INSyT (**I**nnovative **N**etwork **S**securit**y** **T**echnologies): A Python Package for Network Intrusion Detection.

Bronze Frazer, Damon Tingey, Isaac Peterson, Taeyung Kim

April 2024

This report presents our approach to solving the notorious network intrusion detection problem. We detail a brief background of the problem and the data and methods we applied, as well as the results of our efforts. An in-depth description of our model deployment and packaging is also included.

1 Executive Summary

Network intrusion refers to the unauthorized access or activity on a computer network, often with malicious intent, such as stealing data or disrupting services. Network and computer system security is a matter of national and commercial importance, with the data of private companies and government organizations worth billions of dollars annually. Network intrusion detection, however, is a notoriously hard problem, with new attack vectors constantly being discovered. Traditional approaches include things such as rule and signature-based detection, anomaly detection, and stateful protocol analysis. These methods can be complex to manage or may not be effective against certain types of attacks.

The recent advancement of large language models presents an opportunity to apply new techniques to the problem of network intrusion detection. Large language models are well-suited to analyzing natural language, and by extension may be able to be applied in other unstructured language-like contexts. In this project, we attempt to use large language models as the basis for network intrusion detection. Using network and system log lines as inputs, we use large language models to "learn" the pseudo-language of computer log files. Our goal was to use the information provided from a raw log line to correctly classify it as benign, or to its appropriate attack classification.

The dataset used to train our model consists of over 3 million log lines, with their associated label indicating their network intrusion attack classification. Despite a large imbalance in the dataset classes, our fine-tuned DistilBERT model achieved an accuracy of at least 80% across all 23 classifications. DistilBERT is a transformer model, which outperform other model architectures in analyzing raw text data. The "Distil" in DistilBERT signifies that this is a distilled version of the original BERT model, making the model smaller and more efficient, with relatively little loss in performance. We used the smaller model because we wanted our model to be available to the vast majority of users, without the need for specialized hardware. From our tests, CPU inference will run in on average ~ 0.02 seconds.

In order to validate the results of our model, we ran five-fold cross validation on the dataset. We used stratified sampling to ensure that each class label showed up in both the training and test set, and then averaged the results of the cross-validation to give us our final confusion matrix. As mentioned previously, we ended up getting over %80 accuracy on all classes, and some classed achieving upwards of %100 on the test set. It should be noted that we were only able to validate the model with data from the same dataset, so while the model was never trained on the test data, there might be indicators within the dataset that caused the model to achieve higher accuracy. In order to measure true performance, we would need to test the model on new data, outside the current dataset.

To make the model accessible to everyone, we built a full-stack, user-friendly, data science tool to run and visualize the results from the model. The tool is hosted on PyPI, and and be installed with `pip install insyt`. This package includes both log line classification and

analysis as well as a front-end website that connects to a local database and displays important information to the end user.

This project provided valuable insight on the application of large language models across various fields. Future work would involve testing the model on more data, and potentially building out the infrastructure for an online model that is trained continuously. The objective of this project was to explore more practical approaches to network intrusion detection. As a result, we introduce this product as a solution for achieving that goal.

2 The Problem

Network intrusion is a significant concern where an external party gains unauthorized access to computer networks, systems, or private data. This can lead to a range of negative consequences such as data breaches, theft of sensitive information, disruption of services, financial losses, and damage to reputation.

Identifying network anomalies and recognizing which abnormalities lead to intrusion is an important security measure that can prevent or mitigate the damage done by unauthorized network access. However this can be a difficult task, as traditional techniques can be complex to manage, or not effective on specific types of attacks.

3 The Solution

To address these challenges, organizations employ a range of network monitoring and anomaly detection solutions, including intrusion detection systems (IDS), intrusion prevention systems (IPS), network behavior analysis (NBA) tools, machine learning algorithms, and threat intelligence feeds.

We attempt to address this problem by building a Large-Language Model(LLM)-based network analysis tool that can be used to analyze network and system behavior and identify network intrusion and anomalies. The tool and its components will be detailed in the following sections. In summary, it integrates a machine learning LLM to recognize patterns and irregularities from system log lines, while also including a full-stack application with frontend and backend that can be used for monitoring and analytics.

