UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

SecBERT: Analyzing reports with BERT-like models

Matteo Liberato

M.Sc. Thesis in Cyber Security
December 2022

Supervisors:
Dr. Andrea Continella
Thijs van Ede, MSc.
Zsolt Levente Kucsván, MSc.

Comittee Members:
Dr. Anna Sperotto
Dr. Lorenzo Gatti

Services and CyberSecurity Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

SecBERT: Analyzing reports using BERT-like models

Liberato Matteo

Abstract - Natural Language Processing (NLP) is a field of computer science which enables computers to interact with human language rough the use of specific software. Generic NLP tools do not work well on domain-specific language, as each domain has unique characteristics that a generic tool is not trained to handle.

The domain of cyber security, has a variety of unique difficulties, such as the need to understand ever-evolving technical terms, and, in the case of Cyber Threat Intelligence (CTI) reports, the extraction of Indicators of Compromise (IoCs) and attack campaigns.

After evaluating how existing systems addressed these issues we created SecBERT by training BERT, a state-of-the-art neural network for NLP tasks, using cyber security data.

We evaluated SecBERT using a Masked Language Modeling task, in which sentences from cyber security reports were masked and SecBERT was used to predict the hidden parts. The performance of models trained on the cyber security language-domain improved in precision by 3.4% to 5.2%, compared to the baseline of models trained on general language performing the same task.

I - Introduction

For larger organizations, defense and recovery from cyber attacks are handled by a Security Operation Center (SOC) [1], a centralized location in which a security team monitors the systems that they are working with. The purposes of a SOC include taking care of detecting, preventing and responding to cyber security incidents. Inside a SOC there are several roles, such as security operators, whose main purpose is to monitor the infrastructure of an organization and to respond to possible threats. A useful resource for the operators of a SOC is Cyber Threat Intelligence (CTI).

CTI is cyber security data that has been analyzed and evaluated by experts using specialized techniques, which account for its source and reliability [2]. CTI provides security operators with the context of the threat, which helps them in understanding the threat actor's targets, motives, and actions, analyzing it also allows security operators to remain up to date on information regarding new attacks, vulnerabilities, and possible threats. Collecting CTI allows the victim of the attack, or whoever works on their behalf, to make faster, more informed security decisions, when it comes to prevention, defense, and recovery [3].

However, CTI is being released and updated every day, so it is important to develop methods for the automation of cyber security knowledge extraction, to make it easier for experts to access and share [4]. A system to automate the knowledge extraction from CTI reports would improve efficiency in SOCs, allowing faster extraction and sharing, which would be useful for preventing and defending against cyber attacks. CTI reports are written in natural language, which is not directly understandable for a computer. Natural Language Processing (NLP) tools could be used in order to bridge this issue and create such a system.

Natural Language Processing is the field of computer science that focuses on the interaction between computer software and human language, in both written and spoken form. NLP works combining the use of machine learning and deep learning models, models that simulate the behaviour of human brains, learning from large amount of data through the repetition of a task. In the context of machine learning tasks are used to train and evaluate models, forming a prediction based on the data provided to the model. NLP tools have proven valuable for generic text on a variety of tasks, including translating text from one language to another and interpreting voice commands. The problem is that in specialized domains, such as cyber security, generic-language tools perform worse [5]. The tasks related to automatic knowledge processing of CTI face many of the generic challenges of NLP, such as information retrieval and the correlation of sentences, made harder by the use of field-specific terminology. These challenges hinder the use of NLP tools in the domain of cyber security, where they could be used to automate or assist workers in certain tasks, including the extraction of information from CTI reports. Our purpose is to help in the creation of an NLP tool that is suitable for security-specific language. Our hypothesis is that by using a domain-specific model we can achieve better results than a generic model.

For this purpose we work with BERT [6], a neural network which represents the state-of-the-art in many NLP tasks. We create SecBERT, a version of BERT trained on CTI reports and cyber-security-specific language. We experiment with SecBERT, creating different versions built with the same process but different datasets, in order to see the variation in the performances of the model. The results show that, compared to a baseline generic-language model, it is possible to improve the performances of said model in the execution of NLP tasks on domain-specific texts.

Contributions To verify our hypothesis, that a domain-specific model can achieve better performances than a generic model, starting from BERT[1], we trained SecBERT. SecBERT is a neural network specialized in cyber security, which could then be used as the foundation for specialized versions of tools used for tasks such as knowledge extraction and information processing.

[1]https://huggingface.co/docs/transformers/model_doc/bert

In short, in this work:

1. We analyze the precision of the existing NLP model BERT in predicting masked tokens in sentences taken from CTI reports.

2. We propose SecBERT, a variant of BERT which is trained specifically on technical CTI text, compared to BERT.

3. We create variants of SecBERT, differentiated by their training dataset, to observe how the dataset composition affects the precision of the model.

4. We evaluate the performance of both BERT and SecBERT on an MLM task and show that a generic NLP tool trained on specific technical language can obtain better performances than an equivalent tool trained only on generic-purpose language.

II - Background

BERT. A language model is a model built to estimate the probability distribution of linguistic units, such as words or sentences. BERT itself is a neural network architecture, and within this architecture various parameters can be set. Some of the parameters in a transformer are the number of encoder layers, the hidden size and the number of self-attention vectors. The number of layers is the amount of neurons clusters, which provide the functionality of the neural network. The hidden size is the number of neurons inside the neural network, and the self-attention head is the number of times that the self-attention module repeats its computations in parallel, improving the encoding of relationships between words. In its original configuration BERT has 12 encoder layers and 768 hidden layers, but other configurations were introduced by the creators of BERT, such as BERT Tiny, whose use will be analyzed in the Evaluation section.

