Student Internship Programme (SIP)

Final Project Report

at

IMDA

Reporting Period: 12 May 2025 to 1 August 2025

by

Anselm Long

Department of Computer Science
School of Computing
National University of Singapore
2024/2025

Project Title: *Data Science and Analytics for Cybersecurity Use Cases*

Project ID: *SY242256967*

Project Supervisor: *Michelle Yeoh*

Summary

This report documents an internship project at IMDA where a classifier was developed to distinguish malicious SSL/TLS certificates from benign ones. A large corpus of certificate data was collected, parsed into 22 fields and engineered into 32 features, including JA3 fingerprints, certificate lifetimes and issuer information. Various unsupervised and supervised models (Isolation Forest, One-Class SVM, Half-Space Trees, embeddings with k-NN, Random Forest and Graph Convolutional Networks) were evaluated. The best-performing model achieved an F1 score of 0.994 on a balanced, deduplicated test set, demonstrating strong potential for detecting malicious certificates.

Subject Descriptors:

- I.2.6 Learning
- I.5.1 Models (Trees, Neural nets, SVMs)
- I.6.4: Model Validation and Analysis
- K.6.5 Security and Protection

Keywords: anomaly detection, machine learning, cybersecurity, unsupervised learning, feature engineering

Implementation Software and Hardware: Python (VS Code IDE), scikit-learn, pandas, NumPy; developed and tested on Lenovo ThinkPad T14 running Windows 11.

Acknowledgement

I would like to express my sincere gratitude to my supervisors, Michelle Yeoh and Jonathan Samraj, for their invaluable guidance, encouragement, and feedback throughout the duration of this project.

Beyond my direct supervisors, I would also like to thank the entire Data Operations and Intelligence team at IMDA for making my internship both pleasant and intellectually enriching. I am grateful to the School of Computing, National University of Singapore, for providing the opportunity and resources to carry out this Student Internship Project.

Lastly, I would like to thank Jason Baek, my TA, and Ang May Syi, my fellow intern, for their support and insights, which contributed to the successful completion of this work.

Contents

Chapter 1

Introduction

In this project, I am tasked to build a classifier to distinguish malicious SSL/TLS Certificates from benign ones. Such digital certificates validate the identity of a website, organization, or server and provide a trusted platform for the user to connect and share information securely. However, malicious certificates used by malware or other phishing attacks have become increasingly common. By building anomaly detection models and replicating research studies, the best F1 score I derived is 0.994 on a small set of unseen test data.

1.1 Background

Secure Sockets Layer/Transport Layer Security (SSL/TLS) is a protocol to create secure communications across networks. When a client connects to a server, they perform a handshake to negotiate encryption methods and exchange certificates to verify identities. Such certificates validate the identity of a website, and they come in a format called X.509, which is what you see in Figure 1.1.

For this project, we only focus on a few of these fields, namely Issuer Name, Subject Name, Validity Period. I also have access to fields relating to the JA3 fingerprint.

