University of Arkansas, Fayetteville

ScholarWorks@UARK

Graduate Theses and Dissertations

8-2022

Effective Knowledge Graph Aggregation for Malware-Related Cybersecurity Text

Phillip Ryan Boudreau
University of Arkansas, Fayetteville

Follow this and additional works at: https://scholarworks.uark.edu/etd

Part of the Information Security Commons, and the Programming Languages and Compilers Commons

Citation

Boudreau, P. R. (2022). Effective Knowledge Graph Aggregation for Malware-Related Cybersecurity Text. *Graduate Theses and Dissertations* Retrieved from https://scholarworks.uark.edu/etd/4604

This Thesis is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu.

Effective Knowledge Graph Aggregation for Malware-Related Cybersecurity Text

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Phillip Boudreau
University of Arkansas
Bachelor of Science in Computer Science, 2020

August 2022
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

____

Qinghua Li, Ph.D.
Committee Chair

________________________________ ________________________________

Brajendra Panda, Ph.D.
Committee Member

Dale Thompson, Ph.D.
Committee Member

Abstract

With the rate at which malware spreads in the modern age, it is extremely important that cyber security analysts are able to extract relevant information pertaining to new and active threats in a timely and effective manner. Having to manually read through articles and blog posts on the internet is time consuming and usually involves sifting through much repeated information. Knowledge graphs, a structured representation of relationship information, are an effective way to visually condense information presented in large amounts of unstructured text for human readers. Thusly, they are useful for sifting through the abundance of cyber security information that is released through web-based security articles and blogs. This paper presents a pipeline for extracting these relationships using supervised deep learning with the recent state-of-the-art transformer-based neural architectures for sequence processing tasks. To this end, a corpus of text from a range of prominent cybersecurity-focused media outlets was manually annotated. An algorithm is also presented that keeps potentially redundant relationships from being added to an existing knowledge graph, using a cosine-similarity metric on pre-trained word embeddings.

Acknowledgements

I would like to thank my advisor, Dr. Qinghua Li, for providing me with the guidance needed to see this project through.

I would also like to show my appreciation to all the committee members, who have given up their time to help finalize this work.

Lastly, I would like to thank my family, especially my parents, for providing support and encouragement throughout the investigation of this topic.

This work is supported in part by the National Science Foundation under Award #1751255.

Table of Contents

Chapter 1 - Introduction....................1

Chapter 2 - Background....................4

2.1 – Deep Learning....................4

2.2 – Supervised Learning....................5

2.3 – Named Entity Recognition....................5

2.4 – Relation Extraction....................6

2.5 – Word Embeddings....................6

2.6 – Word2Vec....................7

2.7 – Recurrent Neural Networks....................9

2.8 – Long Short-Term Memory Networks....................12

2.9 – Gated Recurrent Units....................13

2.10 – Transformers....................14

2.11 – Self-Attention....................17

Chapter 3 – Related Work....................21

3.1 – Overview....................21

3.2 – Knowledge Graphs in Cyber Security....................21

3.3 – NER....................22

3.4 – Relation Extraction....................22

Chapter 4 - Pipeline....................24

4.1 – High-level Overview....................24

4.2 – NER Pipeline....................27

4.3 – Relation Extraction Pipeline....................30

4.4 – Knowledge Graph Merging....................31

Chapter 5 – Evaluation Results....................35

5.1 – Dataset....................35

5.2 – NER results....................37

5.3 – Relation Extraction Results....................39

5.4 – Knowledge Graph Merging Results....................43

Chapter 6 – Conclusion and Future Work....................49

Bibliography....................51

List of Figures

Figure 1: a diagram of the recurrent neural network architecture unrolled over time....10

Figure 2: Transformer encoder-decoder structure, image sourced from [5]....15

Figure 3: High level overview of workflow pipeline....25

Figure 4: Step-by-step example of shift-reduction parsing on an input sentence of “Fake AV is a trojan”....28

Figure 5: Embed, Encode, Reduce, Predict framework process outline....29

Figure 6: High-level overview of relation extraction process....30

Figure 7: Separate text sequences labelled by the NER component....38

Figure 8A: Relations extracted from 10 articles about the ZuoRAT malware strain (redundancies filtered out)....42

Figure 8B: Relations extracted from ZuoRAT articles (redundancies not filtered out....43

Figure 9A: Knowledge graph for article X concerning recent malware Emotet generated without redundancy removal....44

Figure 9B: Knowledge graph also for article X, this time generated with redundancies filtered out....45

Figure 9C: Knowledge graph for another article, Y, concerning recent malware Emotet generated without redundancy filtering....45

Figure 9D: Knowledge graph for article Y generated with redundancy filtering....46

Figure 9E: Knowledge graph for merged articles X & Y used in Figures 11A & 11C without redundancy removals....46

Figure 9F: Knowledge graph for merged articles X & Y used in Figures 11A & 11C with redundancy filtering....47

List of Tables

Table 1: Counts for different entity types in labelled corpus..36

Table 2: Counts for different entity types in labelled corpus..37

Table 3: Scoring metrics and results for NER evaluation...37

Table 4: Scoring metrics and results for relation extraction evaluation.......................................39

Chapter 1 - Introduction

For cyber security analysts, the importance of keeping up to date with information being reported in cybersecurity-related news articles, blogs, advisories, forums, and databases cannot be overstated, especially with the constant advent of new and increasingly vulnerable technologies such as the Internet of Things (IoT). That is because many of them rely on these resources as a way to stay informed about things like vulnerabilities that may (and probably will) affect their systems, which patches to prioritize, and new threats to look out for. Large public repositories like the National Vulnerability Database (NVD) [16] are also updated from information extracted from such immediate sources. As such, maximizing the speed at which information that can be extracted from these security-related media sources is paramount.

The data that exists for security analysts to find is also extremely fragmented across the web, with different resources often reporting on different aspects of security-related news – all of which may be of interest to the analyst at hand. This is why it is more important than ever to be able to aggregate all of this information effectively, combining the data that is reported on by different sources so that the analyst does not have to spend copious amounts of time pouring over all the sources in search of a specific piece of information. It may also be the case that, while searching for these specific snippets of information, one may have to take in and mentally filter out a significant amount of information that has been repeated between sources.

In this work, we aim to develop an automated approach to extract cybersecurity information from multiple sources and merge them to remove redundancy. Specifically, transformer-based encoder models (RoBERTa) are leveraged along with transfer learning to generate an information extraction pipeline for cybersecurity concepts consisting of Named Entity Recognition (NER), relation extraction, then knowledge graph generation and merging.

Semantic triples are extracted from unstructured cybersecurity text to form knowledge graphs that represent the relationships present amongst different classes of entities. A similarity measure is then defined for these semantic triples so that unnecessarily repeated information can be filtered out from the structure, saving time for analysts.

Many previous works involving these information extraction tasks are focused on feature-based models, which induce significant costs in terms of labor as well as domain knowledge. Complex feature engineering is required to describe the different properties of entities, domain knowledge, entity context, and linguistic characteristics [15]. There is also a lengthy period of trial and error involved with the process, and many feature engineering techniques are reliant on lookup tables to identify known entities [17] which are laborious to build and maintain because of the rate at which information evolves in the cybersecurity field. With the speed at which cyber security-centric information is released by different outlets on the web nowadays, manual feature extraction is simply not a viable option since the features may need to evolve over time. Only recently, neural network-based approaches have started to see a significant surge in the amount of attention received regarding cyber security information extraction. Because neural networks are capable of learning useful non-linear combinations of features, they allow researchers to sidestep the laborious process of feature generation. However, in their vanilla form, deep neural networks are not able to capture the complex dependencies involved in interpreting context-sensitive sequential data. This is why a specific class of neural architecture referred to as the Recurrent Neural Network (RNN), which is designed to work nicely with sequential data such as time series or natural language, has been leveraged extensively in related works. Specifically, more specialized versions of the already specialized RNN architecture such as the Long Short-Term Memory (LSTM) and Gated Recurrent Unit

(GRU) networks have been used as they address the primary drawbacks of the basic RNN architecture.

Only very recently has a new architecture of neural network been developed to rival the performance of architectures like the LSTM network, known as the transformer. It is an encoder-decoder based model that was originally designed to be used for language translation tasks, but it has since shown enormous capability on a wide range of other sequence-to-sequence tasks such as text generation [17], text summarization [19], part-of-speech tagging [20], Named Entity Recognition (NER) [21], as well as even computer vision focused tasks such as image segmentation [18]. This thesis leverages this recent advance for tackling the considered problem.

This thesis' contribution is summarized as follows: First, a dataset for NER and relation extraction in malware-focused cyber security text from news articles and blogs is assembled and manually annotated. Second, transformer-based deep learning models for both the NER and relation extraction are trained. Third, an algorithm for selectively adding relationships to the generated knowledge graphs so as to prevent repeated information is presented.

This thesis is organized as follows. Chapter 2 focuses on some background information that is relevant to the paper. Chapter 3 discusses related works in the literature. Chapter 4 focuses on the results and conclusions for this work. Chapter 4 gives an overview of the data processing pipeline. Chapter 5 gives some detail about the evaluation and results of the methods presented. Finally, Chapter 6 discusses the conclusions and some potential future work.

Chapter 2 - Background

2.1 - Deep Learning

In recent years, machine learning has become ubiquitous in research as well as more and more pivotal to the way many different disciplines approach problems such as image classification, recommendation systems, information retrieval, social network analysis, and so on. Among the sea of machine learning algorithms available, deep learning has seen significant attention due to its increased practicality. This increased practicality comes from the wide amount of easily accessible data as well as significant advancements in different computing hardware technologies among recent years. Ever since the mid-2000s when deep learning was really beginning to take its foothold in the modern industrial workflow due to this newfound computational feasibility, it has been making considerable impacts to a wide array of research fields such as Data Science, Computer Vision, and Natural Language Processing (NLP). For NLP in particular, many researchers are interested in extracting semantic information from unstructured text without having to parse it via methods involving human interaction. This domain of research includes many different types of information, though most notably for this work, Named Entity Recognition (NER) and relation extraction.

Before the advent of deep learning, the viability of machine learning algorithms lied heavily on the effectiveness of the data representation. If the data is not represented in such a way that encodes the necessary information required for the algorithm to learn, then performance can suffer greatly. Deep learning algorithms perform this feature extraction in an automated fashion, saving a large amount of time and work for researchers. These deep neural networks are able to form a layered representation of the feature set where low-level features are extracted by

the model in the beginning layers of the network, and higher-level features extracted by the later layers.

2.2 - Supervised Learning

Supervised learning is a technique used widely within the discipline of machine learning, especially deep learning. In the case of deep learning, this technique involves using labelled data to train a multi-layered perceptron (MLP) model to generate outputs that correspond with the labelled examples. More precisely, given a set of input vectors $X = \{x_1, x_2, \ldots, x_n\}$ and a set of corresponding output vectors $Y = \{y_1, y_2, \ldots, y_n\}$ where y_i is the corresponding correct output for input x_i, the model learns to generate some output $\hat{y}_i = f(x_i)$ such that the value of a chosen loss function $L(y_i, \hat{y}_i)$ is minimized. At each step, the network will use some backpropagation method, usually gradient descent, to tune its parameters and find an effective local minimum on the loss function (ideally the *global* minimum, but this is often not achievable in practice).