BERT was created with the idea of building a generic language model, which could later be optimized on specific tasks. BERT is defined as generic because after the pre-training phase, in which the model is trained on a generic language task, the model can go trough a second step, in which it is fine-tuned on a specific task. The idea is that after the pre-training, in which the neural network is trained to perform an initial task, the newly obtained model is used as a starting point for a new task, the down-stream task. An example of downstream task that BERT can be fine-tuned for is Question Answering (QA), in which given a document and a question, the model can retrieve the answer from the document[2]. In specializing the model for the downstream task, the fine-tuning phase starts from an advantaged position, because BERT learnt the general language patterns during the pre-training phase.

NSP. BERT uses two tasks in the pre-training phase, Next Sentence Prediction (NSP) and Masked Language Modeling (MLM). NSP is a task used to train the model to understand the relationship between sentences. Given a corpus, the NSP task consists in choosing two sentences, and having the model determine whether the second sentence follows the first one in the corpus, which it does in 50% of the cases during the training phase. Training BERT to understand term dependencies across different sentences is important for downstream tasks such as Question Answering and Natural Language Inference (NLI), a task in which the model, given an premise, determines whether a given hypothesis is true, false or undetermined[3].

MLM. The other task used in the pre-training phase is MLM, which is executed on a text by dividing it into tokens [7], the input unit of BERT, then masking at random a percentage of the input tokens. Each token might represent a whole word or part of one. The model then predicts the masked token, producing a probability distribution for their real value, based on the information contained in the rest of the sentences. By predicting masked words, BERT learns correlation between words in sentences. The model predicts the masked tokens based on other tokens in the sentence, training the bidirectionality, meaning that BERT can consider the context both before and after the masked tokens.

Before BERT, standard language models were trained left-to-right or right-to-left, meaning that a model could only see half of the context surrounding a token. Trying to see the entire surrounding context with the techniques used before BERT would allow each word to see itself, creating a biased model [6]. In the original BERT paper, 15% of the tokens were masked. Once the output of the model is generated, in the form of vectors, the part of those vectors that correspond to the masked token are fed into a softmax function, which converts vectors into a probability distribution[4]. The probability is distributed across the model's vocabulary, which is the list of tokens recognized by the model, and the list of predictions for each token is determined by this probability distribution. The vocabulary is built by the tokenizer used to train the model, which divides the input sentences of the training dataset into a list of tokens, whose length is set at 30,522 in standard BERT models.

III - Approach

Overview. Our purpose is to create a neural network that could improve the performances of NLP tools when applied to cyber security data. To accomplish this purpose, we design a task which we use to evaluate the performances of the models. We decided to focus on MLM instead of NSP, we took this decision due to previous research into ROBERTA [8] and ALBERT [9]. These researches show that NSP is ineffective as a task when compared to MLM, with the removal of it matching or slightly improving performances in downstream tasks[5].

Our approach follows a series of steps, shown in Figure 1, starting from the tokenization of a group of sentences in batch, meaning that a group of sentences are taken as input and each of them is split into tokens. After the tokenization, we apply a mask

[2]https://huggingface.co/tasks/question-answering

[3]nlpprogress.com/english/natural_language_inference.html

[4]https://huggingface.co/docs/transformers/main$_{c}lasses/output$

[5]https://www.thepythoncode.com/article/pretraining-bert-huggingface-transformers-in-python

to a random token in each sentence. We then pass the masked sentence to a prediction function, which uses the model to generate five possible predictions for each masked token. Once the predictions are generated, we check whether the correct token has been found. After executing this process on every sentence in the batch, we evaluate the overall performance of the model. This evaluation includes overall precision, the distribution of the correct answers over the five predictions predictions-specific precision, and execution time.

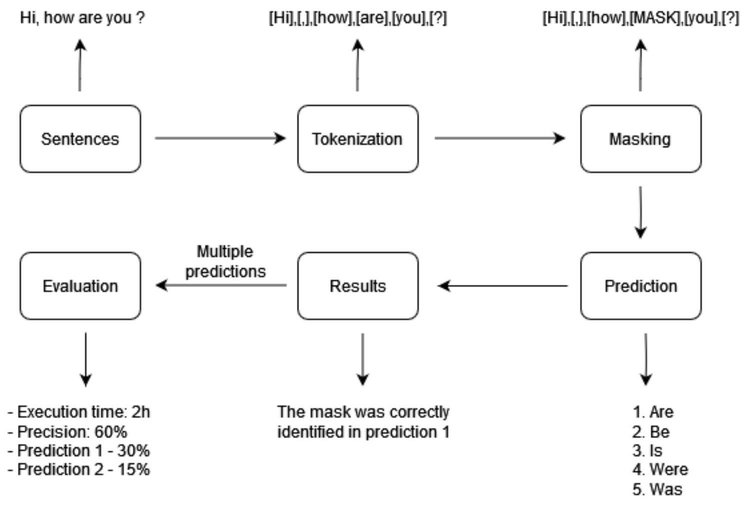

Figure 1: The steps in the MLM task

MLM pipeline. The implementation of the MLM task works by tokenizing a text and masking a pre-established percentage of the tokens, substituting them with mask tokens, which will later be predicted. The Huggingface implementation[6] of this function works, but it doesn't allow us to see the logic of the sub-functions that it uses, working trough a black-box system in which we just provide the input and receive the output. In addition the output given by the function, while relevant, is not perfectly compatible with the purpose we had in mind, since it made it harder to obtain data that could be used to create evaluation metrics, such as the precision of the model. For these reasons we decided to manually rebuild part of the functions, in order to have more control on the way they work and to have more feedback both during their executions and in the results obtained.

Tokenization pipeline. The first part of the MLM functions that we implement is a tokenizer, which executes the tokenization. To execute the tokenization we create a pipeline program, which removes most of the complex code by offering dedicated API for the task they are built for. This means that this pipeline receives the input and produces the output through the use of pre-built functions, hiding the complexity behind the production of said output.