4 The Data

The dataset used for this project consisted of over 3 million log lines generated from a group of researchers out of the Austrian Institute of Technology that simulated 22 different attacks on 8 different servers over the period of 7 days [2] [3]. As mentioned on their website, the data was collected from the following logs: “Apache access and error logs, authentication logs, DNS logs, VPN logs, audit logs, Suricata logs, network traffic packet captures, horde logs, exim logs, syslog, and system monitoring logs.” So as such, our network security detection model is fine tuned on such logs.

As can be noted above, there was a significant class imbalance in our data. We considered several solutions such as Synthetic Minority Over-sampling Technique (SMOTE), which, simply put, would augment the minority samples, and undersampling. Ultimately we ended up limiting samples to 100,000 and getting good results.

To gain a better understanding of the separability of the classes, we used principal component analysis (PCA) to reduce the vector representations of the log lines to two dimensional space and plot them.

Interestingly enough, even when taking our high dimensional vector representations and lowering them down to two dimensions, there is definitive separability between the different classes. This gives some intuition on how the model is able to separate the high dimensional vectors into their appropriate classes.

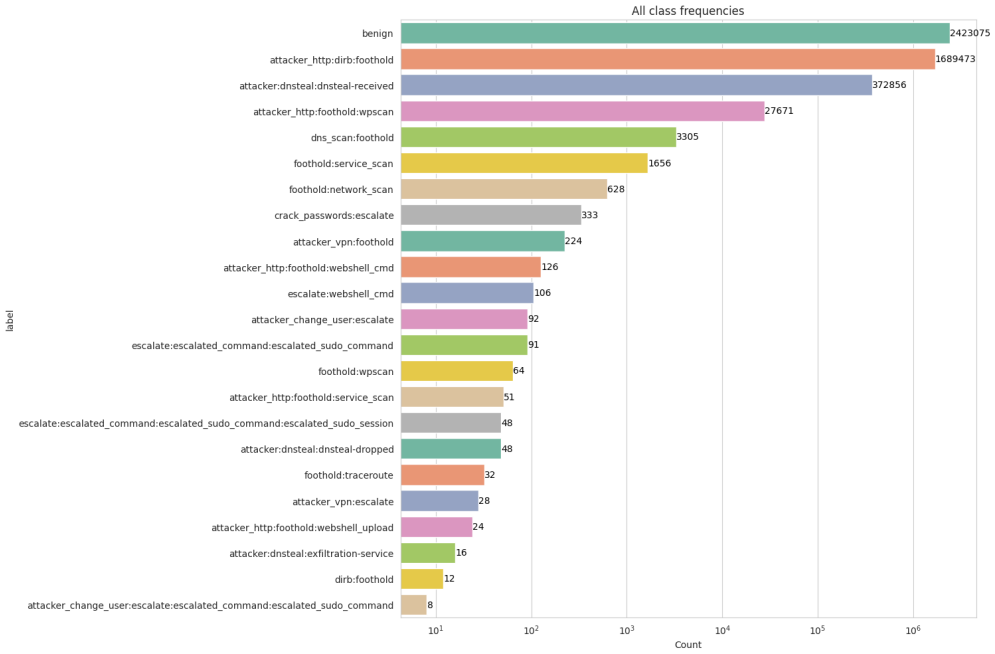

Figure 1: Distribtuions of network intrusion classes.

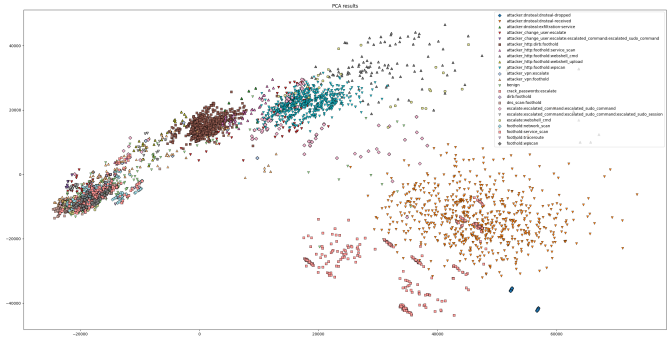

Figure 2: Plotted PCA results.

