BACHELOR THESIS
COMPUTING SCIENCE

RADBOUD UNIVERSITY

DGA-based Malware Detection Using DistilBERT

Author:
Abdulkarim Abdulkadir
s4840933
abdulkarim.abdulkadir@ru.nl

First supervisor:
Assistant Professor Katharina
Kohls
kkohls@cs.ru.nl

[Second supervisor:]
title, name
e-mail adress

Second assessor:
title, name
e-mail adress

January 12, 2022

Abstract

The abstract of your thesis is a brief description of the research hypothesis, scientific context, motivation, and results. The preferred size of an abstract is one paragraph or one page of text.

Contents

Chapter 1

Introduction

As the purpose of our digital devices continue to expand, the importance of protecting our sensitive data has become more crucial. The pandemic has proven how much we rely on these devices. The value of our digital resources has increased, which makes this an appealing target for exploitation.

A malicious software, or malware, is a software that target our digital resources. The emerging presence of malware have created different types of malware and new attack methods to our computer systems. According to the AV-Test report, in 2021 the number of malware totaled around 1320 million, a 13 times increase of the report in 2012, which was around 100 million [6].

Most types of modern malware communicate with external servers of the attackers using different network protocols. DNS (Domain Name Server) is a network protocol used frequently to connect to these externel servers [23]. The malicious external servers have different domain names to make sure to be available to the malwares. An external server having a single domain or a fixed IP address can be blacklisted. This would make the server not accesible anymore. Therefore, external servers create multiple domain names for the malware to connect to. The multiple domain names are created by generating them using a domain generated algorithm (DGA). The malwares are packed with the same DGA that the external server uses, so that it can generate the same domain names. The domain names generated by the servers are registered in advance to secure those domain names. The malware uses Domain Name Server (DNS) services to connect to each generated domain name till it succesfully conncected to the IP address of the malicious external server.

Recently malware that uses DGA are polymorphic. The malware can generate domain names dynamically. One way to do that is by using time information as a seed for the DGA [5]. The polymorphic aspect of the malware improves the concealment and robustness of the malware and brings great challenges to DGA-based malware detection.

Hence, there is a solution needed to defend against DGA-based malware. Traditional detection methods use DNS traffic or domain name language characteristics to extract feature out of the DGA malware. After that machine learning is used to analyze the extracted features and complete the identification and classification of DGA domain names. However, it is difficult to determine the DNS traffic and domain name language characteristics of different types of DGA. Specifically, DGA types that use wordlists to generate domains are difficult to differentiate from benign domains. Therefore, detection schemes based on feature extraction and DNS traffic have a high time and bandwidth cost, and the features that are extracted are not flexible.

More comprehensive tactics are necessary to detect malicious domain names and differentiate them from benign domain names. An improved detection method compared with previous approaches is the detection model based on deep learning. The benefit of deep learning is the automatic extraction of the DGA domain features as well as understanding the context of the domain names.

We propose a nouveau detection model based on deep learning, the Bidirectional Encoder Representations from Transformers (BERT) to detect DGA domains. BERT is a transformer-based machine learning technique which allows for bidirectional training in models. This in contact to previous efforts, which train on a text sequence from left to right or right to left. The second benefit to BERT is that it is already pretrained on different language representation models [11].

The original English-language BERT base model is pretrained from unlabeled data from the BooksCorpus [30] with 800M words and English Wikipedia with 2500M words [3]. In our research we will use a distilled version of BERT, called DistilBERT [21]. This model recuces the size of BERT by 40%, while still retaining 97 % of its language understanding capabilities and being 60% faster. With our DistilBERT model that is pretrained on uncased English words, we are able to detect DGA domains with low-cost, while still surpassing previous detection models.

The paper is structured as follow. In chapter two we will shortly explain what malware is and what kind of malware types there exist. We will describe how botnets use domain generated algorithms to stay online and avoid detection. After that we will describe machine learning, specifically different neural networks techniques, from older neural network techniques to newer ones. We will discuss what kind of problems and shortcomings these older neural networks have compared to the newer ones. Lastly we will introduce a new deep learning model, called the transformer model. As well as unfold the BERT and DistilBERT models that are transformer-based. We will clarify the benefits of these transformer-based models and how it solved several problems of the older neural network techniques.

In chapter three we will show a detailed implementation of our DistilBERT detection model to detect DGA-based malware domains. Thereafter, show the kind of results our DistilBERT model has produced. Finally, we will compare our results with previous results of DGA detection models.

In chapter four we will examine previous research done in DGA detection, while looking at their results and their shortcomings.

In chapter five we will evaluate all our results and discuss the problems our model has and how to improve it. We will conclude by giving any suggestions for future research in this field.

Chapter 2

Preliminaries

This chapter will describe malware, the different types and how it utilizes DGA to perform malicious acts. It will also explain the basics of machine learning and different types of neural networks that are necessary for this research paper.