Masking. Before masking we filter the sentences, excluding those that are too short to carry information, meaning below 5 tokens in length, and those who are too long to be real sentences, meaning over 512 tokens in length, because these sentences were occasionally found in the dataset due to errors in the parsing of the text. After the batch tokenization, each sentence is treated individually. The program then obtains the length of the sentences in tokens, and, as shown in Image 2, using the length we select a random position inside the sentence, and substitute the associated token with the mask.

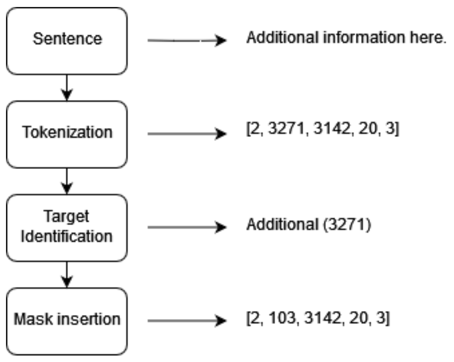

Figure 2: Tokenization and mask insertion, 2 and 3 are tokens that mark the beginning and end of a sentence, 103 represents the mask.

[6]https://huggingface.co/docs/transformers/tasks/language_modeling

The masking function then returns the tokenized sentence with the mask, the value of the token that was substituted by the mask and the index of the mask, which will be used by the other functions.

Predictions. The prediction of the masked token works using the tokenized line, with the mask inserted, and the index of the mask. From the tokenized line we extract the input ids, the token type ids and the attention mask. The input ids are the unique identifiers associated to each tokens, and they are fundamental in the prediction function. The token type ids is a binary tensor, representing where individual sequences begin and end. The attention mask is a binary tensor indicating the position of the padded indices, if padding is used, allowing the model to ignore them. Since every sequence contains a single sentence, these last two tensors are always identical to each other fore every individual sequence. We use the model and these tensors to produce a tensor containing the predictions. The index of the mask is used here to select the prediction of the masked token inside the prediction tensor, obtaining their logits. The logits are the vector of predictions that a model generates, which is usually then normalized with a normalization function, which allows us to adapt the data to the range used by the prediction function. We normalize using an exponential function, and after the normalization, we can obtain the list of the k most likely predictions for the mask. We choose to set k to five after observing that setting it to three would lower the precision of the model, due to a small amount of tokens found correctly by the fourth and fifth prediction, while a bigger k would not improve the precision significantly.

Models. The models that we used in the context of this work were all built on the same skeleton, provided by the Huggingface BERT implementation. To create these model we trained them depending on whether we wanted to use them as a generic-language comparison baseline, in which case we would use the Wikipedia dataset[7], as a CTI focused model, where one or both the CTI datasets were used, or as a model in-between, with a mixed dataset. Both types of datasets will be further explained in V - Datasets section.

IV - Implementation

The implementation of the MLM task works by tokenizing a text and masking a pre-established percentage of the tokens, substituting them with mask tokens, which will later be predicted. The Huggingface implementation[8] of this function works, but it doesn't allow us to see the logic of the sub-functions that it uses, working trough a black-box system in which we just provide the input and receive the output. In addition the output given by the function, while relevant, is not perfectly compatible with the purpose we had in mind, since it made it harder to obtain data that could be used to create evaluation metrics, such as the precision of the model. For these reasons we decided to manually rebuild part of the functions, in order to have more control on the way they work and to have more feedback both during their executions and in the results obtained.

Tokenization and Masking. We built a program that would allow us to experiment with various characteristics of the tokenization process. We recall from the III - Approach section that sentences must be tokenized in order to be used as input for SecBERT. To this end, we used SpaCy to identify individual sentences, and the Huggingface tokenizer to create tokens. The tokenization process was initially limited to individual sentences, and was implemented with tensors as the output, a data structure that can store data in different dimensions. We used PyTorch tensors, which we considered more adapt to effectively retrieve data such as the tokenized sentences, the input ids, and to create the dictionary used in the predict function, which is the list of all the unique tokens recognized by the model. We experimented with various parameters of the tokenization such as the padding, truncation, and the different output formats, starting from simple arrays and later moving to tensors. We implemented a randomized choice for the mask token, which was selected in each sentence between

[7]https://en.wikipedia.org/wiki/Wikipedia:Database_download
[8]https://huggingface.co/docs/transformers/tasks/language_modeling

all the tokens that we knew to represent parts of the sentence, avoiding tokens that are used to define the structure of the sentence, such as padding tokens and separations tokens.

We also implemented some filtering in the sentences that were passed to the prediction function. If the tokenized sentences were over 512 tokens long they were discarded, since that is the maximum length that BERT-based model can accept. These sentences were discarded rather than being truncated because the only sentences that were that long were those containing URLs in their entirety, or those generated from parsing errors, both of which do not carry useful information. Sentences that were less than 5 tokens long were also removed, because we deemed them too short to contain information, and were often created from elements of the web pages being parsed by mistake, such as buttons and search bars.

The tokenization process worked, handling each sentence individually one after the other. We started working on implementing batch tokenization inside the program, meaning that group of sentences would be tokenized together, to improve efficiency. Data in a batch is saved in a list of strings, each string containing a sentence, and the list is processed by the tokenization function as a whole.

Prediction. After the tokenization phase, the program works on individual sentences. The masked sentence is passed to the prediction function, from which we obtain the input ids. The token type ids and attention mask ids are generated from the input ids. These three tensors are used to build a dictionary, where each of them is associated to their data type, in a format compatible with BERT-based models. This dictionary is then fed to the model to extract the logits, giving a tensor as output, containing predictions for each token. We slice this tensor to obtain a smaller tensor containing only the values corresponding to the mask, called mask token logits, and we apply a normalization function. We use an exponential function, e^x, applied to every component of the tensor, where x is the component itself. We normalize in order to format the output to the range used by the prediction function. We then use both the mask token logits and the dictionary to obtain the predictions tensor. From this last tensor using the function *top k*, contained in *pytorch*, we produce the list of most likely predictions for that mask.

