THE UNIVERSITY OF CHICAGO

TRANSFORMER MODELS FOR PROTOCOL ANALYSIS: A CASE STUDY IN TLS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCE
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
ANDREW CHU

CHICAGO, ILLINOIS
NOVEMBER 9, 2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Achieving generalizable, detailed methods for protocol analysis is difficult: Existing tools and approaches are not protocol-agnostic, and are typically specifically tailored for analyzing a specific protocol. Further, for sub-tasks of protocol analysis such as detecting and mitigating misconfiguration, tools detect errors only individually, and cannot provide insight into common patterns across protocol deployments that cause misconfiguration at large. In this paper, we perform a case study using transformer models for protocol analysis, specifically towards detecting and mitigating misconfiguration in the Transport Layer Security (TLS) protocol, examining if these models can (1) effectively detect TLS misconfiguration and (2) provide a different perspective towards more generalizable and proactive reasoning of errors in TLS implementation. To do so, we train a BERT-based model that learns semantically meaningful numerical representations (embeddings) for sites' TLS server configurations. The model we develop contextualizes the fine-grained differences in various implementations of TLS, achieving $95+\%$ accuracy for standard BERT pre-training tasks. We then fine-tune our model to classify (1) properly configured and misconfigured TLS implementations and (2) the reasons for misconfiguration; we also visualize the resulting prediction embeddings. We observe distinct clusters in these visualizations, and verify that these clusters indeed represent different reasons for proper configuration, or misconfiguration.

CHAPTER 1

INTRODUCTION

Large language models are pervasive in today's society, with interactive sessions such as OpenAI's ChatGPT and Google's Bard allowing users to query and receive answers for a diverse set of general downstream tasks. Although they are highly developed and tailored, the underlying architecture of the models for these systems (GPT, LaMDA) stems from the transformer model (Vaswani et al., 2017), where each word in a given input is represented by one or more *tokens*, and each whole input is represented with an *embedding*, a high dimensional representation of the input that captures its semantics. *Self-attention* is then used to assign a weight score for each token in an input based on its importance to its surrounding tokens. This mechanism allows transformer models to capture fine-grained relationships in input semantics not captured by other methods (e.g., traditional neural networks), and produce output not based on a fixed feature sets, but all inputs.

The transformer approach for reasoning about data transfers well to problems that (1) can be represented with sequences of structured input and (2) have large input spaces that any one feature set cannot sufficiently represent. In computer networking, one such problem is that of *protocol analysis*. In this case, manual parsing and analysis of network traffic can be tedious and impractical. Attempting to apply other machine learning techniques to the problem is similarly difficult: exhaustive mapping to represent byte offsets/header fields and their data types for all protocols, as well as considering all values a field may take, is typically far too extensive for any one group of features to capture comprehensively. In particular, the protocol analysis task of detecting and mitigating misconfiguration seems particularly well suited to transformer models, where small nuances, interactions, or misinterpretations of protocol settings can cause complicated corner cases and unexpected behavior that may be challenging to encode with static rulesets or formal methods approaches.

Misconfiguration may occur in many networking protocols, and such misconfiguration can have varying consequences. For instance, an configuration or implementation error in a protocol such

as dynamic host configuration protocol (DHCP) could result in inconvenience to users on a local network. On the other hand, misconfiguration in the Transport Layer Security (TLS) protocol can have serious security and privacy implications, with broad implications for a wide range of Internet users. Unfortunately, TLS misconfiguration is common, with the flexible nature of the protocol leading to difficulty in correct implementation. With over a thousand possible parameters to select from for 19 configurable fields, properly setting TLS may be an understandably difficult or confusing task (IANA). Existing, state-of-the-art tools and resources for mitigating TLS misconfiguration work to *reactively detect* implementation errors to allow site operators to deploy fixes that correct the flaws (IBM; nsacyber; drwetter; Qualys, b). Unfortunately, these mechanisms are only effective if a site operator is aware of possible vulnerabilities and has the wherewithal to apply these techniques; the tools also only detect known, existing misconfigurations and vulnerabilities. Unfortunately, historical trends in patching TLS vulnerabilities show tools fall short, and operators are not so proactive. For instance, in October $2015, 33\%$ of the top $150{,}000$ Alexa-ranked sites still supported SSLv3, despite disclosure of the POODLE vulnerability a year prior (Qualys, a; Program). As such, creating better *proactive* methods to prevent TLS misconfiguration by understanding trends in why users misconfigure TLS to begin with, would be valuable for improving the security of the internet at large.

