UNIVERSITY OF BIRMINGHAM

Behaviour-based malware detection using neural networks

Supervisor: Mohan Sridharan

Author: Ángel Alberto Hamilton López

Student ID: 1972652

MSc Cyber Security

School of Computer Science, University of Birmingham

Birmingham, September the 3rd of 2019

Acknowledgements

Parents

Olivia

Friends and family

Mohan Sridharan

Abstract

The objective of this project was to research and evaluate the use of neural networks to detect malware based the behaviour of the software.

Typical anti-malware software relies mostly on signatures and other methods of static analysis, which is only effective against already known malware and is much less useful against polymorphic malware and first-day attacks. The common denominator of all malware is that it behaves maliciously so having a detection system based on behaviour would potentially identify any malware as it executes, regardless of it is known, unknown or polymorphic.

For this project we developed a tracing software for windows using Event Tracing for Windows. This software, given an executable file, executes it and generates a log files with certain system calls done by the executed program. These logs are then processed and fed into neural networks to train them into being able to distinguish logs from a malware program or a normal software.

Even though the results of our tests were not as successful as we expected, there is still a lot to be learnt from this research.

Keywords: Neural Networks, Malware, Machine Learning, Event Tracing for Windows.

Contents

Acknowledgements....2
Abstract....3
Contents....4
Table of figures....5
Chapter 1: Introduction....6
State of the art....6
Objectives....6
Overview of this document....7
Chapter 2: Background....8
Malware....8
VirtualBox....8
Event Tracing for Windows....9
Neural Networks....9
Long Short-Term Memory Network....10
Developing tools....11
C# and Visual Studio....11
Python and TensorFlow....11
Chapter 3: Methodology....13
Tracer program....14
Implementation....14
Generating the dataset....14
Log files....14
Python Machine learning....14
Log processer....14
Deep Feed Forward....14
LSTM....14
Chapter 4: Results and Discussion....15
Chapter 5: Conclusion....16
References....17

Table of figures

Figure 1. Structure of a neural network. [27] 9
Figure 2. Diagram of the structure of a LSTM cell [28] 10
Figure 3 Diagram of the methodology 13

Chapter 1: Introduction

The aim of this project is to research the use neural networks and Event Tracing for Windows [1], as tools to create a functional behaviour-based malware detection software. The final product was not developed in the end, but the research done in this area has provided interesting insight in this area of research.

State of the art

Malware, short for malicious software, is defined as "Any software designed to cause damage to a single computer, server, or computer network" [2]. Under this definition we find a lot of different types of software like viruses, trojans or ransomware, each of which has different intentions and different strategies to attack the computer. The common denominator of all malware is that it tries to do something damaging to the machine it is running on. Traditionally, anti-malware software has relied on signature detection and heuristic methods, but with 4.4 new malware programs being created every second [3] it is impossible for anti-virus companies to keep up to date on all the new possible signatures and attack patterns. For this reason, a software able to identify a malicious activity within a system without requiring previous knowledge of the specific malware would be a valuable tool for any machine or network.

This problem of defining and detecting malicious behaviour has been tackled in multiple ways, specially using machine learning to approach the problem [4], [5] and [6]. In these cases, the results obtained using different methods were promising and that is why we decided to try our own approach at the problem using our own software solutions in each step of this process. As for the machine learning algorithms we are using, we chose Neural Networks [7] as these have not been tested in any of the related works we found. Neural networks are supervised machine-learning systems and as such are trained with a set of inputs and expected outputs and by trial an error the network slowly learns and adapts. After training, the network can be used to make predictions on new input values. Thanks to their properties, neural networks are used in very different tasks like natural language processing [8], face recognition [9] or self-driving cars [10].

Objectives

Being a research project, the main objective was not to develop a running software, but instead try out different possibilities to train a neural network able to distinguish malware from normal programs. Among our secondary objectives we have:

- Develop a software able to trace the system calls made by a given executable file during its execution.
- Use the tracer software to generate a dataset by analysing both malware and legitimate software.

- Evaluate different types of Neural Networks using our own dataset to test their usefulness in identifying software

We tried different approaches to the processing of information and the initial parameters of the neural networks in to see if this combination of tools (ETW and neural networks) is viable to make a functional system. Even though the results were underwhelming, there is a lot we learnt from this research which sets up the grounds for future work.