The term "malware" is coined by blending two words: malicious and software, a software that is malicious in nature. Malware can have multiple purposes, such as cybercriminals using it to extract data from the victims' computer to leverage against them for financial gain. This data can range from financial data to sensitive personal data, like healthcare records, personal emails, passwords and countless other possibilities.

The most common ways victims receive malware is through the internet and email. Malware can penetrate a victims' computer in different ways, such as: surfing malicious websites, viewing malicious ads, downloading infected files, and installing malicious programs or apps. When a malware infects the computer system of a victim, it can end up in a network of infected computers.

2.1 Botnets

A compromised machine that is infected by malware can end up in a network of infected machines (botnets). This machine is a bot in that network, which receives and responds to commands from the command & control server (C & C). The C & C server is controlled and receives commands by a human controller called a botmaster. The botmaster conceals itself by employing a number of proxy machines, called the stepping stones, between it and the C & C server. The life cycle of a botnet can be divided into four phases. Only the first two phases are significant for this research.

The first phase is when the machine (bot) receives the malware and executes the binary. After the machine is infected, this machine (bot) tries to contact the C & C server to announce its presence and communicate with it. This establishment phase is called Rallying. There are two ways that the bot can contact the C & C server. The first way, the bot uses the IP address of the C & C server to contact it. This IP address can be hardcoded into the binary of the bot. The problem with this, is that the IP address can be exposed by reverse engineering the binary. The IP address can also be seeded, where the bot is provided by a list of peers. The second way is that the bot knows the domain name of the C & C server. The domain name will be hardcoded into the bot binary, which also makes it vulnerable to reverse engineering the binary.

2.2 Domain Generated Algorithm

Another way that the malware can connect to the C & C server is by generating a domain name. This is done by using a domain generated algorithm (DGA). Bots can dynamically contact the C & C server using DGA. They attempt to resolve the generated domain names by sending DNS queries to the C & C server until one of the domains resolves. Domains that do not resolve will result in Non-Existent Domain (NXDomain) responses.

Domain names that are generated by DGA are also known as Algorithmically Generated Domains (AGD). The DGA uses a seed that serve as a shared secret between the botmaster and the bot. There are two types of seeds: static seed and dynamic seed. The seed is required for the DGA to calculate the AGDs. The DGA takes the seed value as input to generate pseudo-random strings and append algorithmically TLD (Top Level domains) to the domains, such as *.nl, .com, .org, .edu.* The static seed can be a dictionary of words, random strings that are concatenated, numbers or any other value that the botmaster can come up with. Dynamic seeds change with time, which makes them dynamic. These seeds can be currency exchange rate, daily trending twitter hashtag, weather temperature, and current date and time. The static and dynamic seed elements are then stitched together to generate a pseudo-random string.

The botmaster uses the DGA to generate a large number of domain names for the C & C server. The constant change of domain names for the C & C server is known as Domain-Fluxing. The botmaster tries to register generated domain names in advance in order to reserve those domain names. When the bot receives the malware, the malware queries the pre-registered domain name and resolves the IP address using DNS. Often the botmaster registers the domain name a few hours prior to an attack and disposes of it

within a day. Whenever the bots can not resolve the previous domain name, they query the next set of generated domain names until it finds a domain that does work.

The DGA and constant domain-fluxing of the C & C server provides agility and resilience to the infrastructure of the botnet. This makes it hard to predict what domain names a bot will try to resolve. On the other hand, analysts will re-engineer DGA by analyzing the malware and understanding how the algorithm works. The difficulty of analyzing DGA is to predict what kind of seed these DGA will use at a specific time. It is also infeasible to report all the domain names that can be generated. Since some DGA use English dictionaries as static seed values, it makes it even harder to distinguish benign domain names from malicious ones.

2.3   Machine Learning

Machine learning has recently been an attractive tool used in security. One way to combat DGA is to use machine learning to classify the structure of the generated domains. There are two machine learning methods: supervised and unsupervised learning. Unsupervised learning uses algorithms to analyze and cluster data, which in our specific case are the domains. These algorithms discover hidden patterns or data groupings, without a need for human intervention. There are three ways to approach unsupervised learning: clustering, association and dimensionality reduction. The domains are divided into clusters to find statistical attributes for each group. To produce a cluster with good generalization capabilities, it can take a lot of time and effort [15]. Supervised learning does not rely on the statistical attributes for each group to classify DGAs. Supervised learning attempts to understand, classify the input and predict the outcome accurately. The relationship is represented as a structure to predict the outputs for specific future inputs.

2.4 Neural Networks

Artificial Neural Networks are artificial systems that are inspired by the biological counterpart. The systems learns in a supervised manner to perform tasks by training on various datasets and examples. These neural networks are composed of multiple node layers: input, hidden and output layers. Each node is connected to another node and has an associated weight w and threshold t. When the threshold t of a specific node is above a certain threshold value, then that node is activated, otherwise no data is passed along to the next layer of the network. This is determined by a specifically used activation function in the network.