5 The Model

We used the DistilBERT model [4], BERT[1], because it can handle natural language as input, and DistilBERT, because it runs much faster with good inference speed even on a CPU. The class of BERT models are trained using the attention mechanism, which has been shown to outperform other methods when it comes to analyzing natural language [6]. DistilBERT was trained using a teacher student architecture with the original BERT model as the teacher. As mentioned in their paper the DistilBERT model is "a general-purpose pre-trained version of BERT, 40% smaller, 60% faster, that retains 97% of the language understanding capabilities" .

A quick note on the logits that form the output of our model. These logits represent probabilites of a log line being in a certain class. In network intrusion detection, an underlying problem with current solutions is that of an overwhelming amount of false positives. This results in the network intrusion systems being ignored by administrators. One solution we propose is a logit filter, that provides the front-end user with the ability to filter non-benign classifications to those with increased probability of being malicious, using a confidence level of $> 99\%$ for example. These logits are stored in the database when classifying the log lines.

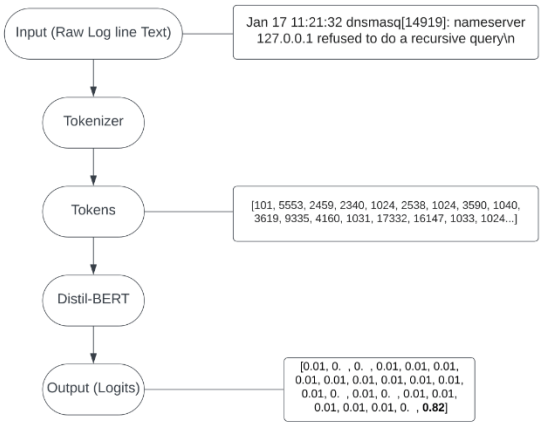

Figure 3: High level flowchart of DistilBERT model.

6 The Results

In order to ensure that our model was indeed performing well, we ran five fold cross validation and averaged their results on the test set. Each iteration the model was trained from scratch and we used stratified sampling to ensure that there were samples from each class in the training and test sets. An obvious limitation that we encountered when validating our model is not being able to test on data outside of the dataset we collected. Ideally we would be able to get more data that would be more representative, to see how well our model generalizes.

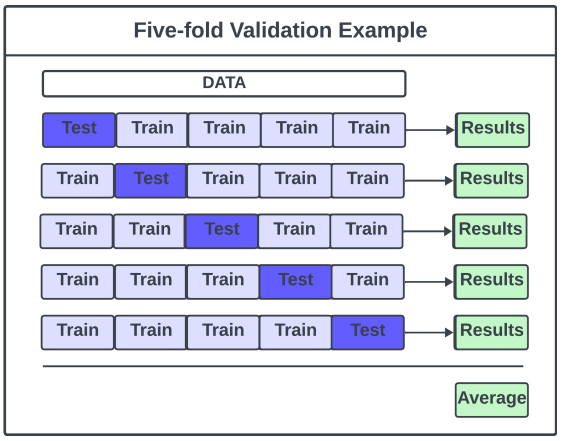

Figure 4: Outline of the five fold validation process.