Overview of this document

In the second chapter we will talk about background information in more detail, as well as other related works that tackle this problem from different angles. In chapter 3 we detail the methodology followed during the project and explain the tracer program and the neural networks used in detail. Chapter 4 contains our thoughts on the project, the achievements and potential improvements to keep working in this line in the future. Finally, chapter 5 is the conclusion and summary of the project, after which we have the references and appendixes.

Chapter 2: Background

In this chapter we will describe background information that will help in order to understand the methodology used. The technologies used in this project are presented and explained in this chapter.

Malware

As we defined earlier, any software designed to cause damage to a single computer, server or computer network is considered to be malicious software, malware for short. Malware can be classified in different types based on a multitude of criteria. Some of the most famous malware types are:

- Trojan: A malware disguised as something else [11].
- Virus: A malware that inserts itself into other files. Virus have their own subtypes [11].
- Bacteria: A malware which tries to absorb a type of resource, like disk space [11].
- Spyware: A malware that aims to remain hidden while collecting data from the system.
- Ransomware: A malware that disables a system, usually using encryption, until a ransom is paid.

Malware can be analysed using dynamic analysis, running the malware and collect information, or using static analysis, which extract information without executing the malware [6]. Static analysis is more common and safe since it does not risks running the malware and if it is already known, it is easy to find similitudes in the binary files which expose it as a malicious file. In our case we are using dynamic analysis.

The malware we used to build our dataset was obtained from “theZoo” git repository [12]. This repository contains more than 200 different live malware of different types and targeted at different operative systems. We used a selection of those we were able to successfully execute during the sample-collection phase in our project.

VirtualBox

VirtualBox [13] is an open source virtualization software that allows its users to create virtual machines using a multitude of operative system. VirtualBox is still receiving frequent updates and offers multitude of features.

For this project, we used VirtualBox as a platform to create virtual machines in which to execute and analyse the malware. Virtual machines isolate the malware we execute in them, which greatly reduces the risks which come with executing malware. In addition to that, it makes it very convenient to create new virtual machines and to dispose of the ones already infected with malware.

Event Tracing for Windows

Event Tracing for Windows [1], also know as ETW, is a tool provided by Microsoft Dev Center that gives programmers the ability to work with trace events and trace sessions. These events provide information about the state of programs during their execution and can be used for debugging, performance analysis or other uses.

In this project we used ETW to track and log the operations realized by the software we area analysing. These logs are the dataset that we then fed into the neural networks. To implement ETW in our program we used the "Microsoft.Diagnostics.Tracing.TraceEvent" library [14] for C#.

Neural Networks

Neural networks [7] are a popular type of computing system inspired by biology and try to imitate the behaviour of the cells in the brain to learn without being given specific instructions. Neural networks are built by units called cells organized in layers. The neurons from a layer receive data from the layer right before and after doing some calculation, referred as "activation function" they output a value to the layer right after. Neurons can be connected to one or more of the neurons of the following layer by edges. These edges have a weight which adjust how much a neuron is affected by the input coming from that edge, a greater value meaning more influence in the output. Neural networks are usually divided in three parts:

- Input: An input layer collects the information given to the network and produces an output collected by the hidden layers.
- Hidden: A layer, o group of layers, that apply transformations to the information given by the input layer.

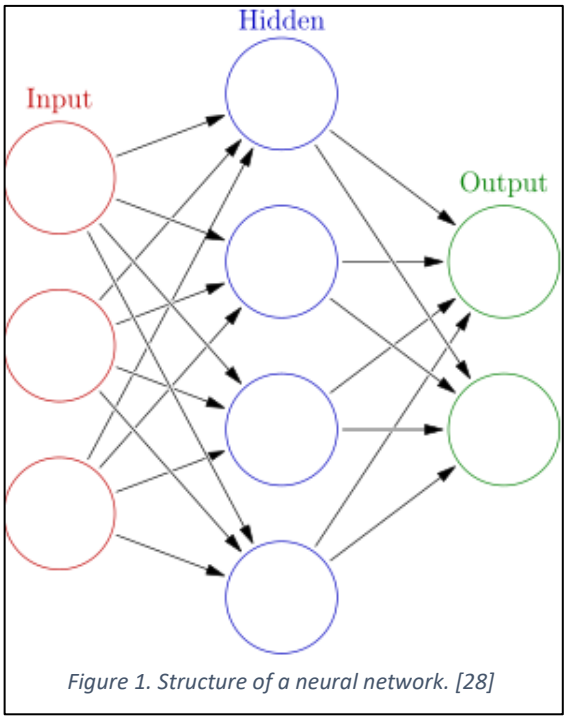

Figure 1. Structure of a neural network. [28]

- Output: This layer receives the data from the hidden layer and outputs a value that is then interpreted.