2.3 – Named Entity Recognition

Named Entity Recognition (NER) is the process of identifying which pieces of text correspond to a given class of entity. For example, in the text "John is the owner of an MX10 speed bike", someone may be interested in classifying "John" as a person, and "MX10 speed bike" as a vehicle. More formally, if you have some text $T = \{t_1, t_2, \ldots, t_n\}$ represented as an ordered sequence of tokens, then named entity recognition is the act of identifying which contiguous subsequences $s = \{t_i, t_{i+1}, \ldots, t_j\}$, $1 \leq i \leq j \leq n$ actually refer to some concrete instance of an entity. This is useful for several reasons, such as identifying the prevalence of certain product types mentioned in a dataset. The approaches toward this problem are usually divided into two categories: rule-based and machine learning based [1]. In this work, we develop

an annotated corpus for NER and relation extraction in the cybersecurity domain. In addition, a machine learning language model is trained for this task that deals specifically with cybersecurity-related entity types.

2.4 - Relation Extraction

Relation extraction is somewhat of a downstream task from NER, as it typically involves taking the entities extracted from the text via NER techniques and determining the relationship between them. For example, in the previous example sentence of “John is the owner of an MX10 speed bike”, where “John” is of entity type person and “MX10 speed bike” is of entity type vehicle, it may be of interest to be able to identify the relationship between John and the speed bike as being “owns”. The typical human-friendly representation of these relationships is via a structure known as the knowledge graph. In this representation, a relation is captured as a semantic triple $r = (e_1, p, e_2)$ where p is the predicate (i.e., relationship), e_1 is the subject entity, and e_2 is the object entity. These triples form a directed graph where the subject and object nodes are connected by an edge (the predicate). In this work, a dataset and pipeline are developed for supervised relation extraction among the entities identified by the NER component. Knowledge graphs are then generated from these relations and merged along with a similarity score metric between triples to reduce redundancies in the graphs, making it easier for analysts to absorb larger amounts of information more rapidly.

2.5 - Word Embeddings

Deep learning models learn to associate inputs in a particular vector space to outputs in another vector space. This means that, when training deep learning models on tasks involving unstructured text, it is paramount that we can come up with a meaningful vector representation

for each of the different tokens that make up a document. These vectors can then be used in a myriad of downstream NLP tasks, such as text categorization [9], parsing [10], information retrieval [11], and named entity recognition [12].

2.6 - Word2Vec

Somewhat surprisingly it turns out that neural networks can be leveraged to solve the problem of generating these multi-dimensional vector representations for other downstream deep learning tasks. This is significant because having to develop a feature set to effectively represent the words manually is non-trivial and requires much time to be spent researching the efficacy of different possible features. One such neural network-based method is Word2Vec [13] which leverages shallow neural networks consisting of a single hidden layer to generate these vector representations.

There are two main flavors when it comes to the way Word2Vec structures its inputs and outputs. The first is a method known as Continuous Bag-of-Words (CBOW), which formulates the problem as a many-to-one mapping problem. The idea is that, given a number of surrounding context words for a piece of text (the number of which varies by application), we can train the network to generate a target word given that context. For example, given the sentence "Neil Armstrong walked on the moon", and a CBOW span length of 5, we would split the sentence up into 4-tuples, where the first two elements are the two preceding token encodings and the last two are the two following token encodings for a given center token. With a CBOW span length of 5, the previous example sentence would be split up into a set of tuples:

$$X = \{(x_{Neil}, x_{Armstrong}, x_{on}, x_{the}), (x_{Armstrong}, x_{walked}, x_{the}, x_{moon})\}$$

The starting encoding scheme for these word vectors can just be a one-hot encoding, where each word in the corpus gets a unique vector component that corresponds to it – so each word will have all 0s as every component except at the unique index that identifies it. This means that the dimensionality of the encoding will be equivalent to the number of unique words in the corpus. The network is then trained to predict whatever word is surrounded by the context. So, for the first tuple above which consists of the encodings for “Neil”, “Armstrong”, “on”, and “the”, the network is trained to predict that the output for the sum of those context encoding vectors should represent “walked”.

The second task formulation for this problem is referred to as the skip-gram model. This approach is essentially equivalent the CBOW formulation but inverted. That is, instead of supplying surrounding context words and trying to predict the middle word, the model is supplied the middle word and is trained to predict the surrounding context words. Mikolov et al. [13] found in practice that the CBOW formulation trains faster and better represents more frequently appearing words while the skip-gram approach works better with smaller datasets and is more capable of effectively representing words that appear less frequently.

These learned embeddings can be used to calculate the similarity between two pieces of text, as both the CBOW and skip-gram formulations leverage the cosine similarity metric in the definitions for the conditional probabilities being maximized. For the skip-gram model, the log probability is maximized:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-l\leq j\leq l, j\neq 0}\log\left(p\left(w_{t+j}\middle|w_t\right)\right)$$

Where T is the number of training samples, l is the size of the context window, and $p(w_{t+j}|w_t)$ represents the conditional probability of predicting one of the surrounding contextual words (w_{t+j}) given a center word (w_t). This conditional probability is then defined as

$$p(w_{out}|w_{in}) = \frac{\exp(\hat{v}_{w_{out}} \cdot v_{w_{in}})}{\sum_{w=1}^{W} \exp(\hat{v}_w \cdot v_{w_{in}})}$$

Where $\cdot$ represents the vector dot product operation, W is the number of words in the vocabulary, and v & $\hat{v}$ represent the target and context vector representations of the respective words. Since the scoring mechanism used here is the dot product and the one-hot encodings of the vectors are all of equal magnitude, maximizing these probabilities results in an output vector space where words that encode similar semantic information are pointing in similar directions. This means that the encodings can be compared via the cosine similarity metric, defined as the normalized dot product between two vectors, to give a value that corresponds to how similar the information represented by the words is. This is because the cosine similarity measure between two vectors is lowest when they are orthogonal, and highest when they are parallel.

2.7 – Recurrent Neural Networks

Deep learning approaches for problems like NER and relation extraction can be somewhat nuanced, because generating features for token representations that accurately encode the semantic information desired in something like unstructured text is highly dependent on the context of the token. That is to say, the meaning held by a token is highly reliant on the tokens that surround it. Because of this, recurrent neural networks (RNNs) were originally viewed as an effective architecture for extracting these token representations. This is because RNNs are a type of neural architecture that was designed to work on sequential data. Their recurrent nature makes

them decently suited for it since each token is passed through the network individually to generate some hidden state, and this hidden state is used as an input back into the same network along with the vector representing the next token. The result is that the outputs at every point should be somewhat “aware” of the information that preceded them. Specifically, RNNs work by taking an input as an ordered sequence of vectors $X = \{x_1, x_2, \ldots, x_n\}$ and learn to map these inputs to a new sequence of “hidden states” $H = \{h_1, h_2, \ldots, h_n\}$ where h_i encodes some information about the ith token, while also considering the context of tokens x_j , $1 \leq j \leq i$.

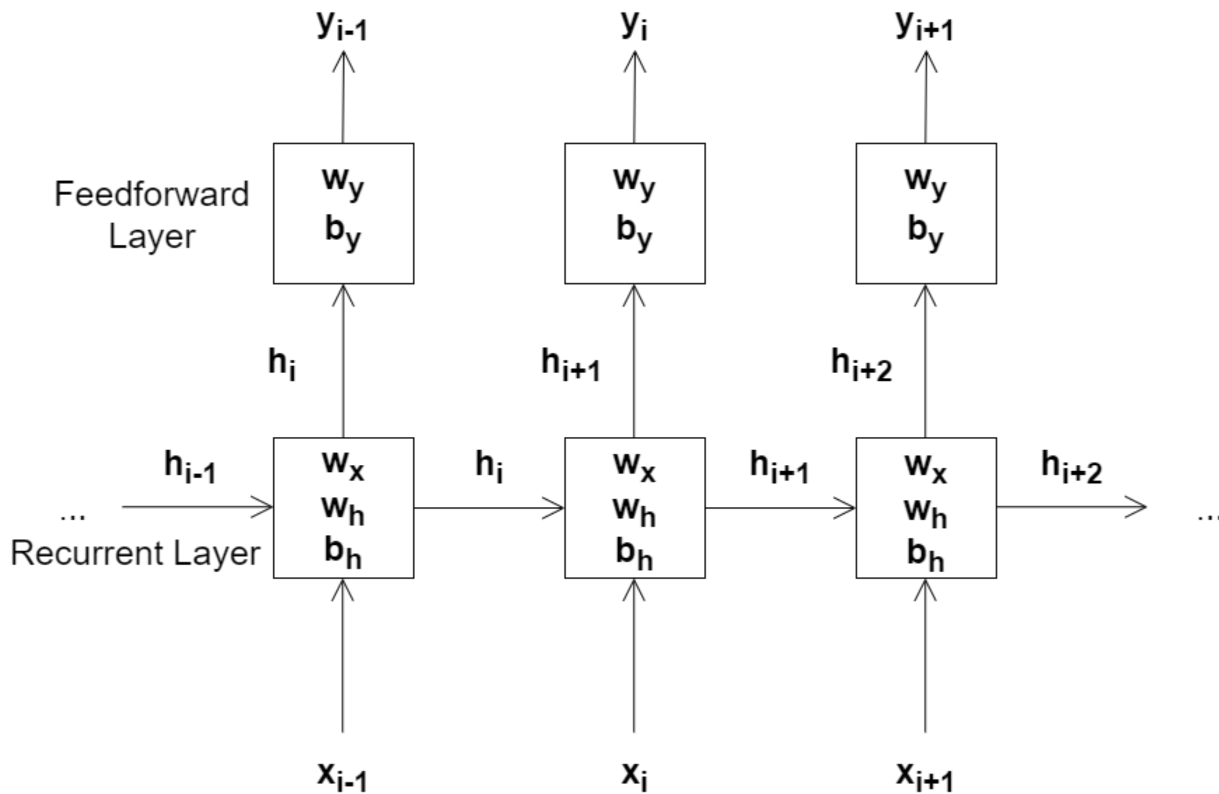

Figure 1: a diagram of the recurrent neural network architecture unrolled over time

Figure 1 shows the recurrent nature of the RNN architecture in an unrolled fashion – that is, in the diagram above, there is only a single recurrent component, it is just that its output is fed

back into itself as input. So, in the diagram, each horizontal arrow represents a forward timestep. In practice, if you have T timesteps that you plan on backpropagating through, the recurrent network is unrolled over time into a standard feedforward network consisting of T duplicates of the original network. h_i represents the output of the hidden layer at timestep i, x_i the input at the ith timestep, and y_i the output at the ith timestep. w_x represents the learned weights for the input layer, w_h represents the weights that are learned by the hidden layer, and b_h the biases for the hidden layer. w_y and b_y are the weights and biases learned for the feedforward output layer. The initial h_0 parameter is usually initialized as all zeros. Biases are needed so that the function approximated by the network can be shifted by an arbitrary constant depending on the nature of the function being approximated by the network. Assuming activation functions f_h and f_y for the hidden and output layers respectively, then the hidden output can be calculated as

$$h_{i+1} = f_h(w_x x_i + w_h h_i + b_h)$$

And the output at the ith timestep is calculated as

$$y_i = f_y(w_y \cdot h_i + b_y)$$

Where $\cdot$ is the dot product. The activation functions f_h and f_y can be any of many that are typically used in neural networks, such as the Sigmoid function ($\frac{1}{1+e^{-x}}$), the ReLu function ($\max(0, x)$), or the Tanh function ($\frac{e^x - e^{-x}}{e^x + e^{-x}}$). The weights learned by the network are shared across the different timesteps. The gradient descent technique used with recurrent neural networks is known as Backpropagation through time (BPTT) [23]. The BPTT algorithm works by treating the network as visualized in Figure 1, unrolling it through time and minimizing an aggregated cost which is calculated as the average of the individual costs for each time step.