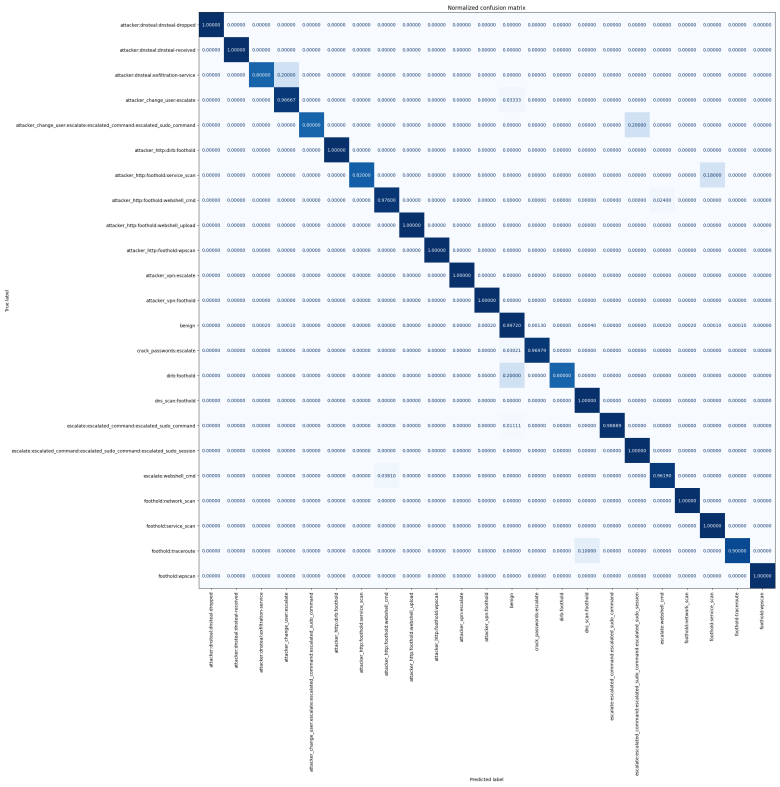

Figure 5: The average confusion matrix from the five fold validation results.

The confusion matrix shows that our model was able to correctly classify most of the log lines to their respective 23 classes. Future work would involve testing our model on more data to gain a better insight to its generalizability.

7 Packaging

For convenience, the INSyT model has been packaged into an easy to use Python package, hosted on Github and PyPI. This includes both a back-end inference server and database, as well as a graphical user interface. To ensure user data privacy and security, both aspects are run on the users local machine, with no data leaving the system. The specifics of both the back-end and front-end architectures are described below.

7.1 The Backend

The back-end system of our INSyT package is split into three main parts: (1) an inference server to run the INSyT classification and generative models, (2) A file watcher to monitor and process log files, and (3) a database and frontend server to interact with the database and serve the front-end user interface as static files.

7.1.1 Inference Server

While using the INSyT Network Instrusion System, the package will start a simple FastAPI inference server on a local port of the users machine. This inference server provides a simple API for the user or the other parts of the package to interact with the INSyT detection and analysis models. Currently there are two models available on the inference server. The first is the BERT-based INSyT classification model, describes in other sections of the report. This model takes in log lines, and classifies each one as either benign or one of any number of common attacks. The second model is a generative model, used to take any log lines that have been classified as a network intrusion attack and offer the user some simple analysis of the attack, as well as give recommendations for next steps and how the user might respond to such an attack.

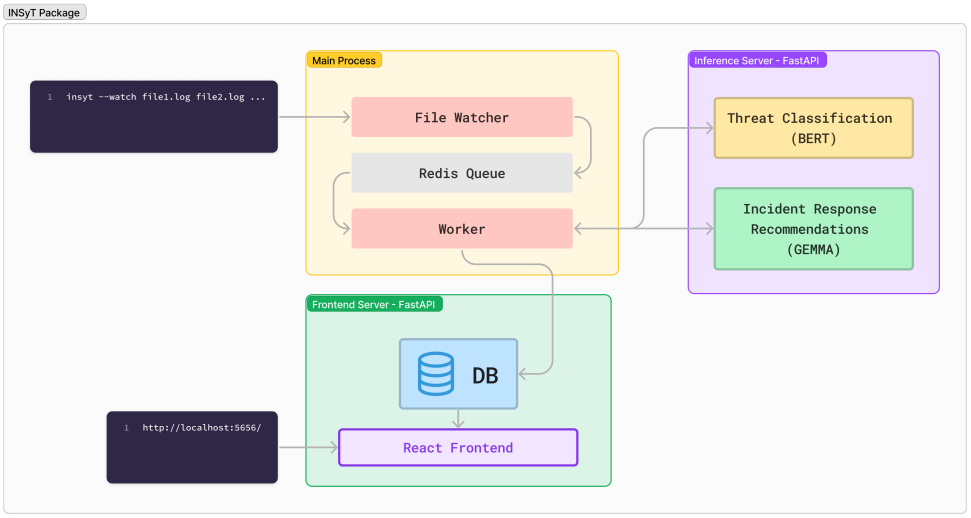

Figure 6: The architecture and tech stack for the INSyT package. The package is package is divided into a file watcher/worker, an inference server, and a front-end/database server.