In this paper, we aim to evaluate the place of transformer models in reasoning about protocol misconfiguration, specifically in the TLS protocol. Our goal is not to replace current leading methods of detecting TLS misconfiguration, but rather to: (1) verify if transformer models can be effective in such a task, and (2) examine if the embeddings transformer models use for prediction can provide a different perspective towards more proactive reasoning and comparison of errors in TLS implementation.

We approach these aims by training a transformer model that learns the semantically meaningful relationships between fields in TLS server handshake messages. We find our model understands our parsed representations of TLS server handshake messages well, yielding 98.8% and 95.9% accuracy

in the standard masked-language modeling and next sentence prediction pre-training tasks. We then fine-tune our model toward two types of downstream tasks: (1) binary classification of websites as having either properly configured or misconfigured TLS, and (2) multi-label classification of websites with misconfigured TLS, based on misconfiguration reason. Visualizing our models' classification predictions for sites reveals distinct clusters representative of the different classes. We verified that these clusters indeed represent different reasons for proper configuration, or misconfiguration. This paper does not solve all problems relates to the use of transformers for protocol analysis or misconfiguration detection. However, it provides a compelling use case for this application of transformers, and in doing so also motivates future research using transformer models to tackle a variety of protocol analysis problems in communications networks.

CHAPTER 2

RELATED WORK

Transformer Models in Networking: The format of networking data differs greatly from any of the previous domains. Specifically, raw networking data is not naturally segmentable into words, sentences, or paragraphs as done the natural language domain. It also cannot be treated purely as raw data as done in the audio and computer vision domains, as networking data is inconsistent in form (e.g., one packet header field may represent message extension struct while another may be simply of type integer). As such, a number of works focus on how to best approach adapting/parsing networking data for use with transformer models (Houidi et al., 2022; Dietmüller et al., 2022). Our work differs from these previous efforts in that we train our model on sequences of larger protocol communication (TLS handshakes), compared to sequences of only some singular field value(s) of a protocol. These papers also raise the question of identifying an effective representation of networking data. Our work offers the results of using a parsed, text-based representation of networking data as a possible approach.

Other works similarly adapt BERT to networking problems, largely focusing on classifying traffic or device types. In another work, Le et al. train a custom BERT model on sequences of fully-qualified domain names from IoT devices, and evaluate their model's performance on predicting IoT device type (e.g., camera, doorbell) and manufacturer (Le et al., 2022b). As noted above, our work differs from this effort as it uses more a more comprehensive representation of network data for its training process. SHAPE, created by Dai et al. (Dai et al., 2022), PERT, created by He et al. (He et al., 2020), and ET-BERT created by Lin et al. (Lin et al., 2022a) (and its variations (Shi et al., 2023a,b)) are models trained using tokens and inputs parsed from end-to-end flows that resemble natural language words/sentences, and achieve high performance across a variety of encrypted traffic classification tasks. These models are related to our work as they adapt BERT, and generate tokens/input from larger communication in flows, into a format similar to natural language. However, because these models must tokenize encrypted traffic, they convert

pairs of bytes in flows into bigram strings (PERT and ET-BERT), or combine data from the packet payloads of sampled flows with sampled header field values (SHAPE). Our work focuses on a different objective of classifying reasons for TLS server misconfigurations, and as such also uses a different approach for generating model tokens and inputs. Finally, these works are primarily concerned with demonstrating their model achieves new state-of-the-art performance on encrypted classification tasks. In our work, we also hope to achieve high performance, but additionally want to examine if visualizing the embeddings our models use for prediction can be used to provide a different perspective towards understanding misconfigured TLS implementations.

TLS Misconfiguration: Proper configuration of the TLS protocol has long been studied by the networking community. Kotzias et al. present a longitudinal study of trends in TLS deployments from $2012 - 2018$, and find that while new advances in TLS are quickly deployed in clients, legacy support for deprecated components may lead to vulnerability (Kotzias et al., 2018). Works presenting changes to the Chromium browser interface and notification system (Zeng et al., 2019), and tools such as TLSAssistant (Manfredi et al., 2019), and PoliCert (Szalachowski et al., 2014) aim to fix various instances of misconfiguration both at scale, and on individual levels. Finally other works try explore how users configure their TLS servers. Simoiu et al. examined the security and proper configuration of the HTTPS protocol and present three major influences that affect how machines are configured (Simoiu et al., 2021). Krombholz et al. conducted a user study with security auditors and found that even these users find proper configuration of security related protocols difficult (Krombholz et al., 2017). Our work differs from the first four efforts in that we aim to not only detect misconfiguration, but also give perspective on how different instances of misconfiguration compare to each other. The last two efforts provide valuable insight for understanding how users configure their sites, but do not provide more fine-grained, explicit patterns that may cause misconfiguration. Our work aims to work towards this goal. We also try employing transformer models to investigate misconfiguration, compared to empirical analysis/user-studies.