While this all sounds good, in practice it has been found that RNNs break down when it comes to learning long range dependencies [3] (instances where the meaning of one token is highly dependent on the meaning of another token that is many positions removed in the input sequence). In particular, these long-range dependencies are hard to learn because of the vanishing and exploding gradient problems. These vanishing/exploding gradients typically come about when doing backpropagation through time for two main reasons. Firstly, the number of timesteps tends to be relatively large. Felix et al. [27] show that even just 10 timesteps are too much for a standard RNN to handle effectively. Since backpropagation is done via unrolling the network according to the number of timesteps, these unrolled network representations tend to be very deep. And since gradients are unstable in deep feedforward networks [24][25], you suffer from the exact same problem in RNNs that work over a significant series of timesteps. Another aspect of RNN architecture that increases their chances of falling victim to vanishing/exploding gradients is that the weights are shared temporally. Since the formulation for the gradient of early layers involves a product of the weights of later layers, using these same weights makes it much more likely that the product (and thusly, the gradient) will either grow or shrink exponentially.

2.8 – Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks [26] are a specific type of RNN architecture that aims to mitigate this issue via the addition of supplementary “gates” in the recurrent cells that attempt to maintain relevant contextual information for longer periods of time. Typically there is a forget gate, input gate, and output gate added. The forget gate consists of a set of weights that are trained to help decide which information should be prioritized in the network’s memory. This means that if its outputs are all high (close to 1), then that is a strong indication that the output for the corresponding current input is very dependent on the history of

the sequence (i.e., context). Conversely, if the outputs from the forget gate are mostly low, that means that the output for the current corresponding input is probably not very dependent on the context of preceding items in the sequence. The input gate takes the current input x_i and the previous hidden state h_{i-1} and creates an encoding (extracts features) for the current input via the tanh neuron, then uses the second sigmoid neuron's output to determine which information from the previous cell state should be remembered. If the forget gate's job is to control which information from the past is forgotten, then the input gate's job is to control which information from the present should be remembered. The output gate determines what the hidden state for the next timestep should be.

A specific type of LSTM, known as the bidirectional LSTM, tends to be used for many of context-sensitive sequence-to-sequence tasks. They are known as "bidirectional" because they will process a sequence of tokens in both the forward and backward directions, allowing context to be considered from both directions in the sequence. For things like text tagging, this is extremely important because it is often the case that the appropriate tag for some token (say, for instance, an entity type) can only be effectively known in relation to the tokens that follow it. For instance, if you had the sentence "Apple, the undisputed tech giant, has just unveiled a new tablet device" – the only way to possibly know that "Apple" is referring to the company and not the fruit is to understand it in the context of the words that follow it – namely the sequence of tokens "the undisputed tech giant" tells us that we are not referring to the fruit.

2.9 – Gated Recurrent Units

Gated Recurrent Units (GRUs) [29] are an architecture of recursive neural network that are conceptually similar to LSTMs but are less expensive to train as they have fewer parameters. While they are cheaper, they still achieve comparable results to more expensive LSTM

implementations [30][31]. They were also designed to tackle the exploding/vanishing gradient problem affecting sequential deep learning models. Just like LSTMs, GRUs have gate units within the recurrent cell but they do not maintain separate cells for memory. There is no longer a separation between the internal memory state and hidden state such as in the LSTM architecture, since GRUs lack the output gate that transforms the final cell memory state into the hidden state for the next timestep. This lack of a need for a cell memory state gives GRUs an advantage over LSTMs in terms of memory requirements. The model for the GRU is also simpler than that of the LSTM, using just two specialized gates versus three. The first gate, the update gate, simply helps the network decide what portion of the information from previous timesteps in the hidden state, as well as the current timestep's input, needs to be preserved moving forward. The reset gate is very similar to the forget gate of the LSTM model, as it is used by the network to determine what portion of previous and current information should be forgotten.

2.10 - Transformers

However, LSTMs and GRUs also have a couple major downfalls – firstly, they are both still expensive to train in general. This is because of the sequential nature of the processing – both architectures require that the first token be processed so that its information can be encoded and used for the encoding of the next token in the sequence. This means that the process is not inherently parallelizable. Secondly, while both are still better than a standard RNN at encoding long-range dependencies, they are still not perfect at it. This is because LSTMs and GRUs have a very limited notion of "attention" built in. This is the idea that, in the context of certain tokens, information represented in the memory state for certain previous tokens is more important than others. Since LSTMs and GRUs alike need to squash the cell memory state / hidden state down to a fixed size, the range of their attention is extremely limited. Transformers mitigate this issue

via the notion of "self-attention" and their non-sequential nature [5]. Whereas RNN architectures like LSTMs and GRUs are inherently recursive, the transformer architecture is not. Instead of processing the text token-by-token by recursively passing its outputs back into itself, the transformer architecture supports processing on the sequence as a whole, all at once. This makes it parallelizable and therefore less expensive to train than equivalent RNN models. This non-recursive property also aids it in overcoming the long-range dependency problem. Since transformers process sequential inputs as a whole and not one-by-one, there is no longer the issue of having to implement a memory module to encode information about surrounding tokens, as their context-sensitive embeddings are all available at once. These are all reasons why the transformer architecture is focused on within this work, as it is considered the state-of-the-art approach to sequence tagging.

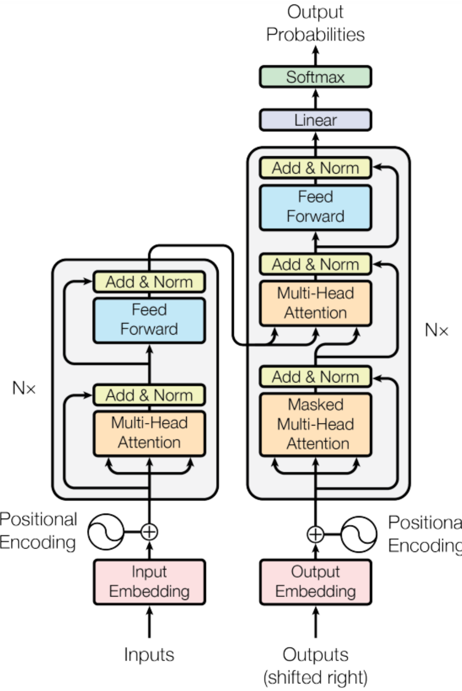

Figure 2: Transformer encoder-decoder structure by Vaswani et al. [5]

In Figure 2, we can see a detailed description of how the transformer encoder-decoder architecture is laid out. It begins with the inputs, which are the original tokens in the sequence each converted to a unique representation, such as one-hot encodings. These token encodings are then passed through an input embedding layer which generates the actual token embeddings, which capture things like semantic similarity between tokens that represent similar things or ideas. Some common techniques for automatic learning of these embeddings are Word2Vec and gloVe. While these word embeddings encode things like semantic similarity well, they still lack positional information that could be useful for models performing context-sensitive sequence-to-sequence tasks. Whereas RNNs and even CNN models are able to make use of the order of the sequence in the way they evaluate the inputs, transformers do not – so some positional information is injected into the embeddings to capture it. From the original paper [5], these positional encodings are calculated as

$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{\left(\frac{\left\lfloor \frac{i}{2} \right\rfloor}{d_{model}}\right)}}\right) \text{ if } i \text{ is even} \\ \cos\left(\frac{pos}{10000^{\left(\frac{\left\lfloor \frac{i}{2} \right\rfloor}{d_{model}}\right)}}\right) \text{ if } i \text{ is odd} \end{cases}$$

where pos is the 0-based positional index, i is the individual dimension, and d_{model} is a model hyperparameter representing the dimensionality of the inputs and outputs – it is chosen to be 512 in the base architecture. This means that each component of the generated positional encoding will correspond to a unique sinusoid. The authors propose that these positional embedding are meaningful because they allow the model to learn to consider relative positions since for any

fixed offset k, PE_{pos+k} is a linear function of PE_{pos} [5]. These positional encodings are then combined with the word embeddings via element-wise addition to create the new position-aware word embeddings. The new position-aware embeddings are then passed through to an encoder module stack where each encoder consists of a multi-head attention component, feedforward component, and two normalization components. Stacking encoders simply means that the output of the first encoder is piped forward as the input to the second encoder and so forth. The number of encoders N_x is a variable hyperparameter of the network and is also typically equal to the number of decoders.

2.11 - Self-Attention

A key component of the model architecture is the multi-head self-attention mechanism. As previously mentioned, the high-level goal of self-attention is to determine which tokens in the input sequence carry more relevant information when trying to predict an output for some other token. Multi-head attention is just the idea of having multiple layers that perform this attention task in parallel. For an input sequence of N_{tok} tokens, the encoder unit takes in a matrix of shape $N_{tok} \times d_{model}$ and each attention head in the multi-head attention portion has output shape $N_{tok} \times \frac{d_{model}}{N_{heads}}$ (where N_{heads} is the number of attention heads). This is because the output of all the individual attention heads will be concatenated across the second axis so that the final output of the multi-head attention component will be back to the original input shape it received, $N_{tok} \times d_{model}$.

The individual attention layers themselves are concerned with learning three sets of weights: W_Q, W_K, and W_V. These weights correspond to the query, key, and value vectors (each with d_q, d_k, and d_v components, respectively) learned for each element in the input sequence by

the individual attention heads. The idea is that the attention mechanism resembles a sort of lookup where you have a query and a set of key-value pairs that you map to an output. The query can be thought of as the information being searched for, the key how relevant some information is to the query, and the value the actual potential result for the query. This mapping from a query and set of key-value pairs to the output comprises the attention function. More formally, this mapping is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where Q, K, and V are matrices that simply pack up the query, key, and value vectors for each of the input embeddings. Thus, their respective dimensions are $N_{tok} \times d_q$, $N_{tok} \times d_k$, and $N_{tok} \times d_v$. The QK^T operation generates a new matrix where each row represents the relevance of every other value in the sequence for a particular query. This requires that $d_q = d_k$. The scalar division by $\sqrt{d_k}$ then helps to prevent the individual dot products calculated in QK^T from blowing up as d_k increases. Passing the result through the softmax function then serves to normalize each of the row vectors. Finally, multiplying the result matrix that comes out of softmax by V gives us a weighted average of the value vectors, signifying which ones have more effect on the output.

The "Add & Norm" layer that follows the multi-head attention layer and feedforward layer in the encoders consists of a residual connection which performs element-wise addition on the output of the layer with the input of the layer followed by layer normalization [32]. These steps simply comprise some computational strategies designed to improve the convergence time of the model. After this first normalization in the encoder the data is passed along through a

standard feedforward network consisting of a single hidden layer, using ReLu activation. In the base architecture, 2048 neurons are used for the hidden layer, while input and output dimensions remain the same (d_{model}). This output is then normalized again, same as the output from the multi-head attention layer.

The decoder portion of the architecture (the right portion of Figure 2) is relatively similar to the encoder portion, with the addition of a *masked* multi-head attention layer at the beginning. The decoder's job is to generate the output sequence given the intermediate representation of the input that was generated by the encoder portion. This means the decoder also uses the output of the encoder as an input – specifically, to its non-masked multi-head attention mechanism (which works in the same fashion as the multi-head attention mechanism in the encoder). The decoder is autoregressive, meaning that it predicts the output token by token and uses its previous outputs as inputs. This autoregressive property means that the decoder is not inherently parallelizable – though while doing supervised training, there is a commonly used process for training sequence-to-sequence models known as "teacher forcing" [33] which allows the decoder to be trained via the previous ground truth output for each timestep instead of having to condition itself in an autoregressive fashion, which allows for parallelization (though this only applies to the training process – generating outputs for general use is still inherently sequential and thusly non-parallelizable). This autoregressive property is also the reason for the "masking" in the first multi-head attention layer in the decoder. Since it is trained to generate the output sequence token by token, it needs to learn not to pay attention to future tokens. This masking process is exactly that – it prevents the attention mechanism from computing attention scores for future tokens in the output sequence. In particular, each of these attention scores for future tokens is driven to 0 by the mask. Another caveat of this autoregressive property is that the output

sequence must be shifted right by one position, prepending/appending start and end tags to the output sequence so that the model has a previous input for the first token (the start tag).