Evaluation. We implemented the evaluation functions, built on a system of arrays and variables which stored the masked tokens and the predictions during the execution. The five predictions for each sentence are compared to the corresponding correct value for the mask, which we save in an array containing all the masked tokens. If the tokens match, the variable counting the number of sentences correctly predicted is increased. We determine the precision of the model as the number of sentences correctly predicted divided by the number of sentences who have been masked One array keeps track of the distribution of correct predictions between the five generated for each sentence, with five cells whose value is increased whenever the corresponding prediction is correct. This system of array is also used to generate the confusion matrices for each prediction, which represents the distribution of the correct answers over the five possible predictions, detailed to the individual sentences.

Models. The program we built to generate the models used in this work follows a series of steps[9]. The first part of this process involves preparing the dataset, creating a split for the training and testing subsets. We use 80% of the dataset for training and the remaining 20% for testing. The split dataset is then saved in two separate files, ready to be used for testing the model and to train the tokenizer.

The next step is the creation of the tokenizer[10], in doing so we establish a series of parameters. The first one of these parameter is the size of the vocabulary, the set of all unique tokens recognized, which is set as 30,522, the standard size for BERT-based model. Models usually have a vocabulary size smaller than 50,000, especially if they are built to work only on a

[9]https://www.thepythoncode.com/article/pretraining-bert-huggingface-transformers-in-python
[10]https://huggingface.co/docs/tokenizers/api/tokenize
[11]https://huggingface.co/docs/transformers/tokenizer_summary

single language, because a bigger size would increase both memory usage and execution time[11]. The second parameter is the list of special tokens, which are not part of the vocabulary and have specific functions, this list is comprised of [UNK],[CLS],[SEP],[PAD] and [MASK]. [UNK] stands for "unkwown", a token which designates a word that is not in the vocabulary. [CLS] is the token that represents the class of the input, it appears at the beginning of each sentence. [SEP] is the separation token, which separates two different sentences in the same input. [PAD] is the token used to pad the input, which is processed in batches which should all have the same length. The final token is [MASK], which is set in place of the token that has been masked in a MLM task[12].

The program tokenizes the dataset with the tokenizer we built, building it's dictionary, and the model is then loaded and trained. In the training we started from BERT as a base, we passed the tokenizer to it and we loaded the MLM pipeline. In doing so we initialize the training arguments, setting parameters such as the batch size and the number of epochs, which is the number of times that the model works through the entire training dataset. We set both of these values to 10, the default for BERT. Other parameters are information for the creation of checkpoints, such as their number and the amount of steps, set respectively to 3 and 1,000. This structure was used to train each model, changing the datasets used to train them.

V - Datasets

Dataset		**Size in lines**	**Source**
CTI	Mitre ATT&CK	489k	Mitre ATT&CK
	Chainsmith	156k	Chainsmith dataset
Wikipedia		1000k	English Wikipedia

Table 1: The datasets used in this project, with their size and the sources used to build them

While working on the creation of the MLM task we created datasets that will be used in the evaluation of the models. These datasets, visibile in the table1, are classifiable in two big categories, those which contain information extracted from CTI data files, and those which contain sentences extracted from Wikipedia. The creation of these datasets allows us to measure the performance of the models in executing the MLM task, focusing on the difference in precision when it comes to predicting tokens from both generic language sentences and domain-specific sentences.

Wikipedia. The Wikipedia dataset was created to be used as a baseline for generic language. The entirety of Wikipedia is available as a dump to download[13]. Initially we tried to extract information from this file, isolating only the text part. It was successful, but the file was not ordered and cleaned properly, with many tags containing metadata still found in the extracted text. These include the URL address of the page that had been extracted, links to other pages mentioned in the article and elements used in the construction of the page, such as the caption of images contained in it or the titles of the subsection that formed the page. The file's handling was also complicated due to the size of the entire Wikipedia dataset, which consisted of one compressed 20GB file containing an estimated 1.4 billion lines of text. We also considered that the amount of data in the Wikipedia dataset far outnumbers the number of CTI reports available. This means that training a model with the entire Wikipedia dataset and the CTI dataset would dilute the cyber-security-specific data. We also concluded that, even though using a smaller dataset made it impossible to build models that could be immediately useful, we could still experiment with models trained with less data to see how the performances changed. This information could then be used to guide future work on larger models. For these reasons, we took a different approach to extracting information from Wikipedia after determining that extracting the entire dataset was not ideal.

In order to balance our datasets, we randomly selected a number of Wikipedia articles. As an addi-

[12]https://huggingface.co/docs/transformers/model_doc/bert
[13]https://en.wikipedia.org/wiki/Wikipedia:Database_download

tional benefit, this vastly reduces the time requirement, since the extraction of the entire Wikipedia dataset would take weeks, due to the sheer amount of data. We decided to extract a million lines, this would still provide an extensive base, bigger than the CTI dataset, while removing the problems that came with working with the entire Wikipedia dataset. For this reason we created a program that, connecting with the Wikipedia publicly accessible API, downloads a number of random articles out of all the articles in the English Wikipedia. These articles are then cleaned of all the unnecessary information, such as pictures, titles, paragraph divisions, labels and metadata, leaving only the content of the textual parts of the article. This process allowed us to create datasets of generic language to use in the evaluation of our task.

CTI. The creation of the CTI datasets was more straightforward, we extracted data from two different group of files. The first one was previously created to build Chainsmith [10], using ten different sources, spanning between news websites covering cyber threats, blogs from anti-virus companies and the personal blogs of security experts, amounting to 14k articles containing a total of 157k lines. This dataset contains high quality data, with very few sentences that do not carry cyber security related information. The second dataset was scraped for the purpose of a project analogous to this one, based on Mitre ATT&CK, a publicly-accessible knowledge base of adversary tactics and techniques, used for the development of threat models and methodologies in the cyber security field. We scraped the Mitre ATT&CK repository[14], which consists of a list of sources where we can find the cyber threat reports referenced in the Mitre ATT&CK framework, those sources were scraped in their entirety.