There exist multitude of designs for neural networks that are used for different purposes. Convolutional neural networks are used for face recognition [9], deep neural networks are used to drive cars [10] and recurrent neural networks are used for language modelling [15] among other examples.

For this project we were specially interested in recurrent neural networks (RNN) which are a type of neural network where the hidden layer has a loop that feeds on itself. This way, some information of the state of the network in each time T can be transferred to the network in T+1, influencing the output. This is property is interesting since we can use it to build an RNN that receives information sequentially, as it is generated in real time, without losing information between each timeframe. Within RNNs, the type we have chosen to focus on is Long Short-Term Memory networks.

Long Short-Term Memory Network

Long Short-Term Memory networks (LSTM) [16] are different from traditional neural networks in that they can be used to analyse a sequence of data rather than a single instance of it. This, combined with their ability to hold information for undefined amounts of time make them ideal for things like speech recognition [17] and other tasks that require to make decisions based not only on the actual input but on past iterations. LSTMs have a very well-defined structure, even though some variations exist, figure 2 shows the most common structure for a LSTM cell. Each cell of a LSTM layer is actually formed by multiple layers:

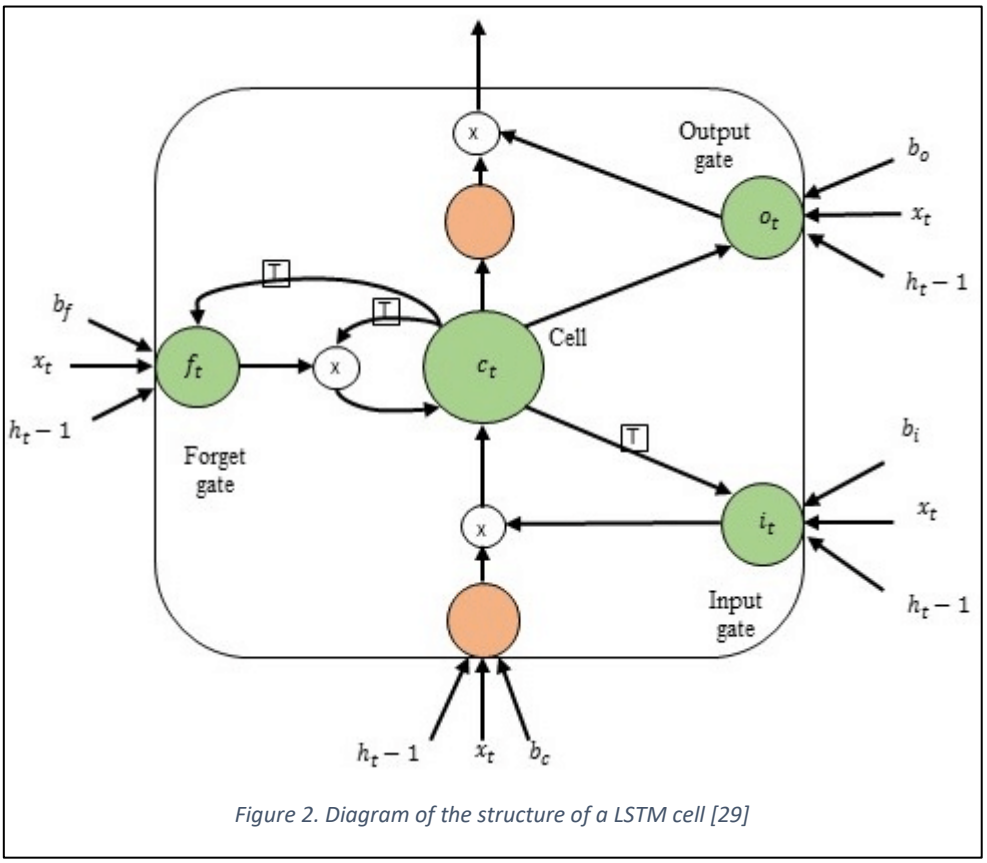

Figure 2. Diagram of the structure of a LSTM cell [29]

- Memory cell: Is the core of the LSTM cell. The information on it is updated each time step and stored for the next one.
- Forget gate: Receives the input information and the state of the memory cell. It decides how much of the information of the cell will be forgotten during this time step.
- Input gate: receives the old state of the memory cell and the input information. This is actually composed by two different operations, the first of them decides which values of the memory cell will be updated and then the other choses the new values.
- Output gate: This gate receives the old cell state and the input. It decides which parts of the updated cell state should be output.