Another concept to grasp here is the certificate chain. A certificate is

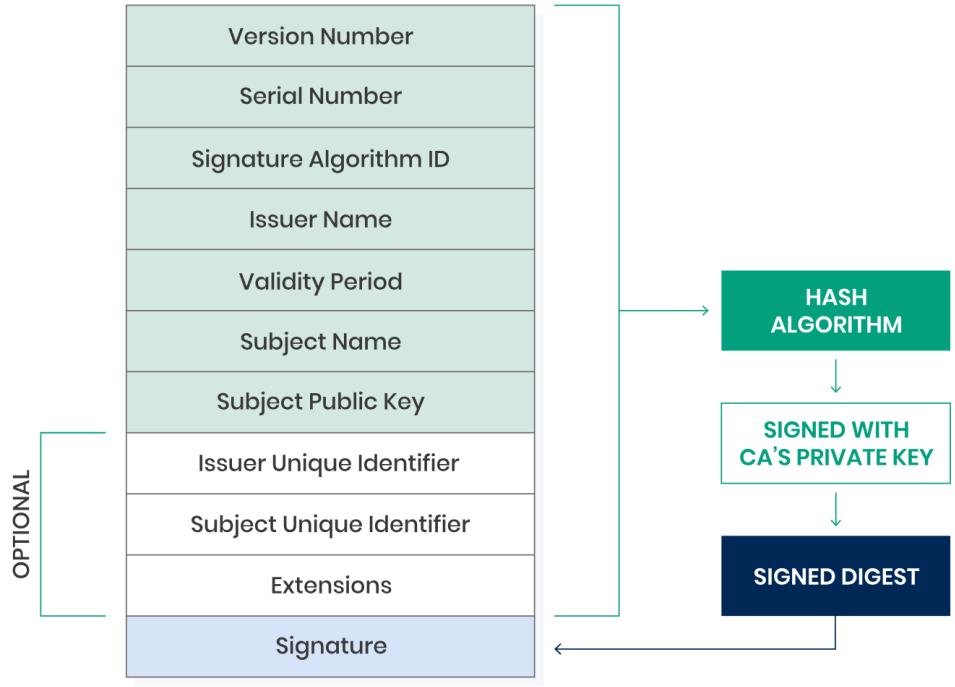

Figure 1.1: Typical fields of a X.509 Certificate
(What Is an x. 509 Certificate and How Does It Work?, 2024)

usually signed by a trusted root authority, known as a Certificate Authority (CA). Root certificates can then sign intermediate certificates, which in turn signs leaf certificates, conveying trust across the chain. However, some certificates are self-signed, meaning that they are not signed by a reputable CA, but rather sign themselves. Such certificates can be for testing purposes, and are generally not trustable. In this project, we define chain length as the number of certificates in this chain. This concept comes up again in the features that I engineer.

Chapter 2

Data

2.1 Initial Dataset

I was given a sample dataset (n $\approx$ 67k), followed by about 49 CSV files of about 70k records each. I combined this bigger dataset into about 3 million entries. For training, I took a sample from this bigger dataset.

Each CSV had 3 columns: certificates, JA3 fingerprints, and count. Certificates contained a JSON formatted string with each certificate in the chain of trust being one JSON object. The JA3 fingerprints column contained a string of dash separated numbers, detailing SSL/TLS Version, Cipher Suites, Extensions, Elliptic Curves, and Elliptic Curve Point Formats. Count is a residual of data processing.

I was also given a dataset (n = 2124) that had high probability of being malicious from a certain malware family. I combined this malicious data with an equal amount of assumed benign data from my initial dataset to create a labelled validation dataset for evaluation metrics. For my unsupervised learning models, I created a 50/50 split with my validation data. This dataset (Dataset A), was split into 50% validation for hyperparameter tuning, and 50% unseen test set for evaluation.

For my supervised learning models (Dataset B), I split this into 60/20/20, to 60% training, 20% validation, and 20% test.

2.2 Parsing Data

The parsed dataset contains a total of **22 fields**, as detailed in **Appendix A**. From the certificate data, both the issuer and subject information were extracted and split into individual columns for each attribute—namely: Country (C), Locality (L), Organization (O), Organizational Unit (OU), Common Name (CN), and State (ST). This results in **12 fields** (6 from the issuer and 6 from the subject). Additional certificate-related fields include the **issuance date**, **expiry date**, and **Subject Alternative Names (SANs)**, bringing the total to **15 fields** derived from certificate metadata.

Five additional fields were generated from **JA3 fingerprint data**, and the final two fields—**chain index** and **chain length**—were introduced during parsing of the certificate chain. For example, if a certificate chain consists of three certificates, each certificate is assigned a **chain index** (0, 1, 2), while the **chain length** remains fixed at 3 across all certificates in that chain.

Chapter 3

Feature Engineering

The total feature list consists of 31 features, where the full list is detailed in Appendix B. Every subsection describes the part of the original data that the features below it are engineered from. This section details a few of the more interesting features that were derived, the rest are presence features or one-hot encoded.

3.1 Issuer Name

3.1.1 Issued from Free Certificate Authorities (CA)

I have listed a few free certificate authorities:

- Let's Encrypt
- ZeroSSL
- SSL Corporation
- Cloudflare, Inc.
- Buypass
- CAcert Tech.

Taken from Ondara (2024) and online research, there is a correlation between certificates being issued from free CAs and malicious activity (Fasllija et al., 2019).

3.1.2 CA Z-Score Lookup (Dropped Feature)

Some Certificate Authorities (CAs) are statistically more likely to issue malicious certificates than others. Splunk (2023) conducted a large-scale analysis of over **5 billion SSL/TLS certificates**, producing a dataset that ranks CAs based on their association with malicious activity. For each CA, Splunk computed the *proportion of risky certificates* issued relative to their total issuance volume, then transformed these proportions into *z-scores* based on the standard normal distribution. Higher z-scores indicate greater likelihood of malicious association.