The dataset obtained was then broken into lines, with each line corresponding to a sentence, and a few steps were executed in order to improve the quality of the data. After tokenizing the dataset, as explained in the IV - Implementation section, lines containing less than 5 or more than 512 tokens were removed, since they did not contain information that could be useful for this work. After this filtering process the CTI dataset amounts to 489k lines, but many lines that do not carry useful information can still be found in it, meaning that while the size of the dataset is three times the size of the Chainsmith one, more sentences are not useful for our work. These two datasets summed together amount to roughly 645k lines, each containing a sentence. With the content of these two sets of cyber-security-related data, we created three datasets, one from each source, and one that contained both of them.

For testing purposes, we also created subsets of both the CTI and Wikipedia datasets, containing 20, 100, 1,000, 10,000, 50,000 and 100,000 lines respectively, which were used to analyze the behaviour of the programs without running said programs on the datasets in their entirety, and to obtain indicative results on the performances of the task.

[14]https://attack.mitre.org/datasources/

VI - Evaluation

We want to show that language models trained over domain-specific text have better performances than the equivalent generic-language model, when applied to text in that domain, in particular regarding the cyber security domain and CTI reports. Proving this would be an important step in the creation of domain-specific tools, capable of applying NLP techniques efficiently to text in the domain of cyber security. In order to observe the way the performances of the model change based on the specific datasets used to train it, we created multiple models, each with different characteristics.

Model	Precision	
	Wikipedia	CTI
BERT	78.65%	67.06%
BERT Tiny	52.07%	40.99%
SecBERT BASE	20.21%	15.90%
SecBERT CTI	18.01%	20.62%
SecBERT FULL	20.38%	21.10%
SecBERT SCRAPE	18.96%	19.34%
SecBERT Tiny	35.02%	33.03%

Table 2: The models and their precision

For each of the models visible in Table 2, we ran our MLM pipeline on subsets of the CTI and Wikipedia dataset, according to what aspects of each model we wanted to evaluate. We repeated this process gradually increasing the size of the subset, starting from 10k lines, then 50k lines and finally 100k lines, running the MLM pipeline multiple times for each dataset size. Precisely, the pipeline was run 10 times using files containing 10k lines, 5 times using files containing 50k lines and 3 times using files containing 100k lines. This process was repeated twice for each model, once with lines taken randomly from the CTI dataset, and once with lines taken randomly from the Wikipedia dataset.

Evaluation Metrics. During the execution of the task the real value of the token that we mask is saved, the list of predictions is compared to the real value of the token, as visible in Table 3, and it is determined whether one of the predictions was correct.

Target Token	Prediction		
	Number	Value	Result
the (1419)	1	? (37)	×
	2	. (20)	×
	3	the (1419)	✓
	4	and (1444)	×
	5	, (18)	×

Table 3: Checking the validity of a prediction, number 3 is correct

The results of the prediction of every individual sentence is sent back to the MLM pipeline, which after predicting every sentence in the file produces evaluation metrics, visible in Table 4, used to calculate the performances of the different models. The main metric is general precision, meaning the percentage of sentences that had the mask token inserted in them and then correctly predicted in five attempts, compared to the total amount of sentences that the model applied the mask to. Other metrics include the number of times in which the first prediction is the correct one out of all five of them, and the same is done for the other four predictions. The MLM pipeline also creates arrays in which to save the values of every prediction made, in order to create confusion matrices representing the overall effectiveness of each of the five predictions. The final metrics are precision and accuracy applied to the individual predictions, which indicates in percentage how the correct predictions were distributed between the five attempts.

Metrics	Results
Duration:	0:17:45
Total predictions:	99,989
Not found:	65,073
Found:	34,916
Precision:	34.92%
Prediction 1 precision:	62.40%
Prediction 2 precision:	16.35%
Prediction 3 precision:	9.43%
Prediction 4 precision:	6.61%
Prediction 5 precision:	5.21%
Confusion matrix prediction 1:	[1, 1, 0, 0, 0, 1, ... 0]
Confusion matrix prediction 2:	[0, 0, 0, 1, 0, 0, ... 1]
Confusion matrix prediction 3:	[0, 0, 0, 0, 0, 0, ... 0]
Confusion matrix prediction 4:	[0, 0, 0, 0, 0, 0, ... 0]
Confusion matrix prediction 5:	[0, 0, 0, 0, 1, 0, ... 0]
Distribution of the correct answers over the 5 predictions:	[1, 1, 0, 0, 0]

Table 4: Evaluation of a group of predictions, the confusion matrix is 99.989 cells long, and the number of total predictions is below 10.000 because some sentences were discarded before the mask application.

BERT. The first model we experimented with is the basic BERT model, imported as the module *bert-base-uncased*, which was tested using both the sentences extracted from Wikipedia and the sentences extracted from the full CTI dataset. This model obtains high performances in the testing task, with an average precision of 78.65% in the prediction applied to sentences extracted from Wikipedia. The performance worsens in predicting the masked token in sentences obtained from the CTI database, with an average precision of 67.06%. This drop in the quality of the prediction is indicative of the problem that originated this research, the lower performance of generic NLP tools applied to domain-specific text.
All of the subsequent models, except BERT Tiny,

were constructed using the modeling program that was created during this research, and they have been evaluated with our MLM task. The structure of each model is similar, with the main difference found in the dataset used, and in the division of the aforementioned dataset into the training and testing sets.