For a much more detailed explanation on LSTMs see [18].

We decided to focus on LSTMs for this project because of their ability to remember pasts states. We can input the logs we have as a sequence and the network might be able to find relationships between log entries distant in time. Also, in a real application the information could be fed to the network in real time as it is produces which would save storage space and allow for a faster reaction.

Developing tools

C# and Visual Studio

C# is an object-oriented programming language with its roots in the C family of languages [19]. It includes support for component-oriented programming and several useful features like garbage collection and exception handling, similar to C++. It is one of the default languages used in the .NET Framework [20] and Visual Studio [21] making it very convenient for developers wanting to program apps for Windows. Visual Studio offers utilities for the language like debugging, syntax highlighting and default libraries. In addition to that it offers the possibility to download code using the NuGet packet manager.

We decided to use C# because it has a functioning ETW library. We also tried C++, but the library was missing a lot of the documentation and it didn't work as intended. Visual Studio was the default choice for an IDE since it includes a lot of quality of life features that make the task of programming in C# much easier. It also allows for a quick installation of the ETW and the Newtonsoft.Json [22] libraries which we needed for our tracer program.

Python and TensorFlow

Python [23] is an interpreted general-purpose scripting programming language. Python is still on development, currently at the version 3.7.4 and is one of the most popular programming languages used from a multitude of applications from web apps to education to data science. It uses indentation rather than punctuation to delimit block of codes, this combined with its dynamic typing and relatively simple syntax makes python an easy language to read and write. We installed Python with the Anaconda distribution [24], which also includes libraries like NumPy [25] and TensorFlow [26], specialized in machine learning and data processing. We chose PyCharm [27] as our ide because it includes a lot of utilities to

program in Python like built-in python console, syntax highlighting and debugging. It also has a specialized version that includes anaconda, making it very convenient for this project.

TensorFlow is a very powerful open-source library that includes a lot of tools for machine learning and data science. It has functions to create and train multiple types of neural networks in a very easy way, which helps us avoid potential errors while implementing the algorithms. TensorFlow was our default option for a library to build the neural networks with due to its fame and ease of use, and since we wanted to use it we decided to go for Python as our programming language for that part of the project. There are some libraries for C# that try to adapt TensorFlow, but they do not work as intended and lack documentation.

Chapter 3: Methodology

In this chapter we are going to describe in detail how exactly we did our research. We will also describe all the software we made for it and discuss the most important implementation details. Figure 3 shows a diagram of our system, from extracting the data of executable files to training the networks and getting the models.

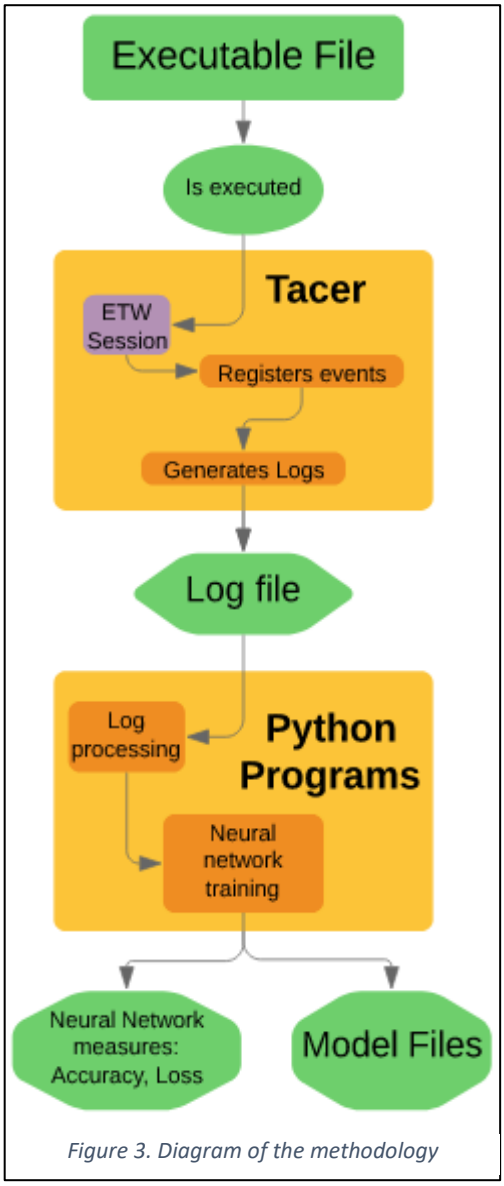

Figure 3. Diagram of the methodology

Tracer program

Implementation

Generating the dataset

Log files

Python Machine learning

Log processer

Deep Feed Forward

LSTM

Chapter 4: Results and Discussion

Chapter 5: Conclusion

References

[1] Microsoft, “Event Tracing,” 31 5 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal. [Accessed 3 September 2019].