The network uses training data to learn and improve the accuracy of the network. This is usually done by backpropagation. Backpropagation is a supervised learning algorithm that computes the difference between the model output and the actual output using gradient descent and the chain rule. It checks if the error is minimized and updates the weights w and biases accordingly. It repeats the process until the error becomes minimum [16].
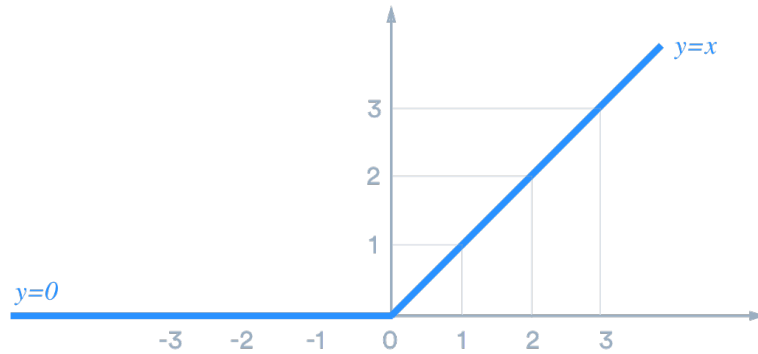
2.5 Activation Functions

The activation functions are functions that determine the output of a neural network. It maps the input value of a neuron to the output value. The function receives the calculated weighted sum of the inputs and the added bias, and then decides if this sum passes through to the next layer or not. There are two types of activation functions: linear activation functions and non-linear activation functions. The linear activation functions are functions that do not update the weighted sum of the input, but instead returns the value directly. A neural networks with multiple layer needs non-linear activation functions, because linear activation functions would make the hidden layer in that network purposeless.

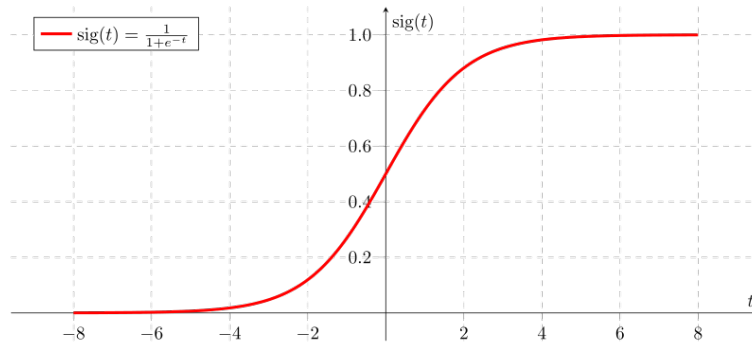### 2.5.1 ReLU Activation Function

ReLU, or Rectified Linear Unit, is a linear activation function. It is linear when the input is positive and 0 when the input is negative. The range of ReLu is $[0, inf)$. The benefit of ReLu is that there is a reduced likelihood of the gradient to vanish 2.6.1, when the gradient is constant. However, the constant gradient results in the network learn faster. Another benefit is the sparsity, as the network has more units in a layer, other activation functions will be processed to describe the output of that network. When the calculated sum value in ReLU is negative, it yields 0. This means there are fewer neurons firing, which makes the network lighter.

The disadvantage is that ReLU tends to blow up, as the range goes to infinity and there is no mechanism to constrain the output when it is positive. Another disadvantage is that if too many activations in the network reach below zero, then the neurons in the network will output zero. This means that the outputs die out, which will prohibit learning. This is called the Dying ReLu problem [18].

2.5.2 Sigmoid Activation Function

The sigmoid activation function is a non-linear activation function that looks like an S-shape. Any small changes in the incoming X value (the calculated sum) will cause the Y value (the output) to change significantly. The range of the function is $(0, 1)$. That means the range is bounded, which means it does not blow up. The disadvantage of the sigmoid activation function are the vanishing gradients 2.6.1.


9

2.5.3 Tanh Activation Function

The tanh activation function resembles the sigmoid function. The difference is that the range of tanh activation function is $(-1, 1)$. Another difference is that the gradients are stronger for tanh than sigmoid. That means the derivatives are steeper. One benefit tanh has over sigmoid, is that it avoids biases in the gradients [27].
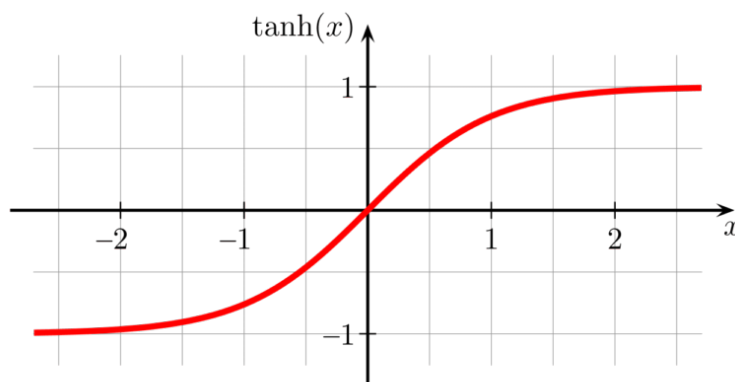
2.6 Recurrent Neural Networks

Recurrent neural networks (RNN), are a type of neural network that uses the output from the previous step and feeds this output as input in the current step. Whereas in traditional neural networks, the network assumes that the inputs and outputs are independent of each other. The cost function or error in RNN can be calculated at any time t. At any time t, the current input is a combination of inputs x_t and x_{t-1}. This makes the neural network recurrent, it has feedback loops at each iteration of the hidden layer. Recurrent neural networks are used for Sequence Modeling. Sequence Modeling is the task to predict future outcomes.