Splunk provided two separate datasets: one for **root CAs** and another for **issuer CAs**, each using a different parsing format. To match these formats, a custom parser was implemented to generate canonical strings by concatenating certificate fields with slashes. For example:

> /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=Microsoft ECC Root Certificate Authority 2017

These parsed strings were used to look up the corresponding z-scores in the Splunk datasets.

Each certificate was then assigned a z-score based on this lookup. If the issuer CA z-score was available, it was used directly. If not, the root CA z-score was used. If neither was found, a default value of 0 was assigned. The resulting z-score served as a feature to evaluate the trustworthiness of the issuing CA.

However, this feature was ultimately dropped after an ablation study conducted using HST (Half Space Trees) showed that its inclusion did not improve model performance significantly. Moreover, the additional processing overhead required to compute the z-scores was deemed unjustified given the minimal performance gain.

3.1.3 Suspicious Fields

This feature leverages a blacklist compiled by Abuse.ch's *Top Malicious SSL Common Names* (2021), which identifies known Common Names frequently

associated with malicious SSL certificates. Specifically, the feature checks both the **issuer Common Name** and **issuer Organization** against entries from this list.

Some malicious entries follow unconventional formatting patterns, such as: `C=US, ST=Denial, L=Springfield, O=Dis`. To ensure accurate detection, the check was extended to additional fields including **Country (C)**, **State (ST)**, **Locality (L)**, and **Organization (O)** for an exact field-wise match.

A boolean flag, `has_suspicious_fields`, is set to `True` if any match is found, indicating potential presence of suspicious certificate metadata.

3.2 Subject Name

3.2.1 Length of Domain

The domain is extracted by parsing the Subject Common Name (CN), specifically by stripping any left-side wildcard (e.g., `*.example.com` becomes `example.com`). The feature `length_of_domain` represents the total number of characters in the resulting domain string.

3.2.2 Entropy of Domain

To identify potentially algorithmically generated or obfuscated domain names, the Shannon entropy of each domain is calculated using the following formula:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

This entropy-based feature is adapted from Feature 7 of Fasllija et al. (2019), and serves to quantify the randomness of character distribution within the domain name. Higher entropy values may indicate suspicious or auto-generated domains.

3.2.3 Number of Hyphens (Dropped)

This feature counts the number of hyphens present in the domain name, based on the hypothesis that longer, hyphenated domains may be indicative of malicious behavior. This approach is adapted from Feature 8 of Fasllija et al. (2019). However, after further evaluation, this feature was dropped due to the observed distribution of hyphen counts in malicious certificates closely mirroring that of the overall dataset.

3.2.4 Inner Top-Level Domain (TLD) in Subdomain

Adapted from Feature 5 of Fasllija et al. (2019), this feature flags the presence of popular top-level domains (e.g., `org`, `com`, `net`) within inner subdomains. As noted in their study, "attackers may include popular top-level domains in the inner domain in order to mislead users that are familiar with them into trusting a fraudulent website." The detection of such patterns is therefore considered suspicious.

3.2.5 Suspicious TLD

Inspired by Feature 4 of Fasllija et al. (2019), this feature targets domains using newly introduced or low-cost TLDs, which have been widely exploited by attackers. A reference list of suspicious TLDs was constructed using Fasllija et al.'s dataset in conjunction with The Spamhaus Project (2025).

To operationalize this feature, a one-hot encoding scheme was applied to capture the most frequently observed suspicious TLDs: `.tech`, `.info`, `.xyz`, `.top`, and `.vip`. All remaining TLDs were grouped into an `others` category.

3.3 JA3 Fingerprint

3.3.1 JA3 Hash

This feature leverages the JA3 fingerprinting method, which encodes TLS client hello parameters into a standardized fingerprint. By applying an MD5

hash to the original JA3 string, I compare the resulting hash against a known list of malicious JA3 fingerprints provided by Abuse.ch (2021). If a match is found, the boolean feature `is_malicious_hash` is set to `True`, indicating a suspicious TLS client configuration.

However, this feature was ultimately dropped due to a lack of relevance in the available dataset. The malicious samples available did not contain JA3 hash information, rendering the feature inapplicable. Additionally, the potential for MD5 hash collisions and false positives further diminished its reliability.