After passing through the masked multi-head attention layer and then normalization, this output is plugged into another (non-masked) multi-head attention component as the values, and the encoder output is plugged in as the queries and keys. This allows the decoder to learn which portions of the encoder's output to be attentive to. Then, after the data passes through this second multi-head attention layer in the decoder, it is piped through a linear layer which acts a classifier, and then a softmax layer to map this classification to a probability distribution. The token associated with the index of the highest probability component of the output is the token being predicted by the model.

Chapter 3 – Related Work

3.1 – Overview

The topic of information extraction in cybersecurity-related text is certainly not one that has not seen its fair share of attention. LSTMs have been used extensively for the detection and extraction of cybersecurity concepts in documents - Gasmi et al. [14] showed that LSTMs alone can be used effectively extract cybersecurity information from cybersecurity-related text, and Jiang et al. [6] showed that bidirectional LSTM networks along with conditional random fields could be used to effectively extract cybersecurity-related concepts and entities from a cybersecurity-focused document corpus.

3.2 - Knowledge Graphs in Cybersecurity

To build a cybersecurity knowledge graph (CKG), one must first establish an ontology that encapsulates the different concepts and relationships that are meant to be analyzed. A cybersecurity ontology generally consists of a set of cybersecurity-related classes along with their attributes as well as the potential relationships that exist between the classes. Iannacone et al. [40] developed the "Situation and Threat Understanding by Correlation Contextual Observations" (STUCCO) ontology for CKGs which incorporates data from a myriad of both structured and unstructured data resources so as to represent all the relevant cybersecurity concepts within. Syed et al. [41] extended works like STUCCO with the Unified Cybersecurity Ontology (UCO) which also encapsulates the core concepts and relationships other important cybersecurity resources like Common Vulnerabilities and Exposures (CVE), Common Vulnerability Scoring System (CVSS), and others as well as STUCCO. Importantly, it also conforms with the STIX [42] ontology, which is a popular community-driven project whose aim

is to generate a structured language to represent cybersecurity threats. Cybertwitter [43] uses cybersecurity knowledge graphs to reason about cybersecurity-related content on Twitter and inform analysts about potential threats in real time.

3.3 - NER

In the NLP domain, approaches to NER tagging problems fall under two main categories: machine-learning based and rule-based. Machine-learning based methods are those that concern themselves with the statistical relationships amongst the relevant data, while rule-based methods typically consist of manually designed dictionary lookups or pattern-matching rules that have to be derived by hand from a domain expert. Machine learning approaches typically involved techniques such as Perceptrons [51], Support Vector Machines [52], and Hidden Markov Models (HMMs) [53], but recently Conditional Random Field (CRF) and neural-based approaches have shown the most promise. To this effect, both Gasmi et al. [8] and Jiang et al. [6] show that bidirectional LSTMs that feed into a CRF classifiers are a very effective approach to cybersecurity-specific NER, beating out the previous state-of-the-art approach which was to just use CRFs.

3.4 - Relation Extraction

Models for relation extraction generally approach the problem from two different perspectives: binary classification and multi-class classification. A binary classifier takes a representation for two entities and its goal is to predict whether or not some specific relation holds. In multi-class classification, however, two entity representations go in and the model attempts to predict which of many possible relations hold [45]. The quality of generated knowledge graphs is entirely dependent on the quality of the semantic triples which make them

up. The main problem that many relation extraction systems face is the lack of adequate training data. This is because, while at first glance annotating relationships between entities in a corpus may seem trivial, it quickly becomes bogged down with vague or ambiguous instances where the correct labelling decision is not clear. This is why many group annotation efforts for relation extraction end up with a large amount of inner-annotator disagreement [44].

In terms of machine learning approaches to the relation extraction task, they fall into three categories: supervised, semi-supervised, and unsupervised. Zhao et al. [46] explore kernel-based methods for supervised relation extraction while Kambhatla et al. [47] investigate feature-driven methods. Pingle et al. [50] perform relation extraction on cybersecurity text based on the UCO ontology by using feed-forward neural networks as classifiers on Word2Vec encodings of words. Semi-supervised methods are investigated by Yarowsky [48] and Blum et al. [49]. Blum et al. use a technique called co-training, where a small set of labelled data and a large set of unlabeled data represented with disjoint feature sets are used in tandem to learn the task. Yarowsky [48] trains a classifier on a small set of seed examples, and then uses that semi-trained model to label a larger unlabeled dataset – only paying heed to predictions where the model is highly confident. The simple set of steps is then repeated until the convergence criteria is met, with the set of labeled seeds growing every iteration and the set of unlabeled entries conversely shrinking. On the unsupervised side of the spectrum, Elsahar et al. [49] use state-of-the-art clustering algorithms to perform relation clustering without the need for any labelled data.

Chapter 4 - Pipeline

4.1 – High-level Overview

Our method aims to generate optimized knowledge graphs from malware-focused cyber security news articles and blog posts. The text used to train the model were taken from articles from various media outlets reporting on prominent malware strains. These were found by simply taking the names of popular malware then searching for them on Google News, extracting text from various articles located on the first five pages of the results. Lengthier articles containing more relevant information for the scope of this approach were prioritized so as to have enough labelled entries. Since having to read all the text from the articles would be tedious and time-consuming, relationships between cyber security-related entities are extracted. The fact that many of these malware articles contain repeated information is an issue, so an algorithm for preventing relationships representing already-seen information from being inserted into knowledge graphs is used to reduce their size. Thus, knowledge graphs can be generated for multiple articles and merged, with the output not being cluttered by repeated information.

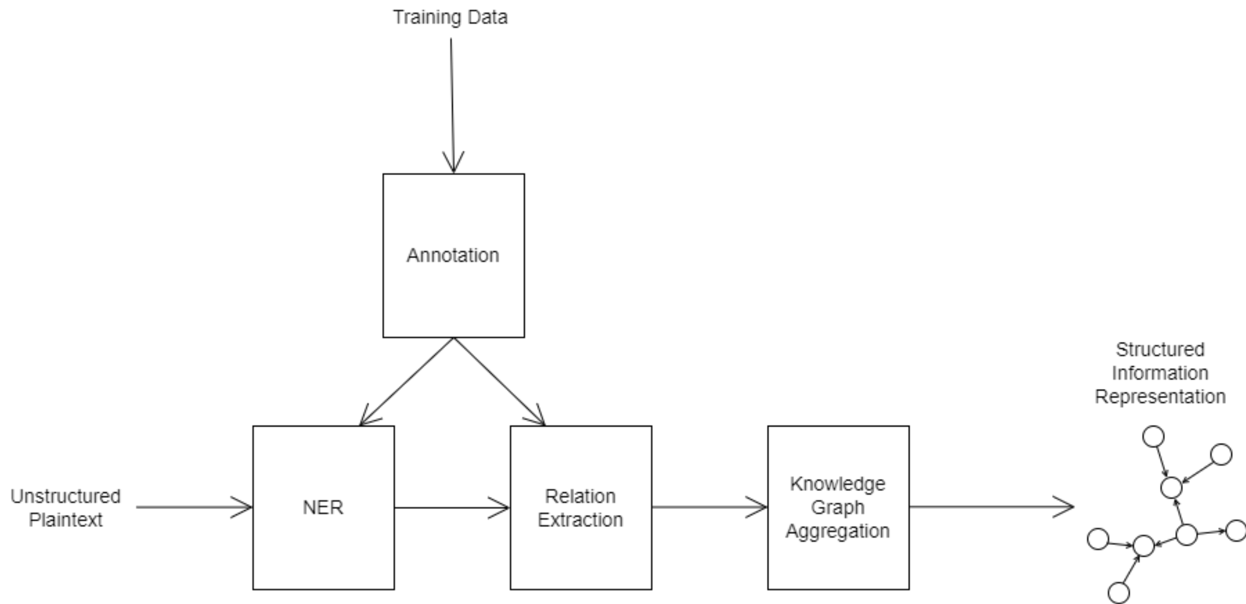

Figure 3: High level pipeline architecture

Figure 3 shows the high-level information flow for the pipeline presented in this paper. First, a corpus of text consisting of articles from multiple high-profile media sources such as cybersecurity-related news articles and blogs is assembled. This manually annotated corpus is used to train the separate NER and relation extraction models. Once the models have been trained, unstructured text can be passed into the NER component to have entities extracted. The NER component uses transfer learning because it allows us to exploit the rich model weights learned on a similar task with much more training data, since the manually labelled corpus created for this work is limited in size. As the model was trained on a similar task, these weights can then just be fine-tuned to our needs, which decreases convergence time as compared to starting with random weights. The eventual set of identified cyber-security entity tags consists of the following: MALWARE, MALWTYPE (malware type), SOFTWARE, SOFTWTYPE (software type), VULNERABILITY, ATTACKTYPE, DEVICE, DATA, and VERSION. Since transfer learning is leveraged on a pre-trained model with rich learned weights at the start, other

non-domain specific entity tags such as LOC (location), PER (person), GPE (geopolitical entity), etc. are also present in the training corpus. Specifically, the pre-trained model is trained to recognize all of the entity tags present in the OntoNotes [34] NER dataset. Once these entities are tagged in the input text, the relation extraction component attempts to identify the relationships between them.

The four relation classes trained to predict are IS_MALWTYPE, EXPOSES, INFECTS, and VULNERABLE_TO. The focus of the IS_MALWTYPE relation class is to help users identify what high-level classification of the malware falls under. Is it a worm, a piece of ransomware, spyware, or a keylogger? Each of these groups, while broad in scope, may be useful when first encountering a new strain of malware to gauge its general capabilities. The EXPOSES class is meant to label relationships that involve some type of data being accessed by a specific malware. The INFECTS relation class represents a connection between a malware and software or device entity, but the name should not be taken too literally. If a piece of malware "infects" some software in this context, that only means that it uses that software as part of its attack vector. And if a piece of malware "infects" a device entity, that simply means that the malware functions on that device. Finally, VULNERABLE_TO is named pretty literally – it is meant to be a relationship between a software entity and a vulnerability entity that shows the software is vulnerable to that specific vulnerability. On top of being the least frequently occurring relations in the training data, the INFECTS and VULNERABLE_TO relation classes are also the only classes who are capable of consisting of variable entity types. A relation like IS_MALWTYPE is *always* between a MALWARE subject entity and a MALWTYPE object entity, which is likely easier for the model to understand. On the other hand, a relation class like INFECTS is sometimes binds a MALWARE subject and SOFTWARE object but also

sometimes binds a MALWARE subject with a DEVICE object. In hindsight, these separate relations are probably better served as separate relation classes.

After these relationships are extracted, they are converted into semantic triples to form knowledge graphs and then relations are aggregated, with common information being filtered out via word embedding similarity scores. The dataset was annotated using Prodigy [52], and the NER & relation extraction pipelines were implemented using the spaCy [53] and Thinc libraries [54].