SecBERT CTI. The first prototype of SecBERT, referred to as SecBERT CTI, was trained using only the CTI dataset, in its entirety. The performances of this protoype in the testing experiment are considerably lower than the base version of BERT. We tested it with sentences extracted from Wikipedia and sentences obtained from the training itself, which creates a testing subset from the dataset used. In the prediction of sentences extracted from Wikipedia, it obtains an average precision of 18.01%, and on the CTI dataset it averages a precision of 20.62%. The decrease in precision is ascribable to the lower amount of data and the absence of generic-domain language in the training phase of the model, which was focused entirely on the CTI dataset. The precision of the model is overall lower, but there is a noticeable improvement in the performances when it comes to predicting the masked token in a sentence whose language is domain-specific for cyber security. These results supports the idea that a generic NLP tool trained specifically to work in a domain-specific context can obtain better results than an equivalent generic tool.

SecBERT BASE. Having re-evaluated the scope of the project, we decided that it would be correct to re-evaluate the baseline for the results of these experiments. In order to establish a baseline for these models we trained SecBERT BASE, a model trained exclusively on the Wikipedia dataset. The model obtained is conceptually similar to the base version of BERT, but also shows the limitations that are inherent in SecBERT due to the technical obstacles to creating a model trained on the entire Wikipedia corpus. For this reason it provides the lower level of the baseline when it is used to predict masked tokens in the context of cyber security domain-specific language. We tested this model on sentences taken from both the Wikipedia dataset and CTI dataset, similarly to the original BERT. This model achieves an average precision of 20.21% in the evaluation of sentences extracted from random Wikipedia articles, as opposed to an average precision of 15.90% when applied to lines obtained from the CTI dataset. The relationship between these two results is in line with the results obtained by BERT, with an higher precision when the model is applied to generic-purpose language, and the difference in the baseline is attributable to the difference in the size of the training dataset.

SecBERT FULL. The second SecBERT prototype, referred to as SecBERT FULL, was created using both the CTI dataset in it's entirety, and the Wikipedia dataset. It was built as an evolution of SecBERT CTI, to observe whether the addition of non-domain-specific language to the dataset used in the training of this version of SecBERT would improve the performance of the model. To test this model, we had to create a testing subset from the CTI dataset before the creation of the model. We had to create the testing set manually because the testing set created by this model would have been an hybrid of both the datasets used, meaning that we could not have used it to test performances on CTI and Wikipedia separately. The performance of this model is similar to the performance of SecBERT CTI, with an average precision of 20.38% for the evaluation of generic-domain sentences, and an average precision of 21.10% for sentences extracted from the CTI dataset. The small improvement in performances can be attributed to the addition of the data obtained from Wikipedia, which improved the precision on non-domain-specific language, supporting the idea behind the creation of SecBERT FULL.

SecBERT SCRAPE. The third SecBERT prototype, referred to as SecBERT SCRAPE, has been built using the lines extracted from Wikipedia and the part of the CTI dataset that was scraped for the purpose of this project. This model was created to observe whether using a single source for the domain-specific data could improve performances, as opposed to the multiple sources used for the previous experiment. This model, when tested on CTI data obtained exclusively from the Chainsmith dataset, obtained an average precision of 18.96%, lower than the

19.34% obtained on average applying the model to the mixed CTI dataset, meaning that the use of separated sources in this case did not improve performances. The performances of this model are worse than the performances of SecBERT FULL when it comes to predicting the mask token in sentences extracted from either dataset. The possible reasons for these results might be multiple. The reason for the drop in performances on the Wikipedia dataset might be due to the smaller size of the training set, regardless of the provenience of the dataset. For the drop in performance over the CTI dataset, it might be the fact that the training was done using a small part of the whole CTI dataset. This part might be too small to significantly improve the performances on sentences taken from the CTI dataset compared to those from the Wikipedia dataset.

BERT Tiny & SecBERT Tiny. Lastly, we decided to change the base model used in creating SecBERT, and we tried using BERT Tiny [11] instead of BERT. BERT Tiny is the smaller pre-trained BERT variant, built with 2 encoder layers stacked on top of each other and 128 hidden layer, as opposed to the 12 encoder layer and 768 hidden layers of BERT. The smaller number of layer lowers the overall performances, BERT Tiny, used as its original implementation, analogously to BERT, obtained a 52.07% average precision in the task when applied to sentences from the Wikipedia dataset, and 40.99% on sentences from the CTI dataset. This lower performances, compared to the original BERT, are accompanied by a much faster execution time. We experimented with this model to see if this model would work better with the reduced size of the dataset used in this project.
We built SecBERT Tiny analogous to SecBERT FULL, meaning that both the Wikipedia dataset and the entire CTI dataset were used in the training, and we tested it using the sentences from Wikipedia and a subset of the CTI dataset that we prepared before the training. This model averages a precision of 35.02% on the Wikipedia dataset and 33.03% on the CTI dataset. We can see a loss of precision from BERT Tiny to SecBERT Tiny, similar to the loss of precision from BERT to the SecBERT models, ascribable to the same reason, the lower amount of general-purpose language data used in the training. Despite the loss of precision from BERT Tiny, this model is the better performing version of SecBERT. The reason for this might be found in the fact that the smaller size of the model, in terms of layers, works better with the smaller amount of data used to train it [12].

Summary. Overall, the results obtained with these experiments are promising, we saw a fall in performances in the baseline, but an improvement when adapting to the new baseline. Considering the new baseline, the data obtained seems to support our theory, that training language models over domain-specific data improves performances compared to equivalent generic-language model. We can see this in the improvements of SecBERT FULL and SCRAPE over SecBERT BASE, although it seems that to create NLP tools capable of surpassing the performances of BERT, even on domain-specific text, it would require an enormous amount of training data.

Runtime Performance. The amount of time required to complete the training changed depending on the model and the size of the datasets. The fastest was SecBERT Tiny built on BERT Tiny trained with the full CTI dataset, which took 140 minutes, the slowest was SecBERT trained with both the entire CTI dataset and the Wikipedia dataset, which took 3 days. SecBERT Tiny was also the fastest model when it comes to execution time, with the task being executed on 50k lines in 10 minutes, as opposed to the other models taking between 1 and 3 hours.