CHAPTER 3

TRAINING MODELS ON TLS DATA

We base our model on BERT (Devlin et al., 2018), a popular transformer-based model that has been successfully extended to a number of domains, with modifications to the underlying vocabulary used during training. At a high level, BERT operates in two phases: pre-training and fine-tuning. Figure 3.1 presents an overview of the training process. In pre-training, BERT is trained over unlabeled input, and is evaluated on two downstream tasks to verify its understanding of the input. After pre-training, the trained model may be fine-tuned with labeled data to perform downstream tasks (e.g., classification, text generation) comprised of the same input format. We describe both pre-training and fine-tuning in this section.

3.1 Model Pre-Training

The process of creating the BERT model begins with pre-training. Here, the model uses two components: (1) a set of tokens resulting from splitting input specific to a domain called the *vocabulary*, and (2) a large training set of unlabeled, domain-specific data. Using these components, the model learns the semantics of the given domain in an unsupervised fashion, and evaluates its "understanding" via pre-training tasks. We detail how to construct these items, and our model's performance on pre-training tasks below.

Building a Vocabulary & Training Set: We begin by collecting sequences of server responses to various configurations of TLS `client_hello` messages. We collect only the server response components of a handshake as we care only about the values found in these messages that indicate proper or misconfiguration of TLS for a site. During pre-training, we do not mind the classification (i.e., proper/mis-configuration) of each input/sequence of handshake messages, as we care only about exposing the model to a sufficient variety of different server side TLS message values for it to develop an understanding of the semantics of these messages, and what these messages mean

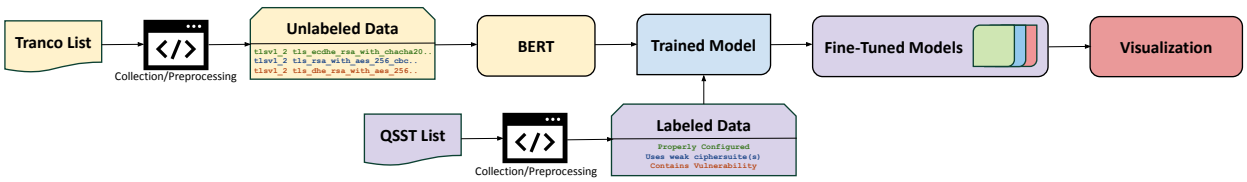


Figure 3.1: Overview of the pre-training (yellow) and fine-tuning (purple) stages in our model training pipeline.

in regard to a server's TLS protocol configuration. We collect handshakes from sites found on the Tranco list (Le Pochat et al., 2019), a top-list of websites generated by aggregating site rankings from different ranking sources, and validating these rankings to yield a manipulation resilient site ranking. We use the Tranco list as a starting point of sites to establish handshakes with, as these "top" sites are likely to be active to respond to handshake requests. We collect the server messages of handshakes themselves by writing a wrapper around the TLS Webserver Configuration Scanner created by Simouiu et al. (Simoiu et al., 2021) to establish handshakes to websites. For each unique ciphersuite configuration value, the scanner returns the server messages of the TLS handshake session, parsed to human-readable values using the Golang `tls` package (Project), in JSON format. We then parse the JSON output to remove unique/noisy values (e.g., cryptographic values, raw data), leaving only the configurable fields that a server operator may set in their TLS implementation. Specifically, each input used to train our model is a concatenation of the remaining values found in the headers of the `server_hello` and `server_hello_done`, as well as any optional server steps of a TLS handshake (e.g., `certificate`, `server_key_exchange`, `certificate_request`).

Performance on Pre-Training Tasks: At the end of the data-collection process, our training set contains $8{,}681{,}927$ inputs, comprising $394{,}839{,}183$ words, from $907{,}857$ distinct websites found in the Tranco top-list. Specifically, each input is a sequence of TLS server handshake messages, and the average input length is 48 words. We preprocess this input to cast all strings to lowercase, and remove non-UTF characters and punctuation (expected format for vanilla BERT). We then tokenize these inputs using WordPiece, the default tokenizer for BERT, and generate a vocabulary of $32{,}000$ tokens. Finally, we train our model for a total of four epochs on a system consisting of an AMD

EPYC 7302 16-Core Processor, 512 gigabytes of RAM, and a Nvidia A30 GPU. On completion of training, we evaluate our model on the standard masked-language modeling (MLM) and next sentence prediction (NSP) pre-training tasks. In MLM, the BERT model randomly masks a portion of the input, and attempts to predict the masked word/value. In next sentence prediction, BERT considers two inputs A and B, and is tasked with predicting if B follows A in the original training input or is instead randomly selected from the training corpus. For MLM, the model achieves accuracy of 98.8%, and cross-entropy loss of 0.075 nats. For NSP, the model achieves accuracy of 95.9% and cross-entropy loss of 0.067 nats. These results are encouraging, and signal our model understands the semantics of our parsed handshake messages.