4.2 - NER Pipeline

The NER component of the pipeline consists of two main pieces – a RoBERTa encoder mechanism that feeds into a transition-based incremental parser model for the actual entity span tagging. In this context, the model being transition-based means that the objective task is to learn to map a sequence of tokens to a sequence of transitions in a state machine. In this work, a shift-reduce parser architecture is used for the entity tagging, following Lample et al. [37]. Figure 4 gives a walkthrough of the objective task from the model's perspective. It shows an example of shift-reduction parsing on an input sentence of "Fake AV is a trojan." The algorithm uses an input stack as well as a buffer to represent its internal state. The buffer begins loaded with the words in the sequence, and the overall goal is to iteratively remove these words from the buffer while following the correct transition actions that lead to the output containing the correct labelled spans. The SHIFT transition moves the first item in the input buffer directly to the internal stack. The OUT transition, on the other hand, moves the first item in the input buffer directly into the output sequence. REDUCE is a parameterized transition, taking the predicted entity class as input. The REDUCE transition pops and concatenates all the items on the internal stack into a single span representation with the correct label and moves that to the output.

Transition	Output	Stack	Buffer
	[ ]	[ ]	[Fake, AV, is, a, trojan]
SHIFT	[ ]	[Fake]	[AV, is, a, trojan]
SHIFT	[ ]	[Fake, AV]	[is, a, trojan]
REDUCE(MALWARE)	[(Fake AV)-MALWARE]	[ ]	[is, a, trojan]
OUT	[(Fake AV)-MALWARE, is]	[ ]	[a, trojan]
OUT	[(Fake AV)-MALWARE, is, a]	[ ]	[trojan]
SHIFT	[(Fake AV)-MALWARE, is, a]	[trojan]	[ ]
REDUCE(MALWTYPE)	[(Fake AV)-MALWARE, is, a, (trojan)-MALWTYPE]	[ ]	[ ]

Figure 4: Step-by-step example of shift-reduction parsing on an input sentence of "Fake AV is a trojan"

The pipeline follows the popular "Embed, Encode, Reduce, Predict" framework proposed by Honnibal [36]. Figure 5 visualizes the process. The first step, "embed", entails embedding words in the input sequence into meaningful vector representations. These representations, however, are not yet contextually aware, which is where the "encode" step comes in. Given a sequence of these non-context-sensitive word embedding vectors, the goal is then to create a matrix that encodes the contextual relationships of the different words. In such a matrix, each row vector corresponds to the contextual meaning of a particular token to the whole sequence. This may sound familiar, and that is because this is the general notion of attention. Traditionally, pretrained word embeddings and attention-based Bi-LSTM/GRU models to achieve the first two steps. The transformer-based encoder architecture (RoBERTa), however, solves both problems in one with its multi-head attention mechanism. We just use the outputs from the augmented transformer encoder stack as the results for the encode step. The third step, "reduce", consists of taking these matrices produced in the "encode" step and reducing them to a single vector to be used in downstream prediction tasks. In this case it is done by taking the RoBERTa-encoded token vectors and concatenating them into one large vector, then passing the large vector through a feed-forward network to get a smaller vector representation of the state. This resultant state vector is then finally used for the transition-based classification task. In figure 5 below, the RoBERTa encoder component composes the encode and embed steps, while the reduce step is

handled by the first feedforward network and the predict step is handled by the second feedforward network along with the softmax layer to map the outputs to a probability distribution.

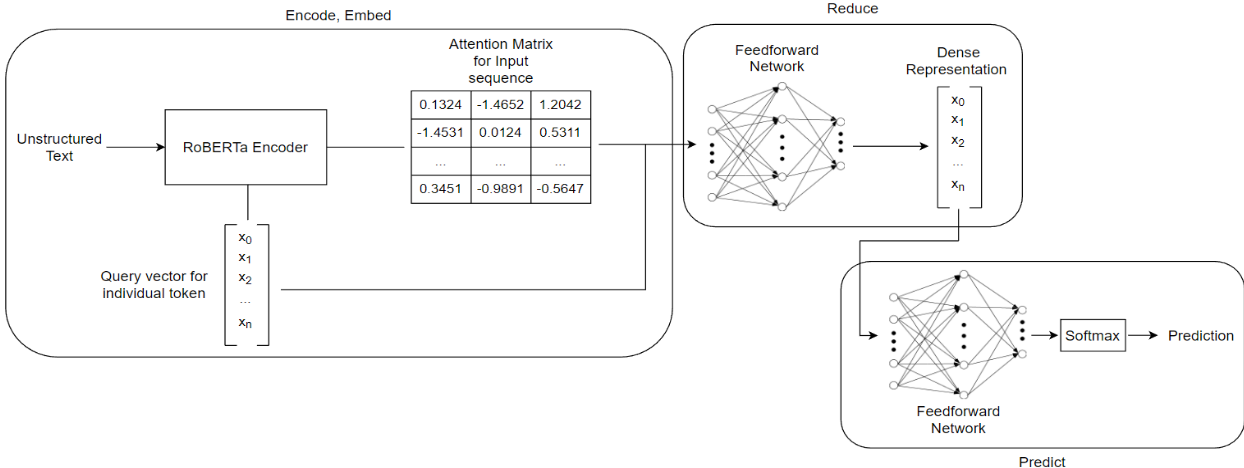

Figure 5: Embed, Encode, Reduce, Predict framework process outlined (Feedforward Network graphic from [56])

4.3 - Relation Extraction Pipeline

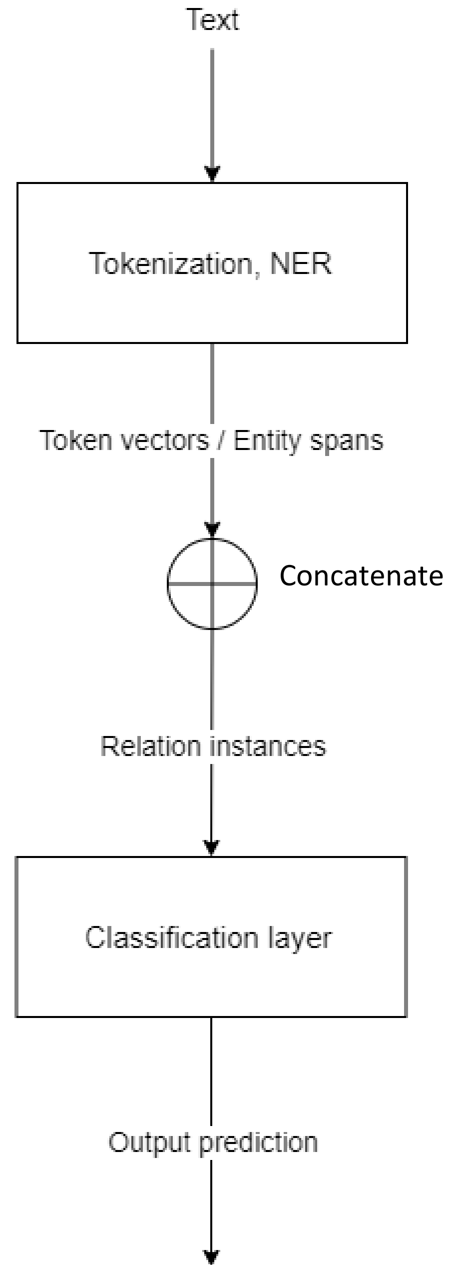

Figure 6: High-level overview of relation extraction model

The relation extraction component of the pipeline is directly downstream from the NER component. In this work, the goal is to extract relationships between entities that convey some sort of potentially useful information to someone seeking information about a strain of malware. For instance, it is useful to know that a given piece of malware is grouped under a specific type (ransomware, botnet, keylogger, etc.) or if it exploits a certain non-malicious piece of software. Since relation extraction in this context only considers relations between specific entity classes and not general tokens, the NER component is used to determine which candidates to check for relations. In particular, the RoBERTa-generated token vectors that belong to an entity span are used to generate the relation input representations. For entities consisting of multiple tokens, their representation is pooled by taking the average of the constituent token vectors. To represent the relations between entities, the entity representations are simply concatenated. So, if we have entities E_1 and E_2 with corresponding representations $e_1 = \langle x_1, x_2, \ldots, x_n \rangle$ and $E_2 = \langle y_1, y_2, \ldots, y_n \rangle$ then we represent the potential relation candidate as $r_{E_1,E_2} = e_1 \oplus e_2 = \langle x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n \rangle$. However, since relations are directional (i.e., A infects B is not the

same as B infects A) a representation for both directions should be captured. Thus, for each pair of entities E_i and E_j in an input span of text, relationship candidates r_{E_i,E_j} and r_{E_j,E_i} are both generated and passed forward to the classification layer, which is a standard feedforward neural network with a softmax output. The classification layer takes these candidate relation instances and maps them to a probability distribution concerning the relevant relation classes: IS_MALWTYPE, VULNERABLE_TO, EXPOSES, and INFECTS. The predicted class with the highest probability is selected, but only if that probability is greater than 0.5. It is worth noting that unlike the original BERT and RoBERTa models, the input texts it can process can have a maximum token length of 300. Any token sequences longer than that are batched into sequences of size 300 and treated independently.

4.4 - Knowledge Graph Merging

After the relations have been extracted, they are stored in the knowledge graph data structure. Conceptually, these knowledge graphs are simply a data structure consisting of semantic relation triples (subject-relationship-object triples). Nodes are entities, and edges are relationships. Knowledge graphs exist as directed graphs, where if there are two nodes u and v and an edge e from $u \rightarrow v$, it means that the entity represented u is related to v by the relationship e. In other words, the semantic relation triple (u, e, v) exists inside the knowledge graph. Implementation-wise, the knowledge graph used in this work consists of three mappings implemented as hashtables: a map from entity plaintext (lowercase) to the corresponding Span data structure that represents the entity span (contains information like token start and end indices, word vector representations, etc.), a bidirectional map from spans to numerical IDs (meaning that for any key-value pair (k_i, v_i) in the mapping there exists another key-value pair (k_j, v_j) where $k_j = v_i$ and $v_j = k_i$), and finally a nested adjacency map where the keys are

subject entity spans that map to new hashtables. Those inner hashtables consist of relation-entity list pairs, allowing constant-time lookup for any query of a specific type of relation on a subject entity. While this structure can be good for querying as it allows constant-time lookup of both subject entities and relationships, it should be noted that it is not optimal in terms of memory performance/locality and the nested mappings in particular will probably suffer in terms of cache performance.

When adding relations to a knowledge graph, pre-trained gloVe word embeddings are utilized to eliminate redundant information via the cosine similarity measure. Recall that word embeddings are designed to project your word representations onto a vector space where representations for similar words lie near each other. By “similar”, what we really mean is that two words have a tendency to be associated with common words. For instance, we can tell the words “laptop” and “desktop” are similar not based on knowledge of their *meaning*, but by the fact that in the training data both “laptop” and “desktop” had a strong tendency to be near words such as “typing”, “websurfing”, “computer”, etc. Exploiting this property for the tokens in the entity spans allows us to remove relations that might mean the same thing but use different phraseology. The algorithm for adding these relations is shown in Algorithm 1.