However, there are some drawbacks to RNN. The first drawback is when the sequence is done in order, there is a limit in how much training can be parallelized. The second drawback is that the farther away the relevant points in the sequence are from one another, the harder and slower it is to make connections between them. This drawback is caused by the vanishing gradient problem.
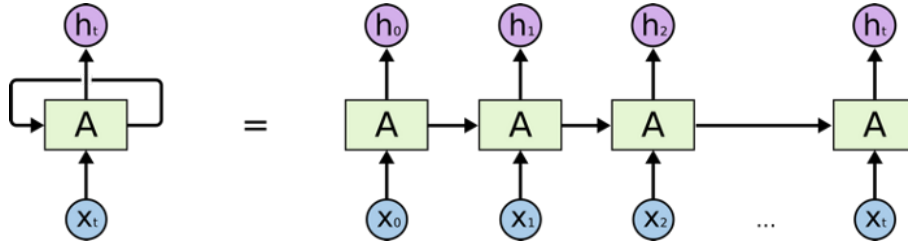
2.6.1 Vanishing Gradient problem

The vanishing gradient problem can be encountered when gradient-based learning methods and backpropagation are used. Adding more layers using non-linear activation functions to the neural network causes the gradients of these loss functions to approach zero. The gradient will be vanishingly small, which in turn prevents the current weight from changing its value. This can lead to the neural network to stop training further. As mentioned before, an activation function like the sigmoid function, squishes a large input space into a value between 0 and 1. The effect of this is that a large change in input would cause a small change in the output. The derivative therefore becomes miniscule. The derivative approaches zero, which causes the gradient of this layer in the network to vanish.

One of the network that suffers from the vanishing gradient problem is a basic recurrent neural network. When the feedback loops occur and the gradient gets lower, it becomes harder for the network to update its weights. The weights of the initial value will not change effectively through the training process, which can lead to inaccuracy in the network.

The solution to the vanishing gradient problem is to use activation functions for the network that are resilient to this problem, such as ReLu. This is because ReLU does not cause a small derivative. Another solution is to use resilient neural networks, such as residual networks [12]. There are also specialized RNNs that are more resistant to this problem. One of these specialized networks is the long short-term memory (LSTM) network.

2.6.2 LSTM

Long short-term memory (LSTM) is a specialized RNN that is capable of learning in long-term dependencies. It is designed to remember information for long periods of time. It does this by adding a forget mechanism. The hidden layer in LSTM is a gated cell. It consists of four layers that interact with each other to produce the output of that cell to pass on to the next hidden layer. LSTM consists of three logistic sigmoid gates and one tanh layer, compared to traditional RNNs that use only a single layer of tanh. These gates are used in order to limit information or pass information through the