3.2 Fine-Tuning to Classify Misconfiguration

Having achieved good performance in pre-training, we wanted to apply our model to more practical downstream tasks, including reasoning about misconfiguration. Specifically, we wanted to assess if our model could perform (1) simple binary classification of websites as having properly configured or misconfigured TLS, and (2) in sites with misconfigured TLS, more detailed multi-label classification of the reason for misconfiguration. We acknowledge that such tasks are solvable using existing tools such as those mentioned in Section 2. However, these tools do not provide ways to reason about how different properly configured/misconfigured websites compare to each other. In this section, we present the performance of our fine-tuned models, and investigate if their embeddings for inputs used for prediction can give a different perspective for comparing server configurations.

Data Collection: To build a ground-truth dataset for classification, we wrote a crawler that collects the results of user submitted queries to Qualys SSL Server Test (Qualys, b) (QSST). QSST is a service that runs comprehensive evaluations of websites, checking TLS implementation details. QSST simulates TLS handshakes with websites using 66 distinct client configurations (i.e., operating systems, web browser), testing on each support for weak ciphersuites and other components affected by protocol attacks (e.g., BEAST, DROWN). QSST then assigns a site a letter grade ranging from

"A+" (best) to "F" (worst), using a score computed from the test results. Due to the thoroughness of its process, we consider the results of QSST to be reliable for creating a ground-truth dataset of properly configured/misconfigured sites.

Data Labeling: Our crawler periodically examines the public facing results page of QSST, recording all websites with grade "A+" or "A" to a set of "properly configured" sites, and recording all websites with grade "C" and below to a set of "misconfigured" sites. We assign labels for the binary classification task by simply labeling all sites in the "properly configured" set as "Properly Configured", and labeling all sites in the "misconfigured" sites set as "Misconfigured." To assign labels for the multi-classification task, we input the sites from the "misconfigured" set to the testssl.sh tool (drwetter), an open source script that outputs verbose reasons for why a site is assigned its grade. testssl.sh provides very similar output to QSST (testssl.sh uses the Qualys grading scheme to evaluate sites), but is locally deployable, allowing us to parallelize label collection. We parse this output to create two labeling schemes: Multi-Label 1 (ML1) and Multi-Label 2 (ML2). ML1 assigns the most serious reason for misconfiguration as the label for a site (e.g., "Contains Vulnerability"). However, a site may have many reasons for misconfiguration. ML2 accounts for this, assigning the concatenation of the reasons for misconfiguration as the label for a site (e.g., "Contains Vulnerability, Certificate Issues"). Thus, we fine-tune towards three classification tasks: binary, ML1, and ML2.

Dataset Creation: Running our crawler and testssl.sh over the span of a month, we collect approximately $1{,}300$ unique misconfigured sites, and $42{,}000$ unique properly configured sites. This imbalance is due to the output format of the public facing results page of QSST. From observation, sites receiving grades of "C" and below are far less frequently submitted than "A+" or "A" rated sites (a positive surprise). We then again use our modified TLS Webserver Configuration Scanner to collect handshakes from sites, and parse the resulting JSON output to the input format discussed in Section 3.1. This yields $18{,}629$ inputs corresponding to TLS handshakes conducted with misconfigured sites, and $332{,}651$ inputs corresponding to handshakes conducted with properly configured sites. To create a balanced dataset of inputs from either group, we use all inputs collected

from the misconfigured sites, and select a matching $18{,}629$ inputs at random from the larger pool of inputs of properly configured sites. This yields our final fine-tuning task dataset of $37{,}258$ inputs.