The user has the choice between a model based on Google's Gemma [5] (slightly lower quality responses, less resource intensive) or Meta's LLama3 (higher quality analysis, more resource intensive).

7.1.2 File Watcher and Worker

The file watcher monitors log files and directories of log files for changes. A history of log lines is stored in the database, allowing it to pull all new lines from a file if the file watcher detects any changes. New lines are fed into a 'classification' Redis queue in batches. A worker thread will pull jobs from the Redis Queue, and query the inference server to detect whether any of the new lines in that batch represented a network intrusion attack on the system. It will then place any lines classified as an attack into an 'analysis' queue. Analysis jobs query the inference server for analysis and response recommendations, powered by open-source generative models. The worker will load results from any jobs into the database for storage and later retrieval.

7.1.3 Database/Front-end Server

When running the INSyT package, the package will automatically serve the front-end graphical user interface as static files to a local port on the users machine using FastAPI. This frontend can then be accessed from any webbrowser on the machine by navigating to the proper port. It also provides a simple API for the frontend to interact with the database, pulling the results of recent detections for use in the GUI.

7.2 The User Interface

The graphical user interface is a locally running dashboard, providing users a variety of tools and visualizations for easy analysis and insight into recent attacks. Users can see a timeline of recent attacks, an overview of attack types, and dive deeper into each line that the system in monitoring, viewing the context, classification, and analysis for each. All information for the front-end system is queried from the database, which is continually updated as long as the INSyT system is running. The front-end GUI is written in React, which is then built to static files and served using a FastAPI server to a port on the local machine.

8 Conclusion

Network Intrusion Detection is a notoriously hard problem, worked on by thousands of researchers and industry professionals for many years. Our project demonstrates that, within limited contexts, it is possible to detect attacks on a system using large language models, using only network and system log lines as inputs. Our system was able to achieve high accuracy in detecting attacks, using a dataset of simulated attack data on closed systems. We do not claim to have solved the problem of network intrusion detection, but our project is a representation of the potential for large language models to augment and assist cyber-security efforts now and into the future.

8.1 Future Work

Going forward, this project could be greatly expanded upon in a variety of ways. The extension to real attack data and data from different systems could give great insight into the generalizability of our current model, as well as introduce new directions to extend the model. Additionally, combining log line data with more traditional features and algorithms for network intrusion detection (such as tabular system data) would be an interesting next step to trying to improve accuracy and effectiveness.

8.2 Acknowledgements

We would like to thank our sponsors at Sandia National Laboratories, who provided funding for this project. Special thanks to Ryan Holt and Mike Reed, who provided invaluable subject-matter expertise and guidance through every step of this project, without which our project would not have been nearly as successful. Lastly, we would like to thank Dr. Quinn Snell and Dr. Scott Toborg, who provided mentorship, direction, and advisement throughout the course of this project.

You can find the repository containing the python package code and the trained model at the following urls.

Python Package Code:
https://github.com/Isaacwilliam4/INSyT

Trained Model:
https://huggingface.co/isaacwilliam4/insyt

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Max Landauer, Florian Skopik, Maximilian Frank, Wolfgang Hotwagner, Markus Wurzenberger, and Andreas Rauber. Maintainable log datasets for evaluation of intrusion detection systems. *IEEE Transactions on Dependable and Secure Computing*, 20(4):3466–3482, 2023.

[3] Max Landauer, Florian Skopik, Markus Wurzenberger, Wolfgang Hotwagner, and Andreas Rauber. Have it your way: Generating customized log datasets with a model-driven simulation testbed. *IEEE Transactions on Reliability*, 70(1):402–415, 2021.

[4] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[5] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid,

Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.