cell. The inputs of LSTM go through the input, forget and output gate. The forget gate decides to remember or to skip inputs from the previous hidden states. The mechanism of the forget gate mostly solves the vanishing gradient problem. The input gate decides what new information has to be added to the cell. Finally the output gate decides which new or old information has to be passed to the next hidden layer by using the memory state that is updated by the input and forget gate.
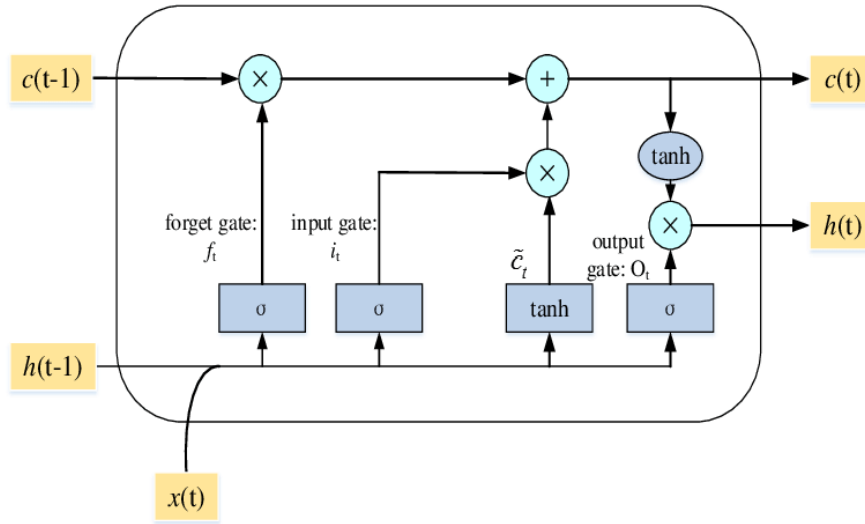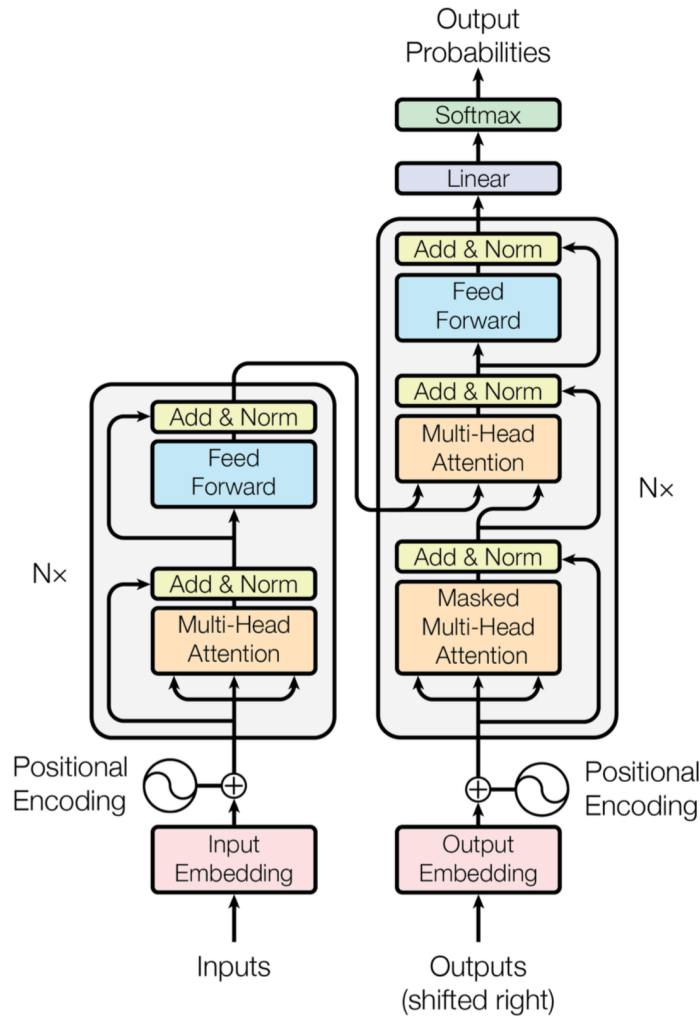

While LSTM overcomes the vanishing gradients in the network, it inherits some problems of RNNs, such as: no parallelization, as there is a sequential path for the data in the network. While LSTMs have mitigated the vanishing gradient problem, it could not completely get rid of the problem. The data in LSTM has to move from cell to cell in a sequential manner, same as in traditional RNN. This problem inherently lies in the recursion of RNNs. RNNs can be slow to train, but LSTMs are even slower to train, because they are more complex than traditional RNNs.

2.7 Transformers

A transformer is a new network architecture in which the input sequence can be passed in parallel, which can increase the speed drastically. The transformer is first introduced by Vaswani et al. in their paper "*Attention Is All You Need*" [25]. The transformer model is based solely on the attention mechanism. Attention is a mechanism that figures out for each token, how relevant all the other tokens are in a sequence. Attention learns to weigh the relationship of each token in the input sequence to other tokens in the output sequence. The core idea is that the transformer model uses only the part

of the input where the most relevant information is concentrated instead of the entire sequence. The same as a neural network is considered to mimic a human brain, the attention mechanism also tries to implement the action of selectively concentrating on relevant things, while ignoring other not relevant inputs in the neural network. Self attention is similar to attention, but it allows the inputs not to only interact with the outputs, but with other inputs as well. The transformer model that Vaswani et al. proposes in their paper uses multi-headed attention layers. In multi-headed attention, each head in the layer learns attention relationships independently. Attention is constructed as a combination of three matrices, where every value in those matrices are learned.

Output
Probabilities

Softmax

Linear

Add & Norm

Feed
Forward

Add & Norm

Multi-Head
Attention

Nx

Add & Norm

Feed
Forward

Add & Norm

Add & Norm

Multi-Head
Attention

Masked
Multi-Head
Attention

Nx

Positional
Encoding

Positional
Encoding

Input
Embedding

Output
Embedding

Inputs

Outputs
(shifted right)

The transformer architecture that is proposed in the paper, uses a sequence to sequence model [2], that consists of an encoder and a decoder. Before the inputs go into the encoder, the input has to be embedded. Input embedding

13

maps every word to a point in space where similar words or meanings are physically closer to each other in that space. This space is called the embedding space. The embedding space maps a word to a vector. In a sentence the same words can have different meanings. That is why transformer models have positional encoders. These encoders are vectors that give context based on the position of a word in a sentence. Adding positional encodings to a transformer model will result in embeddings of words with context information. The resulting input embedding with context information then goes into the encoder. This encoder contains a stack of multi-headed attention and a feed-forward neural network [22]. These feed-forward neural networks are used to transform the attention vectors to make it digestible for the next encoder or decoder block. The decoder is similar to the encoder, only it has an additional multi-head attention block. Transformers are used for sequence to sequence tasks like NLP or machine translation. It is also used as autoencoding language modeling, such as masked language modelling. One of the models that is trained on masked language modeling, created by Google, is called BERT.