Hyperparameter Tuning We split the dataset into training, evaluation, and testing sets for each of the three tasks using a $80{:}10{:}10$ split. This results in a training set of $29{,}806$ inputs, evaluation set of $3{,}726$ inputs and test set of $3{,}726$ inputs, for each task. Using the evaluation set, we perform grid search of model hyperparameters to find the best model for the data, and for each classification task. We fix the max-sequence length to be 128 as determined by our max observed input length, and fix the number of epochs to 4, as recommended in a subsequent work by the original BERT authors (Turc et al., 2019). We then vary *batch size*, the number of inputs processed at a time before updating the model, using five values ($8, 16, 32, 64$, 128) and *learning rate*, the step size by which model parameters are updated during fine-tuning, using four values, (5e-5, 3e-5, 1e-4, 3e-4) similarly as recommended in the mentioned work. For all three tasks, we observe both accuracy and cross-entropy loss to generally improve as batch size increases, and as learning rate decreases, though all combinations of batch sizes for learning rates 5e-5, 3e-5, and 1e-4 yield accuracy of $95{+}\%$. At the end of grid search, we find an optimal model trained with batch size of 128 and learning rate of 3e-5 for the binary classification task. For tasks ML1 and ML2, we find optimal models trained with batch sizes of 64 and 32 respectively, and learning rate of 5e-5.

CHAPTER 4

CAN TRANSFORMER MODELS EXPLAIN TLS MISCONFIGURATION?

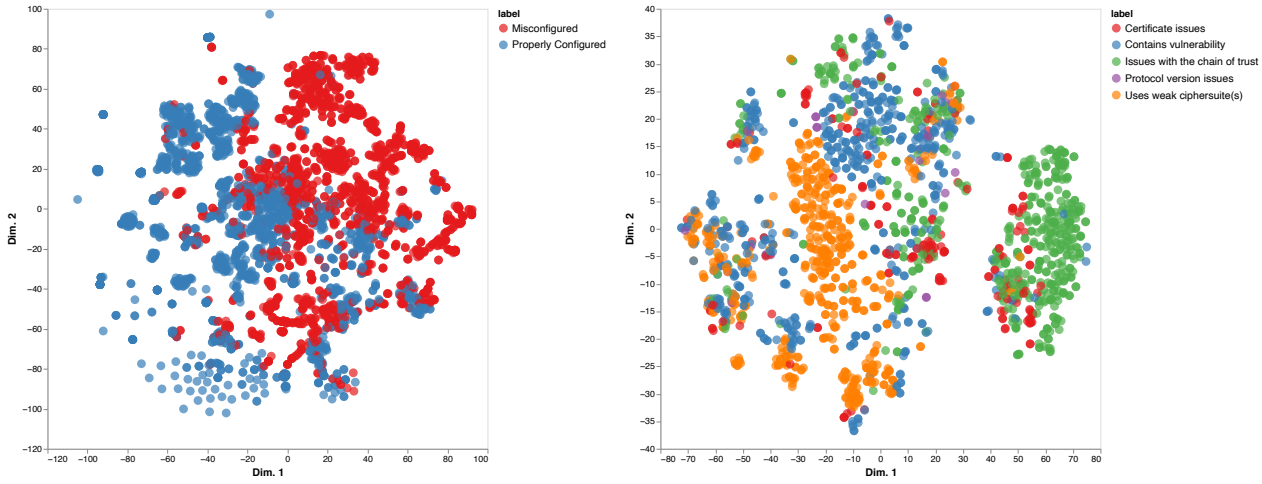
Having found our fine-tuned models accurately detect both sites with TLS misconfiguration, and the more fine-grained reasons for misconfiguration, we wanted to explore how and if our models' predictions for sites' TLS handshake messages clustered together. We discuss our process for visualizing the embedding space of our models, as well as more detailed performance metrics below.

4.1 Visualizing Predictions

Taking the best fine-tuned models from our hyperparameter search, we visualize their final-layer classifications of our evaluation set inputs (i.e., the `[CLS]` token) using t-distributed Stochastic Neighbor Embedding (t-SNE), casting the 768-dimension embeddings of our model to two dimensions. Figure 4.1 shows the results of this process[1]. Each point in either t-SNE represents our parsed representation of a set of TLS handshake server for a site in the test set. For the binary classification task (Figure 4.1a), red indicates a site is predicted misconfigured while blue indicates a site is predicted properly configured. For the ML1 task (Figure 4.1b), various colors represent different reasons for misconfiguration.

For the binary classification task t-SNE, we observe most predicted misconfigured sites on the right half of the plot, and most predicted properly configured sites on the left half of the plot. In either separation we also observe high-density clusters (e.g., properly configured northwest of center, misconfigured northeast of center). For the ML1 task t-SNE, there exists a notable separation between points representing sites predicted as misconfigured due to "Issues with the chain of trust", and all other reasons. Similar to the binary classification plot, distinct clusters of inputs for different predicted misconfiguration reasons are visible (e.g., "Uses weak ciphersuite(s)" slightly west of center, "Contains vulnerability" north of center). Our hypothesis for these clusters was that they

1. The t-SNE for ML2 shows high visual similarity to the t-SNE shown for ML1. We show only one figure for brevity.



(a) Binary classification task.