Algorithm 1: Knowledge Graph relation merging

```
Data: Knowledge Graph (KG)
      Subject Entity (A)
      Relation label (r)
      Object entity (B)
Result: KG with relation (A, r, B) added (if not deemed redundant)

Function addRelation(KG, A, r, B) is
    if A ∈ KG.Vocab ∨ B ∈ KG.Vocab then
        foreach (subject, relationMap) ∈ KG.subjectMap do
            if r ∈ relationMap then
                foreach object ∈ relationMap do
                    A.similarity ← ENT_SIMILARITY(A, subject)
                    B.similarity ← ENT_SIMILARITY(B, object)
                    totalSimilarity ← (A.similarity + B.similarity) / 2
                    if totalSimilarity ≥ 0.8 then
                        if |A.Tokens| + |B.Tokens| >
                         |subject.Tokens| + |object.Tokens| then
                            Replace (subject, r, object) in KG with (A, r, B)
                        end
                        return KG
                    end
                end
            end
        end
    end
    Insert (A, r, B) into KG
    return KG
end
```

A cutoff total relation similarity score of 0.8 seems to work well in practice. This seems a bit high, but this is because all relations between entities will consist of at least one entity who is referred to by a proper noun. Since relations will always contain an entity type of either MALWARE or SOFTWARE, and these entities only match to the *names* of malware (Zeus, Mirai, Emotet, WannaCry, etc.) and software (Outlook, Chrome, Discord, Spotify, etc.), it is almost always the case that any relation that would qualify as redundant would have an entity whose lowercase text is an exact match with one that has already been seen. It is also somewhat necessary as software/malware whose name has not been seen before will most likely not have

pre-trained embeddings present in the embedding table). This does carry the caveat that some software that is occasionally referred to with the creator's name attached (for instance "Google Chrome" vs. "Chrome") will not have an exact match although the two spans may refer to the same piece of software; however, in practice it seems that software of this criterion (Photoshop & Adobe Photoshop, Excel & Microsoft Excel, etc.) tends to be popular enough that both the software names and creator names are present in the pre-trained embedding table so it is sort of a non-issue. The total similarity of two relations is comprised of the average similarity scores between the individual entity spans. If we define the relation triplets as $rel_{A,B} = (A, r_{c_i}, B)$ where A is the subject entity span, B is the object entity span, and r_{c_i} is the relation class that binds the entities, then the relationship between another relation $rel_{C,D} = (C, r_{c_j}, D)$ is computed as:

$$\text{relation_similarity}(rel_{A,B}, rel_{C,D}) = \frac{\text{ent_similarity}(A, C) + \text{ent_similarity}(B, D)}{2}$$

$$\text{ent_similarity}(X, Y) = \begin{cases} 1 \text{ if } X_{lowercase} = Y_{lowercase} \\ \dfrac{\sum x_i \cdot \sum y_j}{\sqrt{\sum x_i^2}\sqrt{\sum y_j^2}} \text{ otherwise} \end{cases}$$

where x_i and y_i are the individual token vectors that may make up a multi-token entity span. This driving of the similarity to 1 for spans that have the same lowercase plaintext drives the average similarity up relatively high for potentially non-redundant relation candidates, so even if the other entity pair in the compared relations scores low in terms of similarity, it will still have a score of at least 0.5 (because the result of $\frac{1+x}{2}$ with $0 \le x \le 1$ is bound between $[0.5,1]$). This is why a lower relation score of say, 0.6 or 0.7 – which some might consider more reasonable at first glance – is not used.

Chapter 5 – Evaluation Results

5.1 - Dataset

For the NER component of the pipeline, a total of 2,324 new cyber security entity examples were labelled using the Prodigy annotation tool. This set of new cybersecurity entities consists of the MALWARE, MALWTYPE, SOFTWARE, SOFTWTYPE, ATTACKTYPE, DEVICE, DATA, VULNERABILITY, and VERSION entity classes. The text samples were selected from news articles concerning different types of specific malware that was breaking at the time. These text samples were gathered by extracting top news articles under the "News" section of Google search for prolific malware strains such as Zeus, Emotet, Mirai, etc. The articles themselves were split into chunks, typically dictated by the paragraph separations in the original publication. This is to compensate for the 128-token span that the transformer component of the NER pipeline can process. Since transfer learning was leveraged with a pre-trained transformer model, it was imperative that the labelled dataset not exclude labels for the classes the original model was trained to detect. If not, the model risks lower performance as well as the fact that it will begin to "forget" the original labels it was trained to predict. Preserving the previous labelling functionality is cheap (simply pre-annotate the corpus using the pre-trained model) and allows for more information to be extracted in the future – such as relation types that involve non-cybersecurity specific entities. The entities are processed using the BILOU (Beginning, Inside and Last tokens of multi-token chunks, Unit-length chunks and Outside) tagging scheme, where every individual token is given a specific tag, with special "beginning" (B-TAG), "inside" (I-TAG), and "last" (L-TAG) entity tags reserved for handling multi-token entity spans and a "unary" (U-TAG) tag for single-token entity spans. Tokens which are not part of an entity span simply receive the label "O" for "outside". Take the sentence

“Microsoft Office 365 is vulnerable to buffer overflow attacks from the virus”. With this tagging scheme it would be labelled as: (Microsoft (B-SOFTWARE), Office (I-SOFTWARE), 365 (L-SOFTWARE), is (O), vulnerable (O), to (O), buffer (B-ATTACKTYPE), overflow (L-ATTACKTYPE), attacks (O), from (O), the (O), virus (U-MALWTYPE)).

Entity Class	Count
MALWARE	571
MALWTYPE	368
DEVICE	342
SOFTWARE	302
SOFTWTYPE	277
DATA	257
VULNERABILITY	98
ATTACKTYPE	80
VERSION	29

Table 1: Counts for different entity types in labelled corpus

The relation extraction data has considerably less training examples than the NER data. In total, there are 670 relation examples across four relationship classes. These relations were labelled on the same text that was used for training the NER component, since the entities need to be identified for labelling the relationships anyway. This is why there are significantly less training instances in comparison – because relation instances are considerably less common in article text than entities themselves. This is to be expected, seeing as that every relation will involve at least two entities but there are plenty of instances of multiple entities that do not form a relation. The four relation classes, IS_MALWTYPE, EXPOSES, INFECTS, and VULNERABLE_TO, were chosen because there seemed to be a decent balance between their utility to a potential analyst and their frequency in malware article text.

Relation Class	Count
IS_MALWTYPE	225
EXPOSES	176
INFECTS	172
VULNERABLE_TO	97

Table 2: Counts for different entity types in labelled corpus

5.2 – NER Results

Metric	Score
F-Score	81.85%
Precision	78.42%
Recall	85.59%

Table 3: Scoring metrics and results for NER evaluation

The NER component of the pipeline achieved decent results, especially for such a limited training dataset, with a final F-score of 82% on the test set. The formulation for precision, recall, and F-score for the NER component are as follows (only *exact* matches considered correct, partially correctly predicted entity spans are given no credit):

$$\text{Precision} = \frac{\text{\# of correctly predicted entities}}{\text{\# of predicted entities}}$$

$$\text{Recall} = \frac{\text{\# of correctly predicted entities}}{\text{\# of actual entities in data}}$$

$$\text{F-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The train-validation-test split for the custom-annotated NER corpus was 70% - 20% - 10%. During training, the corpus data was shuffled and processed as minibatches of size 128. Adaptive moment estimation (Adam) [38] optimization was used when performing gradient descent along

with L2 regularization with a term of 0.01 to lower the model complexity and reduce overfitting. An initial learning rate of 5e-5 was used, and the model took 225 epochs to converge.

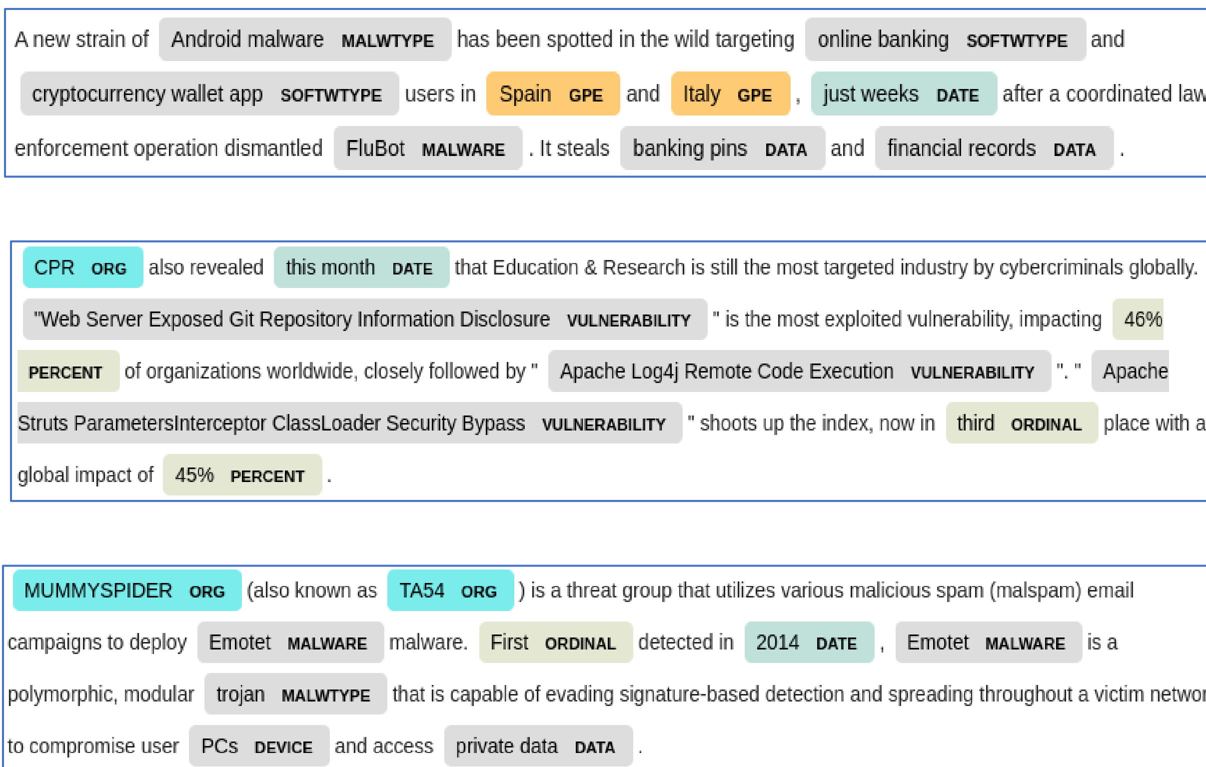

Figure 7: Separate text sequences labelled by the NER component

Figure 7 shows some pieces of example text after being processed by the NER component. Since transfer learning was leveraged in the training of the model, it still detects and identifies the general, non-cybersecurity related entities that it was initially trained on such as ORG, DATE, GPE, etc. This extra functionality could be leveraged later on to extract even more meaningful relations from the text, concerning relations that are not limited to occurring between two cybersecurity-related entity classes. When annotating, it was decided that entity mentions who do not necessarily correspond to a concrete entity should also be labelled as to maximize the amount of potential information extracted. For example, in the third snippet from Figure 7, we can see that the model labelled the “private data” span as DATA. While this span does not refer

to a specific *instance* of a kind of data (password, banking pin, medical records, etc.) it still conveys some level of information to the reader. Just knowing the malware exposes some level of private data may be useful to an analyst – at the very least, it is more useful than no relation at all.

5.3 - Relation Extraction Results

Metric	Score
F-Score	61.68%
Precision	63.46%
Recall	60%

Table 4: Scoring metrics and results for relation extraction evaluation

The relation extraction portion of the pipeline does not achieve great performance, though for the limited data it was trained on it is still significant. After ten separate training sessions, the best model achieved an F-score of 62%. Again, the train-validation-test split was 70% - 20% - 10%, and the model took 112 training epochs to converge. Adam optimization was used for stochastic gradient descent with an L2 regularization term of 0.01 and a learning rate of 5e-5. The transformer portion of this component had some hyperparameters modified from the transformer component in the NER component – namely the span length that the transformer attends to at a time. Instead of having a maximum token length of 128, it was increased to 300. This is to allow for longer distances between entity spans that might be related. The formulation of precision, recall, and F-score in the context of the relation extraction are the same as in the NER component, only considering predicted relations instead of entity classes.