3.4 Issuer and Expiry Dates

3.4.1 Certificate Lifetime

Using the certificate's issued date and expiry date fields, the certificate lifetime is computed by taking the difference between the two dates. After converting both fields to `pandas.Datetime` format, the result is stored as a numerical variable, `cert_lifetime_days`, representing the total validity period of the certificate in days.

3.5 Subject Alternate Name (SAN)

3.5.1 SAN Count

The feature `altnames_count` records the total number of Subject Alternate Names (SANs) present in a certificate. Anomalously high or low SAN counts may indicate malicious intent or misconfiguration.

3.5.2 Average SAN Shannon Entropy (Dropped)

The feature `san_entropy_avg` calculates the average Shannon entropy across all SANs in a given certificate. Algorithmically generated SAN entries are expected to exhibit higher entropy. However, after plotting a histogram of this feature, its distribution was found to closely resemble that of the domain

entropy feature, offering little additional signal. As a result, this feature was dropped.

3.5.3 Presence of Common Name (CN) in SAN

The boolean feature `is_CN_in_SAN` checks whether the Subject Common Name (CN) is also listed among the SANs. While this condition doesn't occur in only 0.66% of all certificates, further analysis showed a notable difference in its distribution between malicious and benign certificates. This suggests that it may be a useful feature and is thus retained.

3.6 Miscellaneous

3.6.1 Number of Missing Fields

The feature `number_of_missing_fields` counts how many expected fields are absent in a given certificate. According to NCC Group (2021), there is a statistical relationship between the number of missing fields and the likelihood of a certificate being malicious.

3.6.2 Chain Length

This feature captures the length of the certificate's chain. While only the leaf certificate is retained for modeling, the total chain length is preserved as a numerical feature, `chain_length`.

3.6.3 Presence Features

To retain feature variance across certificates, a set of 13 binary presence indicators was engineered. These include boolean flags for the presence of common fields in both the Subject and Issuer sections: Country (C), State (ST), Locality (L), Organization (O), Organizational Unit (OU), Common Name (CN), and Alternate Names (SAN). Each feature indicates whether the corresponding field is present in the certificate.

Chapter 4

Results

Model	Accuracy	Precision	Recall	F1 Score
Isolation Forest	0.9463	0.9181	0.9791	0.9476
OCSVM	**0.9651**	**0.9470**	**0.9848**	**0.9655**
HST	0.9228	0.8751	**0.9848**	0.9267

Table 4.1: Unsupervised Learning Performance Metrics on Balanced, De-duplicated Unseen Test Set.

Model	Accuracy	Precision	Recall	F1 Score
Embeddings + kNN	0.9240	0.9357	0.9102	0.9228
GCN	0.9424	**0.9944**	0.8889	0.9387
Random Forest	**0.9941**	0.9886	**1.0000**	**0.9943**

Table 4.2: Supervised Learning models metrics on Balanced, De-duplicated Unseen Test Set.

The tables above summarize the performance of both unsupervised and supervised models on a balanced, de-duplicated test set. Among the unsupervised models, OCSVM achieved the best overall results, with the highest F1 score and recall. For supervised models, Random Forest significantly outperformed the rest, attaining near-perfect precision, recall, and F1 score. These results highlight the effectiveness of supervised learning when labeled data is available, while also showing that OCSVM is a strong choice in unsupervised

settings. However, the approaches will be compared in the following section and a recommendation will be given at the end.

Chapter 5

Approaches

5.1 Clustering

To explore unsupervised detection of malicious certificates, I initially experimented with the KMeans algorithm. Using the elbow method, an optimal value of $k = 21$ was estimated. While Gómez et al. (2023) used clustering effectively with labeled malware family data, the absence of such labels in this study limited interpretability, and clustering was not pursued further.

5.2 Isolation Forest

Isolation Forest was selected as a primary unsupervised approach for detecting anomalous TLS certificates due to its scalability and interpretability. Originally proposed by Liu et al. (2008), this method constructs an ensemble of randomly generated binary *isolation trees*, where each split selects a feature and threshold at random. Anomalous certificates—being rare and distinct—tend to be isolated in fewer partitions, resulting in shorter paths and higher anomaly scores. The algorithm's linear time complexity and minimal memory requirements make it particularly well-suited for large-scale datasets such as those used in this project. Moreover, recent studies, such as Ostertág and Stanek (2024), have validated its effectiveness in detecting anomalies within X.509 certificate transparency logs. In the context of this work, Isolation

Forest achieved an F1 score of 0.9476 on a balanced, de-duplicated test set, demonstrating its utility as a robust baseline for certificate anomaly detection.