Model	Size	Runtime	
		Wikipedia	CTI
BERT	50k	01:52:37	01:00:34
	100k	01:09:11	01:17:48
BERT Tiny	50k	00:08:48	00:06:19
	100k	00:16:25	00:17:42
SecBERT BASE	50k	02:30:01	01:38:36
	100k	06:16:02	06:34:49
SecBERT CTI	50k	00:59:32	02:14:28
	100k	01:17:59	06:11:15
SecBERT FULL	50k	02:32:33	01:46:56
	100k	06:28:29	06:13:29
SecBERT SCRAPE	50k	02:48:16	01:56:26
	100k	06:13:37	06:18:36
SecBERT Tiny	50k	00:06:32	00:04:30
	100k	00:18:34	00:17:39

Table 5: The models and their average runtime on batches of 50.000 and 100.000 sentences extracted from the CTI and Wikipedia datasets.

VII - Discussion

The work done during this research resulted in data that supports our hypothesis, that by using a model trained with domain-specific language we can achieve better results than an equivalent non-domain-specific model. This data was obtained trough an initial research phase, followed by the creation of an evaluation task, the construction of two datasets, the creation of several models and then experimenting with the task to evaluate the models.

Resources. An important aspect in the proceeding of this work was the use of resources. Initially we wanted to train a neural network in a way similar to the training that the original BERT underwent to, using the entire textual part of Wikipedia. We worked on the creation of this task on a personal laptop, with an Intel i7-7500U CPU and 8GB of RAM, these technical constraints forced us to change the scope of our plans. We realized that this process would have taken weeks on this machine, and that would have just been the time necessary to obtain a general language baseline. This time scale for the creation of a model would also mean that we would have been able to create just a single model focused on cyber security language. We would have not been able to experiment with different compositions for the dataset or any other parameter for the creation of the model, and any possible mistake in the creation of said models could have costed us weeks of work. On top of these issues, the use of the entire Wikipedia dataset would have created a strong imbalance in the amount of generic-language data compared to cyber-security-related data, diluting the information found in the CTI dataset.
These are the reasons why we decided the scale of the experiment should be smaller than the creation of full models. For this purpose we extracted a million sentences from random Wikipedia articles, to be used as the generic-language dataset. This change meant that the performance baseline of these models fell. Despite the loss in absolute performance, the experimentation with these models allowed us to see the improved performances that these model presents in working with cyber-security-specific language, compared to a generic-purpose language baseline.

Possible Future Developments. During this process we were presented with a number of possible future developments that could not be investigated due to constrictions dictated by time and resources. These possible future development regard various aspect of the work, which could be explored more in depth by a larger study which does not have the time and resource constrictions that our study did.
The first aspect regards the CTI data used in this project. A bigger study could explore in which way the quality of CTI reports scraped influences and reflects the quality of the model generated. For this project we worked with two CTI databases, one built for the creation of Chainsmith, and one scraped for a project analogous to this one. The Chainsmith dataset presents a smaller amount of data, but the quality is higher, with very few non-relevant sentences or sentences that have been parsed improperly. The second dataset is three times larger, but the quality of the data is lower. Many of the sentences are non-relevant, containing parts of the sites where the information was gathered from, such as links for sharing the content, search bars and other HTML elements that do not contain CTI data. Also some of the text was not scraped properly, resulting in strings of char-

acters that do not form coherent sentences, probably due to the use of different fonts, encoding and other factors that interfered with the scraping process. So it would be interesting to explore whether a more polished dataset of similar or even bigger size than those used in this work would improve performances of the model.
The second aspect is the amount of data used for training the models. Due to time and resources constrictions we decided to use a subset of Wikipedia as the general-domain language to train the models, so it would be interesting to explore how would the use of more data impact the performances. We suspect that increasing the size of the general-domain language data would generate diminishing returns. On one side the models would improve their understanding on general language, as visible from the gap in performances between BERT and SecBERT, but on the other side the CTI-specific data would end up being diluted in the training process. Ideally to mitigate this last problem we would have a much larger CTI dataset, but as far as we know such a dataset does not exist, so possible ways to create it should be explored.
Another aspect that could be explored more is the creation of the models themselves. In creating the model a single blueprint was followed every time, changing only the dataset used and a few variables which reflected the technical capabilities of the device used, such as the batch size used during the training. Other variables could be changed, such as the vocabulary size or the way the data is split between training and testing data. This could be explored after the creation of the larger dataset mentioned before, using subsets of it too observe the performance of models that use different subsets of the same size. Another possibility is to investigate how much of the performance difference between models was due to data quality and how much was due to data quantity. This could be investigated after the creation of the larger dataset mentioned earlier, using subsets of it to observe the performance of models that are created using different subsets of the same size.
We also saw interesting results when using BERT Tiny as the basis for the model, instead of BERT Base. SecBERT Tiny, built with the entire CTI and Wikipedia datasets, outperformed every other SecBERT model. We speculate that the reason for these performances might be derived from the reduced size of the model, which might work better with the smaller amount of data, compared to the dataset used to build BERT Base. If this hypothesis was found to be correct, it would mean that increasing the size of the dataset would produce diminishing returns in the improvement of the model. Presumably other SecBERT models would outperform SecBERT Tiny when trained on the same dataset, with datasets larger than a certain size. It would be interesting to experiment with these same models and more datasets of different sizes to verify these hypothesis. It would be also interesting to work with other models as a basis, such as BERT Small and BERT Large, to see what kind of performances they would produce.
Finally, another aspect that could be explored is the integration with other tools in order to create specialized versions of said tools, which could now perform better in the domain of cyber security.