2.7.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a large transformer masked language model. It is mostly used for pre-training natural language processing (NLP). The main innovation of this technique is applying bidirectional training on a Transformer model. In contrast, other efforts looked at it in a single direction, from left to right or right to left.

BERT uses the encoder mechanism to generate a language model. Bert's encoder reads the entire sequence of words at once. Therefore, it ensures that the model learns the context of a word from all of its surroundings, making it bidirectional. BERT are pretrained on two tasks: language modelling (LM) and next sequence prediction (NSP). It uses Masked LM (MLM) for pretraining language modelling. This is done by replacing 15% of any sequence with a [MASK] token. The model tries to predict the original value of the masked words, using the context provided by the other non-masked words in a sequence. Masked language models are a type of contextual word embedding models. Contextual word embedding gives a model different representation for different sentences.

The next task of the training process, BERT uses next sentence prediction to better understand the relationship between two sentences. While the model is training it receives sentences as input pairs and it learns to predict if the second sentence in the input pair is also the next sentence that was in the original document. BERT separates sentences with a special [SEP] token. Then the model is fed with two input sentences at a time. During training, 50% of the time the second sentence is the subsequent sentence in the original document, while in the other 50% of the time it is a random sentence from the full corpus. The assumption being that the random sentence will be disconnected from the first sentence.
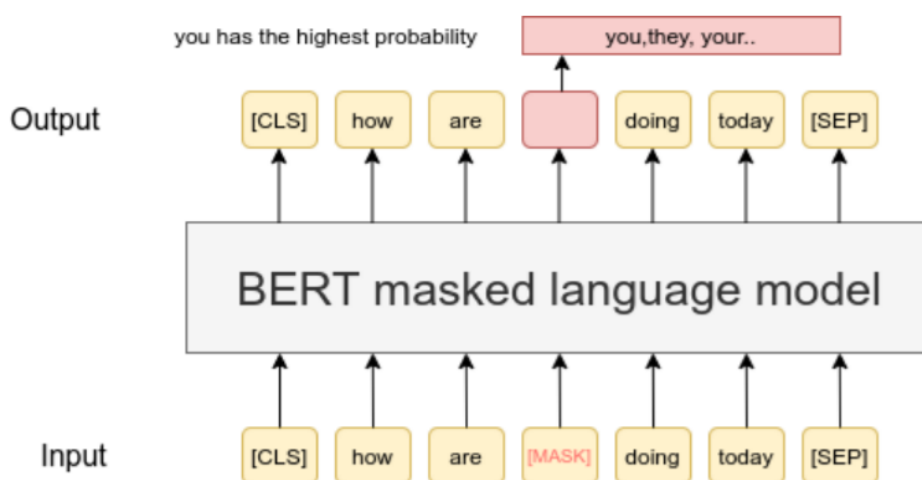
All together the input in BERT is processed in these six steps:

1. Each sentence sequence is separated by a [SEP] token. It is placed at the end of each sentence.

2. Every sentence will replace 15% of its words with a [MASK] token.

3. At the beginning of the first sentence a special [CLS] token will be inserted.

4. Every other word in a sentence is transformed into an embedded token.

5. A sentence embedding is added to each token. They are similar in concept as token embeddings, but these embeddings are used to indicate if sentence A or sentence B is added to each token.

6. At last a positional embedding gets added to every token to indicate their position in the sequence.

Chapter 3

Research

In this chapter we will explain about all the technical details of our Distil-BERT model. First, we will justify which libraries and framework we used for our model. Second, we will showcase how we have implemented the libraries in python to create our model. Afterwards, we will analyze our data set, consisting of malicious domains and benign domains. Lastly, we will demonstrate the results of our model. The python code is accessible in A.

3.1 System architecture

To build and train our DistilBERT model, we used the ktrain library [19]. Ktrain is a lightweight open source wrapper for the deep learning library TensorFlow Keras [10]. It helps to build, train and deploy neural networks in a more accessible and easier way. Ktrain allows you to easily estimate an optimal learning rate for your model given a learning rate finder.

For data analysis on our data set we use the open source scikit-learn library [7]. It is a simple and efficient tool to predict and analyze data, built on NumPy, SciPy and matplotlib.