[2] Microsoft, “Defining Malware: FAQ,” 01 04 2009. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10). [Accessed 3 September 2019].

[3] AVTest, “AVTest Security Report 2018/2019,” 2019. [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf. [Accessed 3 September 2019].

[4] K. Chumachenko, *Machine Learning methos for malware detection and classification,* kaakkois-suomen ammattikorkeakoulu, 2017.

[5] P. T. C. W. a. T. H. Konrad Rieck, “Automatic Analysis of Malware Behavior using machine learning,” *Journal of Computer Security,* 2011.

[6] A. Dhiman, *Malware detection and classification using machine learning techniques,* Department of Computer Science and Engineering, Indian Institute of Technology, 2018.

[7] S. Haykin, Neural networks: a comprehensive foundation., Prentice Hall, 19994.

[8] J. W. Ronan Collobert, “A unified architecture for natural language processing: deep neural networks with multitask learning,” no. ICML '08 Proceedings of the 25th international conference on Machine learning, pp. 160 - 167, 2008.

[9] S. Lawrence, “Face recognition: A convolutional neural-network approach.,” *IEEE transactions on neural networks,* vol. 8, no. 1, pp. 98 - 113, 1997.

[10] T. Tian, “Deeptest: Automated testing of deep-neural-network-driven autonomous cars.,” *Proceedings of the 40th international conference on software engineering.,* pp. 303 - 314, 2018.

[11] M. Kantarcioglu, *Introduction to malware,* The University of Texas at Dalas.

[12] Y. Nativ, “theZoo - A Live Malware Repository,” 2014. [Online]. Available: https://github.com/ytisf/theZoo. [Accessed 12 July 2019].

[13] VirtualBox, “VirtualBox home page,” VirtualBox, [Online]. Available: https://www.virtualbox.org/. [Accessed 22 June 2019].

[14] Microsoft, “Perfview,” [Online]. Available: https://github.com/Microsoft/perfview. [Accessed 20 July 2019].

[15] T. Mikolov, “Recurrent neural network based language model,” *Eleventh annual conference of the international speech communication association,* 2010.

[16] S. a. J. S. Hochreiter, “Long short-term memory.,” *Neural computation,* vol. 9, no. 8, pp. 1735 - 1780, 1997.

[17] A. S. F. B. Has¸im Sak, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling,” Google, 2014.

[18] C. Olah, “Understanding LSTM Networks,” 27 August 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 29 June 2019].

[19] Microsoft, “A Tour of the C# Language,” 04 05 2019. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Accessed 05 September 2019].

[20] Microsoft, “.NET,” [Online]. Available: https://dotnet.microsoft.com/. [Accessed 28 June 2019].

[21] Microsoft, “Visual Studio,” [Online]. Available: https://visualstudio.microsoft.com/. [Accessed 28 June 2019].

[22] J. Newton-King, “Json.NET,” [Online]. Available: https://www.newtonsoft.com/json. [Accessed 20 July 2019].

[23] Python, “Python,” [Online]. Available: https://www.python.org/. [Accessed 05 September 2019].

[24] Anaconda, “Anaconda Distribution,” [Online]. Available: https://www.anaconda.com/distribution/. [Accessed 1 August 2019].

[25] NumPy, “NumPy,” [Online]. Available: https://numpy.org/. [Accessed 01 August 2019].

[26] Tensorflow, “TensorFlow,” [Online]. Available: https://www.tensorflow.org/. [Accessed 1 August 2019].

[27] JetBrains, “PyCharm,” [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed 1 August 2019].

[28] Wikipedia, “Artificial neural network,” [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed 3 September 2019].

[29] ResearchGate, “Structure of a memory cell in long short-term memory (LSTM)-RNN.,” [Online]. Available: https://www.researchgate.net/figure/Structure-of-a-memory-cell-in-long-short-term-memory-LSTM-RNN_fig4_318453428. [Accessed 05 September 2019].