VIII - Related Work

We decided to approach this problem by building on BERT, a tool for general text, to create SecBERT, a specialized version that focuses on CTI data. Other researchers tried different approaches to improve the state-of-the-art in applying NLP to cyber security text.
TTPDrill [13] is one of the tools built to work specifically with cyber security language, introduced in 2017, it has been outperformed, but it served as a basis for many other state-of-the-art tools used in cyber security. TTPDrill was created to extract MITRE ATT&CK patterns from cyber security reports, mapping MITRE techniques, tactics and phases. TTPDrill does this using a set of NLP rules to identify a threat action, which is then divided into specifications, such as verb, object, and action goal. These specifications are then used to construct an attack pattern mapping these actions to the sequence of events that constitute a cyber attack.
Another tool is **Chainsmith**, created in 2018, it is built to extract Indicator of Compromise (IOCs) from a report and associate them to a 4-stage model, ob-

taining the attack campaign. IOCs are evidence that a system might have been compromised by a cyber threat, and the idea behind this system is to connect multiple sources, such as reports and field measurements, to reconstruct all the phases of the described attack, aiming to produce new insights on attacks, analyzing how the parts of the attack might be connected in non-immediately noticeable ways. It works using a series of functions, such as syntactic parser, semantic parser and IOC classifier. These functions identify a possible IOC using regular expressions, and using the parsers on the surrounding context of the sentence they determine whether the candidate is an IOC and the corresponding stage in the attack campaign.

iACE [14] is a tool presented in 2016, based on the observation that, in a technical article, IOCs are usually described in a consistent way, re-using the same sentence structure and terms. Identifying these terms iACE isolates possible IOCs, and then converts them into relationship graphs to identify the relationship with other terms, identifying relevant elements.

The same idea was used by **AttacKG** [15], which works using Cyber Threat Intelligence (CTI), information regarding cyber attacks and the techniques used to perform them, presented as structured, machine-digestible IOCs, or reports written in natural language. AttacKG was created to extract structured attack behavior graphs from reports, which show the IOCs and the way they interact with each other, converting unstructured information into structured information. AttacKG was built to outperform both iACE and Chainsmith in the extraction of IOCs, it works identifying the attack technique using the MITRE ATT&CK knowledge base, which is continuously updated to keep up with the evolution of techniques and threats. AttacKG handles IOCs in a similar way to iACE, with parsers that are capable of recognizing domain-specific terms, but it also gather information that is not contained in IOCs (such as the word ”email”, that can indicate a phishing attack) and aggregates multiple sources of information. This is more powerful than using just IOCs, since it manages to add the context obtained from non IOCs entities. The behaviour graphs enhanced with this information are then called technique knowledge graphs (TKGs), they represents the attack chain, and they are built by aggregating extracted information according to MITRE ATT&CK technique. A system to automate the knowledge extraction from CTI reports using NLP techniques would greatly improve efficiency in SOCs, especially if that information were to be converted in the form of MITRE ATT&CK concepts. While AttacKG is powerful, it's focus is on uniting information coming from multiple sources in order to build the graph and specifically identify the attack technique, which is outside of our scope.

An important characteristic of these tools is that they are specifically designed for working with cyber-security-related data, which means that issues relevant to dealing with cyber security reports, such as IOC recognition, are already taken into account. However, because they were designed specifically for this purpose, these tools lack the flexibility that a BERT-based approach can provide, as its performance is not specifically centered on structure recognition. Another limitation of the other tools is the lack of bidirectionality, which BERT possesses, which improves understanding of the context surrounding IOCs. For these reasons, we choose to focus on BERT.

IX - Conclusion

NLP tools have proven useful for generic text on various tasks, but their efficacy is diminished by domain-specific terminology in CTI tasks. Generic NLP challenges, such as information retrieval and sentence correlation, are exacerbated by domain-specific terminology in CTI tasks. These obstacles hamper the use of NLP tools in cyber security to automate or assist workers in specific tasks, such as information extraction from CTI reports. Our goal is to assist in the development of a domain-specific NLP tool, and our hypothesis is that using a domain-specific model yields better results than using a generic mode.

We developed SecBERT, a BERT-based model trained on CTI reports and cyber-security-specific language, and found it outperforms baseline generic-language models in NLP tasks in the domain of cyber security. Despite the constrictions on time and resources, the results obtained were promising:

SecBERT outperformed BERT in masked language modeling for cyber-security-specific language, showing that domain-specific models can improve the performances of a generic-language model in the execution of NLP tasks on domain-specific texts.
SecBERT could serve as the foundation for specialized tools tasks such as knowledge extraction and information processing in the context of cyber security. Larger datasets, different model compositions, and training parameter experimentation could improve performances. This research paves the way for additional research into the development of NLP tools to automate and assist workers in specific cyber security tasks.

References

[1] Renaud Bidou. Security operation center concepts & implementation. *avalable at http://www. iv2-technologies. com*, 2005.

[2] Md Sahrom Abu, Siti Rahayu Selamat, Aswami Ariffin, and Robiah Yusof. Cyber threat intelligence–issue and challenges. *Indonesian Journal of Electrical Engineering and Computer Science*, 10(1):371–379, 2018.

[3] Wiem Tounsi. What is cyber threat intelligence and how is it evolving? *Cyber-Vigilance and Digital Trust: Cyber Security in the Era of Cloud Computing and IoT*, pages 1–49, 2019.

[4] Casey Hanks, Michael Maiden, Priyanka Ranade, Tim Finin, and Anupam Joshi. Recognizing and extracting cybersecurtity-relevant entities from text. 2022.

[5] Hao Cheng Michael Lucas Naoto Usuyama Xiaodong Liu Tristan Naumann Jianfeng Gao Hoifung Poon Yu Gu, Robert Tinn. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare*, 3(1):1–23, jan 2022.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. cite arxiv:1810.04805.

[7] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[9] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[10] Ziyun Zhu and Tudor Dumitras. Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 458–472. IEEE, 2018.

[11] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in nli: Ways (not) to go beyond simple heuristics, 2021.

[12] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962, 2019.

[13] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, page 103–115, 2017.

[14] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 755–766, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Zhenyuan Li, Jun Zeng, Yan Chen, and Zhenkai Liang. Attackg: Constructing technique knowledge graph from cyber threat intelligence reports, 2021.