3.2 Datasets

This paper uses two open data sets to make the research reproducible. The Tranco 1 million domains [20] are used for benign, not DGA, domains. Tranco is a research-oriented top sites ranking data set, that is hardened against manipulation. Most researchers [4][17][13][24] rely on popularity rankings such as the Alexa top 1 million domain list. However, the Tranco paper [20] finds out that it is trivial for an adversary to manipulate the composition of these lists. The list of Alexa top 1 million can be altered by as little as a single HTTP request by adversaries. Therefore, the Tranco paper comes up with a 1 million domain list that is hardened against these

manipulations. This is the list we use for our DGA domain detector. We only use a fourth of the domains in the 1 million Tranco list, totalling 200000 benign domains.

For the DGA malicious domains, we use the UMUDGA data set [28]. UMUDGA is a dataset for profiling DGA-based botnet. It contains 40 notorious distinct malware variants generated domain lists. For our model, we have opted out for 5000 domain list per malware variant. Thus, we have 200000 DGA domains. In total we have 400000 domains in our dataset, with a ratio of 50/50 between benign and dga domains.

3.3 Detector

Our detector is predicting

3.4 Metrics for validation

The measure our model performance, we calculate multiple metrics that are used commonly in machine learning research. To illustrate the metrics, we will use the following abbreviations: true positive (TP), true negative (TN), false positive (FP), false negative (FN), true positive rate (TPR) and false positive rate (FPR). The metrics are calculated as follows:

$$Precision = \frac{\sum TP}{\sum TP + \sum FP}$$

The precision metrics measures the ratio of correct positively labeled instances to all positively labeled instances.

$$Recall = \frac{\sum TP}{\sum TP + \sum FN}$$

The recall metrics measures the ratio of correct positively labeled instances to all instances that should have been labeled positive.

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

F_1 is the harmonic mean of Precision and Recall.

$$TPR = \frac{\sum TP}{\sum TP + \sum FN}$$

True Positive Rate (TPR) is just a synonym for Recall.

$$FPR = \frac{\sum FP}{\sum FP + \sum TN}$$

False Positive Rate (FPR) determines the rate of incorrectly identified labeled instances.

The receiving operating characteristics (ROC) curve is an evaluation metric for binary classification problems that plots TPR and FPR at various threshold values. It essentialy seperates the 'signal' from 'noise'. The area under the ROC curve, called the AUC, of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

3.5   Results

3.6   Comparison

Chapter 4

Related Work

In this chapter we will discuss some of the previous work that has been done to detect DGA domains. There are multiple approaches to research on DGA domains.

One of the first approach to detect DGA domains is by using unsupervised learning. Chang and Lin [8] propose a dynamic way to detect botnets DNS traffic monitoring. First the known benign and malicious domain names are filtered in the DNS traffic. After which the Chinese-Whispers algorithm is applied to the remaining domains to cluster them according to the similarity of the query behaviour. Zhou et al. [29] use a passive DNS dataset to record the information of domain access, consisting of 18 features, to detect Fast-Flux domains using random forest algorithm. With the insight that not resolved DGA domains result in NXDomain responses, Antonakakis et al. [4] classify and cluster the domains with Hidden Markov Models (HMM). Although, because the clustering strategy relies on domain names' structural and lexical features, it is limited to DGA-based C & C only.

Woodbridge et al. [26] is the first to utilize supervised deep learning for DGA detection. A simple implementation of an LSTM is used for non-specific DGA analysis. They show that their LSTM network outperforms unsupervised learning methods such as character-level HMM and random forest models. However, their LSTM model does not have a high score on suppobox or matsnu, the dictionary DGA families. Their research have given other researchers interest in using supervised learning methods to better identifying DGA domains. In a different angle Anderson et al. [1] uses a generative Adversarial Network (GAN) to investigate if the adversarial learning techniques is able to deceive DGA detection.

Tran et al. [24] presents a novel LSTM.MI algorithm that combined both binary and multiclass classification models to improve the cost-effectiveness of the LSTM. They demonstrate that the LSTM.MI algorithm provides an improvement of at least 7 % compared to the original LSTM. Chen et al. [9] proposes a LSTM Property and Quantity Dependent Optimization (LSTM.PQDO) that dynamically optimizes the resampling proportion of the original number and characteristics of the samples. This research results in a better performance compared to earlier models by overcoming the difficulties of unbalanced datasets. Another research done by Lison et al. [17] alters the structure of the LSTM to a bi-directional LSTM layer. The enhancement of the bi-directional LSTM layer results in a F_1 score of 0.971.

Koh et al. [14] are one of the first that utilized deep learning to train their model. They classify domains based on word-level information by combining pre-trained context-sensitive word emberdding with a classifier. The LSTM is trained both on single-DGA and multiple-DGA data. The model outperforms existing techniques on wordlist-based DGA. Highnam et al. [13] research picks up on Koh et al [14] work. By systematically evaluating deep learning, a novel hybrid neural network, called the Bilbo the fibagginfi model, is created that consists of a model which uses a convolutional neural network (CNN) and a LSTM network in parallel. This CNN+LTSM combo network is the most consistant in performance in terms of AUC, F_1 score and accuracy compared to previous work.