5.3 Local Outlier Factor (LOF)

The Local Outlier Factor (LOF) algorithm is designed to identify anomalous data points by comparing the local density of points to their neighbors. It operates by measuring how much a data point deviates from its surrounding points in terms of density. While LOF is effective in detecting outliers in certain contexts, its quadratic time complexity makes it computationally expensive, particularly for large datasets. Despite initial expectations, the LOF model performed poorly on this dataset, and further analysis indicated that it was not suited for this task. Consequently, LOF was excluded from the final evaluation.

5.4 One-Class Support Vector Machines (OCSVM)

One-Class SVM is a variant of the traditional SVM algorithm, specifically designed for outlier and novelty detection tasks. Unlike traditional SVMs, which are typically used for binary classification, One-Class SVM trains exclusively on data from a single class, referred to as the target class. The algorithm learns a boundary or decision function that captures the distribution of the target class in the feature space, thereby modeling the normal behavior of the data (Mounish V, 2024).

However, One-Class SVM has a quadratic time complexity, which can become computationally expensive with large datasets. Despite this, initial results indicate that it performed well on this project, and it emerged as the best-performing model on my dataset, achieving high accuracy and F1 score compared to other unsupervised methods.

5.5 Half Space Trees (HSTs)

NCC Group (2021) used HSTs on live certificate streams and consistently identified anomalies. This is an ensemble method, where multiple trees are built and results are aggregated. This approach is the most promising one, and hence I will delve deeper into explaining the algorithm.

5.5.1 Introduction

Tan et al., (2011) defines each HS-Tree consists as a set of nodes, where each node captures the number of data items (i.e. mass) within a particular subspace of the data stream. Mass is used to profile the degree of anomaly because it is simple and fast to compute in comparison to distance-based or density-based methods.

5.5.2 Methodology

The algorithm segments the data stream into 2 consecutive windows of fixed size (n = 250) – the reference window and the latest window. During the initial stage of the anomaly detection process, the algorithm learns the mass profile of data in the reference window. Then, the learned profile is used to infer the anomaly scores of new data subsequently arriving in the latest window. New data that fall in high-mass subspaces is construed as normal, whereas data in low-mass or empty subspaces is interpreted as anomalous.

As new data comes in, the algorithm learns its mass profile as well. When the latest window is full, the newly recorded profile is used to override the old profile in the reference window – so it always stores the latest profile to score the next batch. Now, the latest window erases its stored profile to capture that of the next batch. This repeats as long as the stream exists.

5.5.3 Definition

Tan et. al defines a Half Space Tree (HS-Tree) of depth h as a full binary tree consisting of $2^{h+1} - 1$ nodes, where all leaves are at the same depth h.

A tree is constructed by picking a randomly selected feature. Then, we split the feature space in half (the library that I use, River, randomly splits the feature space with padding instead), creating a left and right child for the node. This continues until maximum depth is reached.

5.5.4 Algorithm

Before creating a tree, the algorithm initialises a work space, randomly generates a synthetic range for each feature, creating a unique coordinate space for every Half-Space Tree. This allows the trees to be constructed without seeing any data, while maintaining an ensemble of diverse HS-Trees.

Once trees are constructed, we record the mass profile of the data by traversing every instance through each tree. At each node, we increment the mass count before recursively updating the mass count for subsequent levels of the tree.

5.5.5 Anomaly Score

We score data based on their mass distribution. Let $s_{n,t}$ be the score of a test instance x at a particular node n of a particular tree t, and a particular maximum depth d. Let m_i be the mass of a node at depth level i of the tree.

$$s_{n,t} = m_i * 2^i$$

We keep adding the node scores until we reach a terminal node. A terminal node here refers to a node that reached the maximum depth, or a node that contains *limit* instances or fewer, where *limit* is a parameter that defaults to $0.1w$ (window size).