(b) Multi-Label 1 classification task.

Figure 4.1: t-SNE plots of the embeddings of our fine-tuned models.

represented different implementations TLS that resulted in the same classification (e.g., properly configured or misconfigured due to "Use of weak ciphersuite(s)," etc.).

To investigate this hypothesis, we manually examine the inputs corresponding to points of the same cluster, and find that the server configurations for these sites indeed share related, but slightly different configuration parameters. For instance, in the ML1 task t-SNE, server messages for the points of the "Contains vulnerability" (blue) cluster at values spanning roughly Dim $1 = [-20, 0]$ and Dim $2 = [10, 25]$, differ in their support of varying sets of weak ciphersuites, but all share support for (1) ciphersuites that use Cipher Block Chaining (CBC) as part of their bulk encryption, and (2) TLS/SSL versions TLS v1.0 and older. Support for these items allows for the POODLE and BEAST attacks, which leverage fallback to older TLS/SSL versions to then reveal encrypted content. To contrast, for the binary classification task t-SNE, server messages for the points of the properly configured (blue) cluster at values spanning roughly Dim $1 = [-60, -30]$ and Dim $2 = [18, 52]$ support (1) predominantly non-weak ciphersuites (i.e., stronger than 112 bits), and (2) only TLS v1.1 or higher (in some cases, only v1.2 and higher). Further, we observe in instances where a site supports a weaker ciphersuite (i.e., one exploited by a vulnerability), our models still correctly classify it as properly configured, as the other necessary component for exploitation (older version

of TLS/SSL) is not present. This suggests our models are not simply learning that any one field signifies a classification, but instead incorporate interactions between different configurations of sites' TLS message fields to make this decision.

Task	**Fine-Tuning Parameter**				**Classification Metric**						
	MSL*	E⋆	BS†	TR‡	Acc. (%)	F1 (%)	Prec. (%)	Rec. (%)	AUC (%)	Loss (nats)	MCC $[-1,+1]$
Binary	128	4	128	$3e$-5	97.6	97.6	97.6	97.6	97.6	0.099	0.952
Multi-Label 1	128	4	64	$5e$-5	96.7	96.6	96.7	96.7	97.3	0.109	0.953
Multi-Label 2	128	4	32	$5e$-5	96.5	96.4	96.4	96.5	98.1	0.119	0.951

* := Max Sequence Length, ⋆ := Epochs, † := Batch Size, ‡ := Training Rate

Table 4.1: Optimal hyperparameters for fine-tuning tasks and corresponding classification metrics.

4.2 Fine-Tuned Model Performance

Finally, we evaluate the performance of the same fine-tuned models used for visualization, on their held-out test sets. Table 4.1 provides an overview of the results of this evaluation. We observe overall good performance, with precision, recall, accuracy, and F1 scores of at least 96%, AUC of at least 97%, and cross-entropy loss of at most 0.119 nats, for all three tasks. When constructing our datasets for the multi-label classification tasks, we observed that some reasons for misconfiguration were more prevalent than others, indicating class imbalance. To account for this, we also measure the Matthews Correlation Coefficient (MCC), a performance metric that yields a high score only when a model's predictions match the true labels at a high rate, independent of the ratios of each class in its dataset (Chicco and Jurman, 2020). Ranging between $[-1,+1]$, a MCC of ± 1 indicates perfect prediction or total disagreement, while a MCC of 0 indicates prediction is no better than random. Our models also perform well in this metric, having a MCC of least 0.95 for all tasks.

CHAPTER 5

DISCUSSION AND FUTURE WORK

The performance of our fine-tuned models and corresponding visualizations of their predictions are promising, suggesting transformer-based models can not only detect TLS misconfiguration, but can also provide a different perspective for comparing TLS implementations. We posit that during the fine-tuning process, our models learn the fine-grained interactions and characteristics found between values in our inputs (parsed sequences of TLS server messages) that *together* indicate a class of misconfiguration. This follows similar results applying transformer-models to tasks such as sentiment analysis, in which presence of profanity alone does not definitively indicate positive or negative sentiment. Indeed, intuitively for our case, this makes sense, as the bidirectional transformers of BERT which our models are based on allow them to capture different contexts in which a handshake field-value may appear. Based on cursory results described in the previous section, this appears to be true, though additional tests should be conducted to completely verify this claim. We also verified that our parsed, string-based, space-delimited representation of TLS traffic that follows the expected format for BERT, is feasible for achieving good performance on networking-specific tasks. However, other representations of network data should not be ruled out as effective for transformer-based models.

