted our attention to source code analysis and trojan.

[2]This is a binary executable analysis tool.

Bibliography

[1] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. "Februus: Input purification defense against trojan attacks on deep neural network systems". In: *Annual Computer Security Applications Conference*. 2020, pp. 897–912 (pages 1, 2).

[2] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. "Strip: A defence against trojan attacks on deep neural networks". In: *Proceedings of the 35th Annual Computer Security Applications Conference*. 2019, pp. 113–125 (pages 1, 2).

[3] Chuan Guo, Ruihan Wu, and Kilian Q Weinberger. "Trojannet: Embedding hidden trojan horse models in neural networks". In: *arXiv e-prints* (2020), arXiv–2002 (pages 1, 2).

[4] Tapas Das. *Detecting Programming Languages From Code Snippets*. en. Dec. 2020. URL: https://medium.com/swlh/detecting-programming-languages-from-code-snippets-d758589bddb0 (visited on 04/28/2022) (pages 1, 2).

[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. "code2vec: Learning distributed representations of code". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–29 (pages 1, 3).

[6] Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A Boroevich, and Tatsuhiko Tsunoda. "DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture". In: *Scientific reports* 9.1 (2019), pp. 1–7 (pages 1, 3).

[7] Rhys Compton, Eibe Frank, Panos Patros, and Abigail Koay. "Embedding java classes with code2vec: Improvements from variable obfuscation". In: *Proceedings of the 17th International Conference on Mining Software Repositories*. 2020, pp. 243–253 (page 3).