The tree score, s_t is then the summation of the scores of each node along the path, increasing in depth.

$$s_t = \sum s_{n,t}$$

The anomaly score, S will then be the summation of each tree score in the

ensemble.

$$S = \sum_{t \in T} s_t$$

To fit machine learning convention, we take the inverted score, S' as $1 - S$ so that a value closer to 1 is malicious, and a value closer to 0 is benign.

$$S' = 1 - S$$

To interpret this, a node with smaller mass is likely to contain anomalous data points due to the lack of similar points. Anomalies will end up with lower scores then. The time complexity and space complexity for this algorithm is amortised constant.

5.6 Embeddings + k-Nearest Neighbors (kNN)

Based on Shashwat et al., (2024), I try their approach to classifying certificates based on putting the data through a Large Language Model (LLM), obtaining the embedding, and classifying a new certificate based on its nearest neighbors. The methodology used in the paper consisted of a few steps.

1. Clean Data - fill missing values with NA
2. Process certificates and combine into issuer and subject string.
3. Embed issuer and subject string with Character BERT (C-BERT) separately and combine embeddings.
4. Setting Up a FAISS Vector Database for Similarity Search
5. k-NN Classification with Majority Voting
6. Evaluation

While this method relies on supervised data, I use our small set of curated and labelled benign and malicious data to train and validate this model. From the original parsed benign and malicious certificates, I drop all columns but the 12 fields describing the certificate, such as the Country, Locality, State,

Common Name, Organization, and Organizational Unit. After removing duplicates, I decided to follow the parsing and embedding strategy used in the paper by filling in empty fields with "NA", and joined the fields together with commas as a delimiter. I embedded both issuer and subject separately, and concatenated them together. I took benign and malicious data in a 1:1 ratio. The resulting dataset (n = 2532) was split into train (n = 2025) and validation (n = 507) sets.

After embedding, I store the vectors in a Facebook AI Similarity Search (FAISS) vector database, and conduct kNN search with the validation set. While the paper suggested an optimal k of 5, I did some tuning and found that my results achieved a result of k = 3. This approach was useful in understanding clustering of certificates, as similar certificates were often grouped together.

5.7 Contrastive Learning

Han et al. (2024) proposed a contrastive learning method for unsupervised malicious network traffic detection. Contrastive learning, a form of self-supervised learning, aims to learn representations by comparing positive and negative pairs of data points. In the context of anomaly detection, the method works by embedding data points in a feature space where similar instances (positive pairs) are close together and dissimilar instances (negative pairs) are far apart. This technique has shown promising results in detecting anomalies in network traffic, where the focus is on learning useful representations without requiring labeled data.

While contrastive learning could offer valuable insights for my use case, especially for detecting anomalies in TLS certificates, the complexity of implementing the method and the need for additional computational resources made it infeasible for this project. As a result, contrastive learning was not pursued, but it remains an area of potential exploration for future work.

5.8 2-Layer Certificate Graph Convolutional Network (GCN)

Liu et al. (2022) proposed a graph-based approach to detect malicious certificates by modeling the problem as a graph. In this approach, certificate attributes and certificate documents are treated as nodes, while co-occurrence information between attributes forms the edges between nodes. I implemented this method using a labeled dataset with 4220 samples, consisting of a balanced 1:1 split of benign and malicious certificates. The dataset was divided into 80% for training, 10% for validation, and 10% for testing. The training and validation sets were combined to build the graph corpus, while the training set was used for model training. The validation set was masked during training, and the final 10% of the dataset was used to construct a separate graph for the test set.

This approach can be categorized as transductive learning (where the validation set is part of the graph during training) and inductive learning (where a separate graph is used for the test set). The distinction between these two methods highlights different strategies for utilizing graph-based models in machine learning.

5.8.1 Parsing

The certificate data is parsed into a list of lists, where each list represents an individual certificate. Each field within the certificate is treated as a separate item, and multiple items within the same field (e.g., Alternate Names) are flattened and separated by commas.

5.8.2 Graph Nodes and Edges

In this graph-based approach, nodes represent both certificates and their attributes. Edges between certificates and attributes are weighted using Term Frequency-Inverse Document Frequency (TF-IDF). Here, term frequency refers to how often an attribute appears in a certificate, while inverse

document frequency quantifies the rarity of an attribute across certificates. This weighting mechanism helps assess the importance of each attribute in the graph.

For edges between attributes (Attribute-Attribute), we calculate Pointwise Mutual Information (PMI), which measures the co-occurrence of two attributes compared to their individual frequencies.