In future work, we plan to further explore the above items, and additionally want to examine if we can use our models to better reason about trends in the state of TLS misconfiguration at large. We also are interested in applying transformer-based models to data of other networking protocols, especially encrypted DNS.

CHAPTER 6

CONCLUSION

The TLS protocol is confusing to implement and configure. Although it is easy to detect errors causing misconfigurations, it is more difficult to understand the nature (and cause) of these misconfigurations. In this paper, we investigate if transformer models can help approach this problem by creating a new BERT-based model trained on parsed TLS handshake server messages.

We verify that our input representation of TLS traffic (1) offers flexibility in adapting existing language model architectures to the networking, and (2) retains enough meaning to produce good performance. Our base model achieves $95+\%$ on the standard pre-training tasks, and our fine-tuned models achieve F1 scores of $96+\%$ on tasks classifying TLS configurations. Visualizations of the models' predictions using t-SNE reveal clusters that capture different variations of settings causing the same reasons for misconfiguration, a promising result toward more proactive prevention of TLS misconfiguration—and perhaps the use of transformers for protocol analysis more generally.

REFERENCES

Christopher Allen and Tim Dierks. The TLS Protocol Version 1.0. RFC 2246, January 1999. URL https://www.rfc-editor.org/info/rfc2246.

Abdullah Alsaedi, Nour Moustafa, Zahir Tari, Abdun Mahmood, and Adnan Anwar. Ton_iot telemetry dataset: a new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020.

Amazon. Configure SSL/TLS on Amazon Linux 2. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2.html.

Javed Ashraf, Marwa Keshk, Nour Moustafa, Mohamed Abdel-Basset, Hasnat Khurshid, Asim D Bakhshi, and Reham R Mostafa. Iotbot-ids: A novel statistical learning-enabled botnet detection framework for protecting networks of smart cities. *Sustainable Cities and Society*, 61:103041, 2021.

Tim M Booij, Irina Chiscop, Erik Meeuwissen, Nour Moustafa, and Frank TH den Hartog. Ton iot-the role of heterogeneity and the need for standardization of features and attack types in iot network intrusion datasets. *IEEE Internet of Things Journal*, 2021.

Francesco Bronzino, Nick Feamster, Shinan Liu, James Saxon, and Paul Schmitt. Mapping the digital divide: before, during, and after covid-19. In *TPRC48: The 48th Research Conference on Communication, Information and Internet Policy*, 2021.

Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21:1–13, 2020.

Jianbang Dai, Xiaolong Xu, Honghao Gao, Xinheng Wang, and Fu Xiao. Shape: A simultaneous header and payload encoding model for encrypted traffic classification. *IEEE Transactions on Network and Service Management*, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexander Dietmüller, Siddhant Ray, Romain Jacob, and Laurent Vanbever. A new hope for network model generalization. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 152–159, 2022.

Graylog Documentation. *Splunk Docs*. https://go2docs.graylog.org/5-1/home.htm.

drwetter. *drwetter/testssl.sh: Testing TLS/SSL encryption anywhere on any port*. https://github.com/drwetter/testssl.sh.

Hong Ye He, Zhi Guo Yang, and Xiang Ning Chen. Pert: Payload encoding representation from transformer for encrypted traffic classification. In *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*, pages 1–8. IEEE, 2020.

Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3366–3383, 2021.

Zied Ben Houidi, Raphael Azorin, Massimo Gallo, Alessandro Finamore, and Dario Rossi. Towards a systematic multi-modal representation learning for network data. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 181–187, 2022.

IANA. *Transport Layer Security (TLS) Parameters*. `https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml`.

IBM. Deprecated CipherSpecs. `https://www.ibm.com/docs/en/ibm-mq/8.0?topic=cipherspecs-deprecated`.

Xi Jiang, Saloua Naama Shinan Liu, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Towards designing robust and efficient classifiers for encrypted traffic in the modern internet.

Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the Internet Measurement Conference 2018*, pages 415–428, 2018.

Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. " i have no idea what i'm doing"-on the usability of deploying https. 2017.

Franck Le, Mudhakar Srivatsa, Raghu Ganti, and Vyas Sekar. Rethinking data-driven networking with foundation models: challenges and opportunities. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 188–197, 2022a.

Franck Le, Davis Wertheimer, Seraphin Calo, and Erich Nahum. Norbert: Network representations through bert for network analysis and management. *arXiv preprint arXiv:2206.10472*, 2022b.

Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019. doi:10.14722/ndss.2019.23386.

Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 633–642. ACM, 2022a. doi:10.1145/3485447.3512217. URL `https://doi.org/10.1145/3485447.3512217`.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Sal Candido, et al. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *BioRxiv*, 2022b.