Figure 8A shows relations extracted from ten articles revolving around the new ZuoRAT malware. While there are a number of relations that are not quite correct, there is still a

significant amount of useful, correct information extracted. At first glance, we can see there is a disconnected component of the knowledge graph consisting of one relation: "MIPS INFECTS distributed denial-of-service attacks" – this is essentially nonsense, as MIPS is an instruction set architecture for processors, and distributed denial-of-service attacks are certainly not computing entities which a malware could potentially infect. Local Area Networks (LANs) are probably also not vulnerable to Domain Name Systems (DNSs) as indicated by the graph either. Many of the incorrect relation labelings involve the VULNERABLE_TO class, which is reasonable because this class ended up being pretty underrepresented in the training data. These relations also tend to be a lot more nuanced to infer from the text, compared to something like IS_MALWTYPE. Typically, a MALWTYPE entity is mentioned in close proximity to the name of the actual malware, many times even directly adjacent as these entities are often used as descriptors when introducing the malware (i.e., "The new ZCryptor ransomware…", "Dubbed CloudExe, this new botnet…"). In general, the IS_MALWTYPE and EXPOSES relation classes seemed to be more easily inferable from the raw text than the others (less variation in the sentence structures, etc.) and the model also seems to perform a bit better on them. Some of the relations can also be somewhat vague – as in, they do not provide much specific information or even provide information that is very obvious. For instance, take the relations from Figure 8A which read "ZuoRAT INFECTS systems" or "ZuoRAT EXPOSES other info". While true, these relations do not offer much in terms of useful information because they are either too vague to be useful or trivially obvious.

There are also a number of useful relations extracted as well. From the graph we can see that ZuoRAT is a remote access trojan (RAT), and this is true. Coincidentally, the malware type happens to be a part of the name, however the model is ignorant to this fact when predicting and

it thusly has no effect. We can also see that ZuoRAT infects small office/home office (SOHO) routers, along with another similar malware that was mentioned called VPNFilter. Even broader than that, ZuoRAT infects Internet of Things (IoT) devices along with the Mirai malware strain. Specifically, the graph indicates that ZuoRAT infects a specific type of router, the JCG-Q20. We can also see that the malware exposes the following information: routing tables, HTTP traffic, DNS information, IP addresses, and some sort of credentials. The reason the “DNS” entity does not contain the specifier “traffic” is because the unstructured text it was extracted from was of the form “…DNS and HTTP traffic…”. Had the text instead looked like “…DNS traffic and HTTP traffic…” each entity would contain the specifier.

Figure 8A: Relations extracted from 10 articles about the ZuoRAT malware strain (redundancies filtered out)

Figure 8B: Relations extracted from ZuoRAT articles (redundancies not filtered out)

5.4 - Knowledge Graph Merging Results

The relational similarity measure seems to be relatively effective at reducing clutter in aggregated knowledge graphs, with experimentation even showing that redundant information is pretty plentiful in single articles, let alone collections of articles speaking on the same malware. The knowledge graphs from Figures 8A & 8B, generated from ten separate news articles concerning the ZuoRAT malware, showed a reduction of 27% in the number of relation triples

after using the filtering technique. Below are some images of generated knowledge graphs for two recent articles on the Emotet malware:

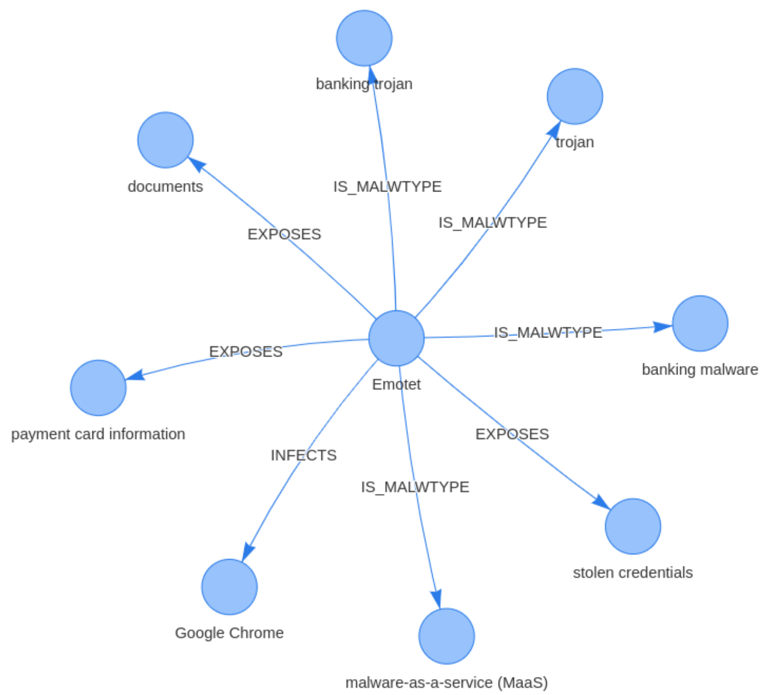

Figure 9A: Knowledge graph for article X concerning recent malware Emotet generated without redundancy removal

Figure 9B: Knowledge graph also for article X, this time generated with redundancies filtered out

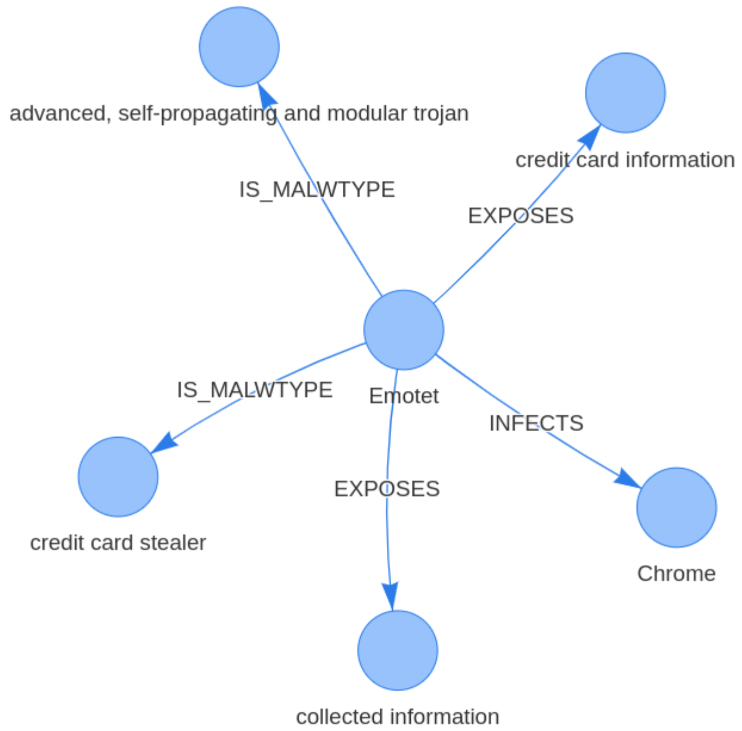

Figure 9C: Knowledge graph for another article, Y, concerning recent malware Emotet generated without redundancy filtering

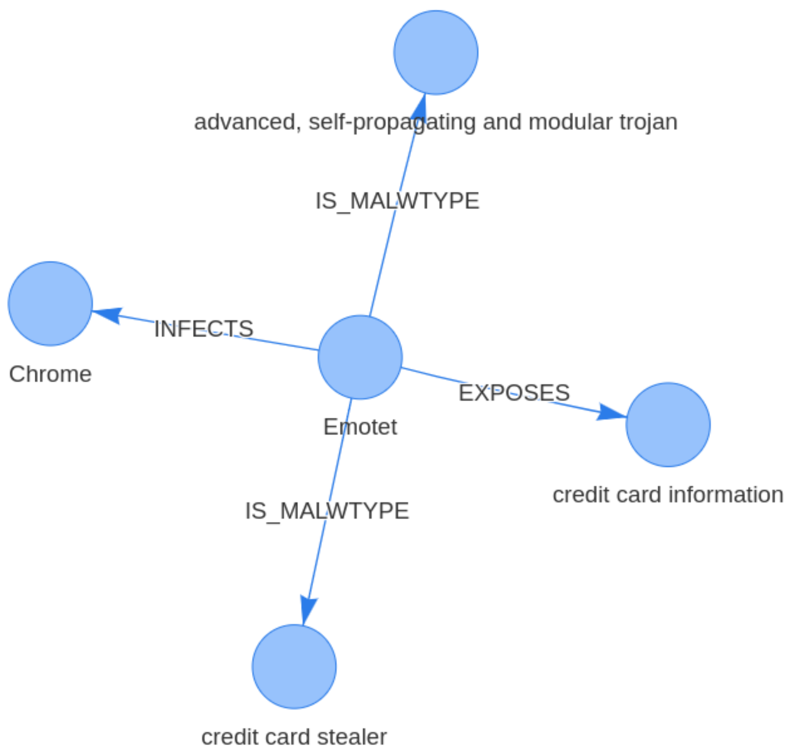

Figure 9D: Knowledge graph for article Y generated with redundancy filtering

Figure 9E: knowledge graph for merged articles X & Y used in Figures 9A & 9C without redundancy removals

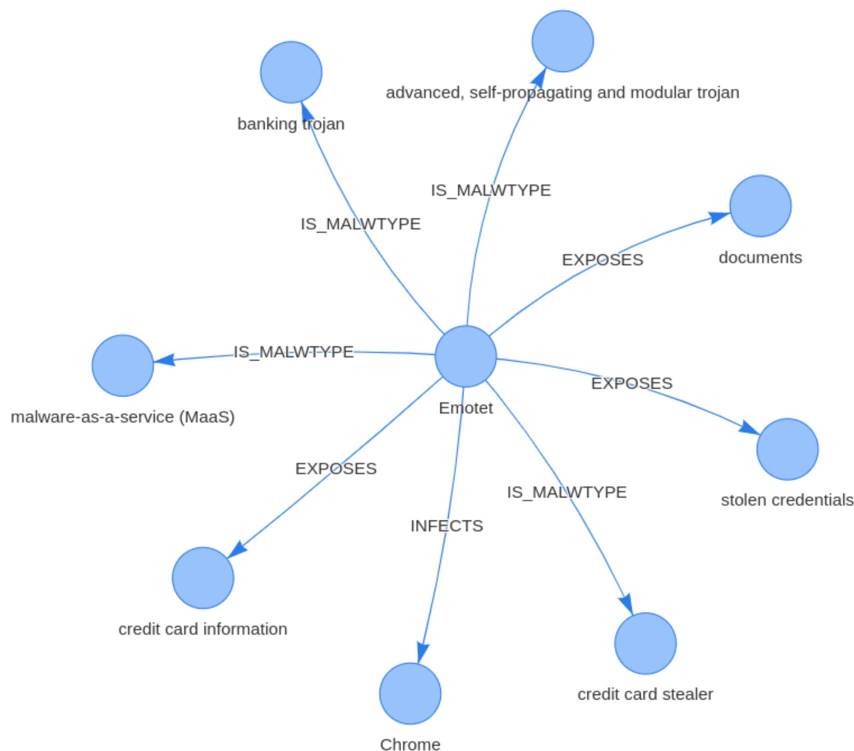

Figure 9F: knowledge graph for merged articles X & Y used in Figures 9A & 9C with redundancy removals

Figures 9A-9F above show a demonstrations of generated knowledge graphs and the results of redundancy removals when creating/merging them. In the end, we see a reduction of 40% in the number of relations in the final merged knowledge graph when using redundancy filtering versus without it. Something that might stick out about these smaller knowledge graphs is that they contain a central node from which all other nodes are immediately adjacent. This is not so unusual, as these are both generated from articles that talk specifically about the re-outbreak of the Emotet malware in June of 2022, and the vast majority of relations are centered around specific malware (i.e., they will contain an entity of type MALWARE as either the subject or object). This is the case for many articles pertaining to a single piece of malware (of which, most of the training corpora consisted of). We can also see that different SOFTWARE entity spans that refer to the same piece of software in the same relation do not break the similarity mechanism (assuming they have pre-trained embeddings) as the relation "Emotet

INFECTS Google Chrome" was never added since "Emotet INFECTS Chrome" was present in the other article. In hindsight, it may have been more reasonable to label "Google" as an organization and only "Chrome" as the software, which would mitigate this potential for misinterpretation. The first article also contained a relation "Emotet EXPOSES payment card information" while the second contained the relation "Emotet EXPOSES credit card information" which was filtered out due to the fact that, while phrased differently, they convey the same information.