4.1   Contribution

This thesis will furthur contribute to detecting DGA domains using deep learning. We propose a novel approach to detecting DGA domains, by utilizing the newly developed transformer models to pre-train our model that bidirectionally classifies context-senstive word embedding of DGA domains. We use an alternative distilled version of the Bidirectional Encoder Representations from Transformers model (BERT), called DistilBERT [21]. It is a smaller, faster and lighter BERT model, by reducing a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster. We are one of the first that have used the BERT model to detect and classify DGA domains.

Chapter 5

Conclusions

In this chapter you present all conclusions that can be drawn from the preceding chapters. It should not introduce new experiments, theories, investigations, etc.: these should have been written down earlier in the thesis. Therefore, conclusions can be brief and to the point.

Bibliography

[1] Hyrum S. Anderson, Jonathan Woodbridge, and Bobby Filar. Deep-dga: Adversarially-tuned domain generation and detection. *CoRR*, abs/1610.01969, 2016.

[2] Andrew Ng - Stanford University. Sequence to sequence model. URL: `https://cs230.stanford.edu/files/C5M3.pdf`.

[3] Issa Annamoradnejad. Colbert: Using BERT sentence embedding for humor detection. *CoRR*, abs/2004.12765, 2020.

[4] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX Security Symposium*, Security'12, page 24, USA, 2012. USENIX Association.

[5] Pieter Arntz. Explained: Domain generating algorithm, Dec 2016.

[6] AVTest. Malware Statistics. `https://www.av-test.org/en/statistics/malware/`, 2021. [Online; accessed 26-Dec-2021].

[7] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[8] Chun De Chang and Hui Tang Lin. On similarities of string and query sequence for dga botnet detection. In *32nd International Conference on Information Networking, ICOIN 2018*, International Conference on Information Networking, pages 104–109, United States, April 2018. IEEE Computer Society.

[9] Yijing Chen, Bo Pang, Guolin Shao, Guozhu Wen, and Xingshu Chen. Dga-based botnet detection toward imbalanced multiclass learning. *Tsinghua Science and Technology*, 26(4):387–402, 2021.

[10] François Chollet et al. Keras. `https://keras.io`, 2015.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[13] Kate Highnam, Domenic Puzio, Song Luo, and Nicholas R. Jennings. Real-time detection of dictionary DGA network traffic using deep learning. *CoRR*, abs/2003.12805, 2020.

[14] Joewie J. Koh and Barton Rhodes. Inline detection of domain generation algorithms with context-sensitive word embeddings. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2966–2971, 2018.

[15] S. Krishnan, F. Monrose, and J. Mchugh. Crossing the threshold: Detecting network malfeasance via sequential hypothesis testing. *43 Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12, 2013.

[16] Claude Lemarechal. *Cauchy and the Gradient Method*, 2010. `https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf`.

[17] Pierre Lison and Vasileios Mavroeidis. Automatic detection of malware-generated domains with recurrent neural models. *CoRR*, abs/1709.07102, 2017.

[18] Lu Lu. Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, Jun 2020.

[19] Arun S. Maiya. ktrain: A low-code library for augmented machine learning. *arXiv preprint arXiv:2004.10703*, 2020.

[20] Victor Le Pochat, Tom van Goethem, and Wouter Joosen. Rigging research results by manipulating top websites rankings. *CoRR*, abs/1806.01156, 2018.

[21] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[22] Murat Sazli. A brief review of feed-forward neural networks. *Communications, Faculty Of Science, University of Ankara*, 50:11–17, 01 2006.

[23] Symantec. Internet Security Threat Report Volume 24. `https://docs.broadcom.com/doc/istr-24-2019-en`, 2019. [Online; accessed 26-Dec-2021].

[24] Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran, and Linh Giang Nguyen. A lstm based framework for handling multiclass imbalance in dga botnet detection. *Neurocomputing*, 275:2401–2413, 2018.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[26] Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja, and Daniel Grant. Predicting domain generation algorithms with long short-term memory networks. *CoRR*, abs/1611.00791, 2016.

[27] Leon Bottou Yann LeCun, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. *Image Processing Research Department AT&T Labs*, pages 1–12, 1998.

[28] Mattia Zago, Manuel Pérez, and Gregorio Martinez Perez. Umudga: a dataset for profiling dga-based botnet. *Computers & Security*, 92:101719, 05 2020.

[29] Yonglin Zhou, Qingshan Li, Qidi Miao, and Kangbin Yim. Dga-based botnet detection using dns traffic. *J. Internet Serv. Inf. Secur.*, 3:116–123, 2013.

[30] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, abs/1506.06724, 2015.

Appendix A

Appendix

Appendices are *optional* chapters in which you cover additional material that is required to support your hypothesis, experiments, measurements, conclusions, etc. that would otherwise clutter the presentation of your research.