Salvatore Manfredi, Silvio Ranise, and Giada Sciarretta. Lost in tls? no more! assisted deployment of secure tls configurations. In *Data and Applications Security and Privacy XXXIII: 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15–17, 2019, Proceedings 33*, pages 201–220. Springer, 2019.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

Nour Moustafa. New generations of internet of things datasets for cybersecurity applications based machine learning: Ton_iot datasets. In *Proceedings of the eResearch Australasia Conference*, pages 1–10. ACDS, 2019a.

Nour Moustafa. A systemic iot-fog-cloud architecture for big-data analytics and cyber security systems: a review of fog computing. *arXiv preprint arXiv:1906.01055*, 2019b.

Nour Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, 61:102994, 2021.

Nour Moustafa, M Ahmed, and S Ahmed. Data analytics-enabled intrusion detection: Evaluations of ton_iot linux datasets. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 727–735. IEEE, 2020a. doi:10.1109/TrustCom50675.2020.00100.

Nour Moustafa, M Keshk, E Debie, and H Janicke. Federated ton_iot windows datasets for evaluating ai-based security applications. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 848–855. IEEE, 2020b. doi:10.1109/TrustCom50675.2020.00114.

Mozilla. *Mozilla Observatory*. `https://observatory.mozilla.org/`, a.

Mozilla. Security/Server Side TLS. `https://wiki.mozilla.org/Security/Server_Side_TLS`, b.

Erik Nijkamp, Jeffrey Ruffolo, Eli N Weinstein, Nikhil Naik, and Ali Madani. Progen2: exploring the boundaries of protein language models. *arXiv preprint arXiv:2206.13517*, 2022.

nsacyber. Mitigating Obsolete TLS. `https://github.com/nsacyber/Mitigating-Obsolete-TLS`.

Nvidia. *ConnectX-5*. `https://www.nvidia.com/en-us/networking/ethernet/connectx-5/`.

CVE Program. *CVE-2014-3566*. `https://www.cve.org/CVERecord?id=CVE-2014-3566`.

The Go Project. *tls package - crypto/tls - Go Packages*. `https://pkg.go.dev/crypto/tls`.

Qualys. *Qualys SSL Labs - SSL Pulse*. https://www.ssllabs.com/ssl-pulse/, a.

Qualys. *SSL Server Test (Powered by Qualys SSL Labs)*. https://www.ssllabs.com/ssltest/index.html, b.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*, 2022.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular SDN Programming with Pyretic. *USENIX; login*, 38(5):128–134, 2013.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

Scapy. *Scapy*. https://scapy.net/.

Zhaolei Shi, Nurbol Luktarhan, Yangyang Song, and Gaoqi Tian. Bfcn: A novel classification method of encrypted traffic based on bert and cnn. *Electronics*, 12(3):516, 2023a.

Zhaolei Shi, Nurbol Luktarhan, Yangyang Song, and Huixin Yin. Tsfn: A novel malicious traffic classification method using bert and lstm. *Entropy*, 25(5):821, 2023b.

Camelia Simoiu, Wilson Nguyen, and Zakir Durumeric. An empirical analysis of https configuration security. *arXiv preprint arXiv:2111.00703*, 2021.

Splunk. *Splunk Docs*. https://docs.splunk.com/Documentation.

Pawel Szalachowski, Stephanos Matsumoto, and Adrian Perrig. Policert: Secure and flexible tls certificate management. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 406–417, 2014.

tlsfuzzer. TLSFuzzer. https://github.com/tlsfuzzer/tlsfuzzer.

Martino Trevisan, Danilo Giordano, Idilio Drago, Maurizio Matteo Munafò, and Marco Mellia. Five years at the edge: Watching internet from the isp network. *IEEE/ACM Transactions on Networking*, 28(2):561–574, 2020. doi:10.1109/TNET.2020.2967588.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ethernet Working Group(C/LM/802.3 WG). *Amendment to IEEE Standard 802.3-2018*. `https://ieee802.org/3/df/proj_doc/P802.3df_031222.pdf`.

Kun Yang, Samory Kpotufe, and Nick Feamster. A comparative study of network traffic representations for novelty detection. *arXiv preprint arXiv:2006.16993*, 2020.

Eric Zeng, Frank Li, Emily Stark, Adrienne Porter Felt, and Parisa Tabriz. Fixing https misconfigurations at scale: An experiment with security notifications. 2019.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*, 2023.

ZMap. *zmap/zgrab2: Fast Go Application Scanner*. `https://github.com/zmap/zgrab2`.