It is a common trend as well that articles themselves contain repeated information, so the application of the relation merging algorithm is also useful when looking at single instances of a document. For example, article X refers to the Emotet malware as both a "trojan" as well as a "banking trojan". The addition of the fact that it is a banking trojan does not differ enough to be relevant, but we can see that in the second article the malware was described (and labelled as) an "advanced, self-propagating and modular trojan". This conveys enough difference in meaning to be ruled as semantically different from a "banking trojan". This is desirable, as a "banking trojan" does not indicate that the trojan itself is modular or, more importantly, self-propagating – so this is information you would probably want to know and not have filtered out. Another thing to notice is that, while "Emotet EXPOSES credit card information" and "Emotet IS_MALWTYPE credit card stealer" seem to represent the same information (that Emotet wants your credit card information) but are not filtered out. This is because relations are only considered for redundancy against each other if they are of the same relation class.

Chapter 6 – Conclusion and Future Work

Transformer-based neural architectures prove to be a relatively effective way to train domain-specific NER and relation extraction language models. They are able to achieve useful results on complex sequence prediction tasks with a fairly insignificant amount of training data from which to learn, while still using less complex supervised techniques. After running unstructured text through a transformer-powered information extraction pipeline, pre-trained static word vectors generated via the gloVe technique are also shown to be reliable when it comes to removing redundant information from knowledge graphs.

In the future, work can be done to expand the entity types to encompass more cybersecurity-related concepts such as files, software components, protocols, etc. though this expanded entity set would undoubtedly require a larger and more robust text corpus for labelling if using the supervised training techniques used in this paper. Even with close to one thousand paragraphs of real article text, some of the entity types such as ATTACKTYPE and VERSION still did not have enough labelled instances for the model to learn to tag them effectively. Expanding the training corpus to a wider range of text material such as tweets or NIST NVD entries may also compensate for the lack of certain entity types – for example NIST NVD entries tend to contain many instances of version numbers.

Unsupervised or semi-supervised relation extraction techniques could also be leveraged to lessen the annotator's burden. Since annotating things like entity spans and relations are somewhat complex compared to a number of other annotation tasks (for example, image classification labelling) this is paramount. Livio et al. [39] have proposed non-fully-supervised training techniques (both semi-supervised and unsupervised) for general relation extraction – if applied successfully to this domain-specific task they could streamline the process greatly.

The redundancy filtering technique could also be expanded upon. In this work, static word vectors which are not context sensitive are used to measure similarity between relations. While this yields usable results, it may be possible to improve the effectiveness by using learned context-sensitive embeddings. For instance, if you have a “charging bank” device entity, and a “bank credentials” data entity, the static word embedding for “bank” in both instances would be exactly the same even though they refer to entirely different things.

Bibliography

[1] Chiticariu, L., Li, Y., & Reiss, F. (2013, October). Rule-based information extraction is dead! long live rule-based information extraction systems!. In Proceedings of the 2013 conference on empirical methods in natural language processing (pp. 827-832).

[2] Soomro, P. D., Kumar, S., Shaikh, A. A., & Raj, H. (2017). Bio-NER: biomedical named entity recognition using rule-based and statistical learners. International Journal of Advanced Computer Science and Applications, 8(12).

[3] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[4] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE transactions on Signal Processing, 45(11), 2673-2681.

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[6] Ma, P., Jiang, B., Lu, Z., Li, N., & Jiang, Z. (2020). Cybersecurity named entity recognition using bidirectional long short-term memory with conditional random fields. Tsinghua Science and Technology, 26(3), 259-265.

[7] Gasmi, H., Laval, J., & Bouras, A. (2019). Information extraction of cybersecurity concepts: an LSTM approach. Applied Sciences, 9(19), 3945.

[8] Gasmi, H., Bouras, A., & Laval, J. (2018). LSTM recurrent neural networks for cybersecurity named entity recognition. ICSEA, 11, 2018.

[9] Sebastiani, F. (2002). Machine learning in automated text categorization. ACM computing surveys (CSUR), 34(1), 1-47.

[10] Socher, R., Bauer, J., Manning, C. D., & Ng, A. Y. (2013, August). Parsing with compositional vector grammars. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 455-465).

[11] Ganguly, D., Roy, D., Mitra, M., & Jones, G. J. (2015, August). Word embedding based generalized language model for information retrieval. In Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval (pp. 795-798).

[12] Turian, J., Ratinov, L., & Bengio, Y. (2010, July). Word representations: a simple and general method for semi-supervised learning. In Proceedings of the 48th annual meeting of the association for computational linguistics (pp. 384-394).

[13] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[14] Gasmi, H., Laval, J., & Bouras, A. (2019, October). Cold-start cybersecurity ontology population using information extraction with LSTM. In 2019 International Conference on Cyber Security for Emerging Technologies (CSET) (pp. 1-6). IEEE.

[15] Gasmi, H., Laval, J., & Bouras, A. (2019). Information extraction of cybersecurity concepts: an LSTM approach. Applied Sciences, 9(19), 3945.

[16] National Vulnerability Database. Available online: https://nvd.nist.gov/ (accessed on 24 July 2019).

[17] Nguyen, T.H.; Grishman, R. Event detection and domain adaptation with convolutional neural networks.In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; Volume 2, pp. 365–371.

[18] Gong, L., Crego, J. M., & Senellart, J. (2019, November). Enhanced transformer model for data-to-text generation. In Proceedings of the 3rd Workshop on Neural Generation and Translation (pp. 148-156).

[19] Gao, Y., Zhou, M., & Metaxas, D. N. (2021, September). UTNet: a hybrid transformer architecture for medical image segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 61-71). Springer, Cham.

[20] Maksutov, A. A., Zamyatovskiy, V. I., Morozov, V. O., & Dmitriev, S. O. (2021, January). The Transformer Neural Network Architecture for Part-of-Speech Tagging. In 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus) (pp. 536-540). IEEE.

[21] Yan, H., Deng, B., Li, X., & Qiu, X. (2019). TENER: adapting transformer encoder for named entity recognition. arXiv preprint arXiv:1911.04474.

[22] Maryam M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. 2015. Deep learning applications and challenges in big data analytics. Journal of Big Data 2, 1 (2015), 1–21

[23] Mozer, M. C. (1995). "A Focused Backpropagation Algorithm for Temporal Pattern Recognition". In Chauvin, Y.; Rumelhart, D. (eds.). Backpropagation: Theory, architectures, and applications. ResearchGate. Hillsdale, NJ: Lawrence Erlbaum Associates. pp. 137–169. Retrieved 2017-08-21.

[24] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[25] Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25). San Francisco, CA, USA: Determination press.

[26] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[27] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471.

[28] J., R. T. J. (2021, September 10). LSTMs explained: A complete, technically accurate, conceptual guide with keras. Medium. Retrieved June 26, 2022, from https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

[29] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[30] Shewalkar, A. (2019). Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. Journal of Artificial Intelligence and Soft Computing Research, 9(4), 235-245.

[31] Fu, R., Zhang, Z., & Li, L. (2016, November). Using LSTM and GRU neural network methods for traffic flow prediction. In 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC) (pp. 324-328). IEEE.

[32] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

[33] Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. Neural computation, 1(2), 270-280.

[34] Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., & Weischedel, R. (2006, June). OntoNotes: the 90% solution. In Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers (pp. 57-60).

[35] George A. Miller (1995). WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41.

[36] Matthew Honnibal. Embed, encode, attend, predict: The new deep learning formula for

state-of-the-art NLP models. Blog, Explosion, November, 10, 2016.

[37] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360.

[38] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[39] Soares, L. B., FitzGerald, N., Ling, J., & Kwiatkowski, T. (2019). Matching the blanks: Distributional similarity for relation learning. arXiv preprint arXiv:1906.03158.

[40] Iannacone, M., Bohn, S., Nakamura, G., Gerth, J., Huffer, K., Bridges, R., ... & Goodall, J. (2015, April). Developing an ontology for cyber security knowledge graphs. In Proceedings of the 10th Annual Cyber and Information Security Research Conference (pp. 1-4).

[41] Syed, Z., Padia, A., Finin, T., Mathews, L., & Joshi, A. (2016, March). UCO: A unified cybersecurity ontology. In Workshops at the thirtieth AAAI conference on artificial intelligence.

[42] Barnum, S. (2012). Standardizing cyber threat intelligence information with the structured threat information expression (stix). Mitre Corporation, 11, 1-22.

[43] Mittal, S., Das, P. K., Mulwad, V., Joshi, A., & Finin, T. (2016, August). Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) (pp. 860-867). IEEE.

[44] Pawar, S., Palshikar, G. K., & Bhattacharyya, P. (2017). Relation extraction: A survey. arXiv preprint arXiv:1712.05191.

[45] Bach, N., & Badaskar, S. (2007). A review of relation extraction. Literature review for Language and Statistics II, 2, 1-15.

[46] Shubin Zhao and Ralph Grishman. Extracting relations with integrated information using kernel methods. In ACL, 2005.

[47] Kambhatla, N. (2004, July). Combining lexical, syntactic, and semantic features with maximum entropy models for information extraction. In Proceedings of the ACL interactive poster and demonstration sessions (pp. 178-181).

[48] Yarowsky, D. (1995, June). Unsupervised word sense disambiguation rivaling supervised methods. In 33rd annual meeting of the association for computational linguistics (pp. 189-196).

[49] Blum, A., & Mitchell, T. (1998, July). Combining labeled and unlabeled data with co-training. In Proceedings of the eleventh annual conference on Computational learning theory (pp. 92-100).

[50] Pingle, A., Piplai, A., Mittal, S., Joshi, A., Holt, J., & Zak, R. (2019, August). Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. In Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (pp. 879-886).

[51] Carreras, X., Màrquez, L., & Padró, L. (2003). Learning a perceptron-based named entity chunker via online recognition feedback. In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 (pp. 156-159).

[52] Montani, I., & Honnibal, M. (2018). Prodigy: A new annotation tool for radically efficient machine teaching. Artificial Intelligence to appear.

[53] Honnibal, M., & Montani, I. (2017). spaCy 3: Industrial-strength Natural Language Processing toolkit, 2017

[54] Honnibal, M., Montani, I., Van Landeghem, S., Boyd, A., DuJardin, J. (2021). Thinc 8.0.0.

[55] Smagulova, K., & James, A. P. (2020). Overview of long short-term memory neural networks. In Deep Learning Classifiers with Memristive Networks (pp. 139-153). Springer, Cham.

[56] Tzafestas, Spyros & Blekas, Konstantinos. (2000). Hybrid Soft Computing Systems: A Critical Survey with Engineering Applications.