5.8.3 Building the Graph

The graph is represented as an adjacency matrix, using a sparse matrix format from SciPy for memory efficiency. Each node is one-hot encoded as a vector, following the approach used in the original paper. The graph is constructed using the PyTorch Geometric framework, which allows for efficient graph data handling and model training.

5.8.4 Training the Model

The architecture replicates the design in Liu et al. (2022), employing a two-layer Graph Convolutional Network (GCN) consisting of: Graph Convolution $\rightarrow$ ReLU activation $\rightarrow$ Dropout $\rightarrow$ Graph Convolution $\rightarrow$ Linear output layer. They suggest that any more layers result in a drop in model performance.

5.8.5 Results

The transductive learning approach, where the validation set is part of the graph corpus but is masked during training, achieved an F1 score of 0.9387. In contrast, the inductive learning approach, where the test set is kept separate and a new graph is created for testing, resulted in an F1 score of only 0.5674.

5.8.6 Conclusion

Due to the poor performance on the inductive test data and the complexities involved in graph data parsing, this approach is not recommended for

production use.

Chapter 6

Validation and Explanation

6.1 Locally Interpretable Model-Agnostic Explanation (LIME)

With the predictions of my models, now it was time to validate and explain the predictions. However, most of my anomaly detection models were black boxes, and there were no inbuilt feature importance attributes I could find. I started by running LIME on my models to reveal the top features that contributed to the classification. However, as LIME is a local model, the results were not generalizable globally. LIME on different models also yield different results. As seen in Figure 6.1, the results were confusing to analyse and often picked features that didn't seem to align with domain knowledge. There was a need for better explainability in my models. Hence, I turned to supervised learning, specifically Random Forest.

6.2 Random Forest Feature Importance

I also evaluated feature importance using a Random Forest (RF) model trained on a small labelled dataset. While this approach provides a global view of feature relevance, the results (Figure 6.2) diverged significantly from those obtained through LIME. Yet, this was more aligned with domain knowledge. With the good results obtained from the RF model, this validated the

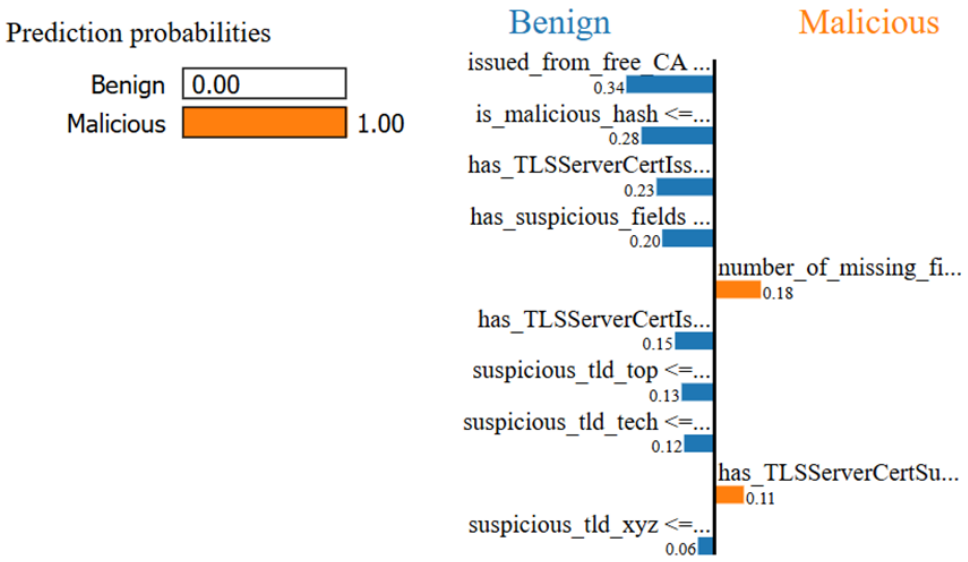

Figure 6.1: LIME on HST, True Positive Instance

features that were picked.

6.3 Half Space Trees Feature Importance

Diving deeper into HSTs, I managed to print out and parase the actual decision trees that were in the model. By referring to the source code, I created a visualizer that showed where anomalies were isolated. By looking at this, I could gain a clearer idea of which features resulted in anomalies. In Figure 6.3, I traced out the decision path for a particular benign sample, and this helped to explain the predictions in greater detail.

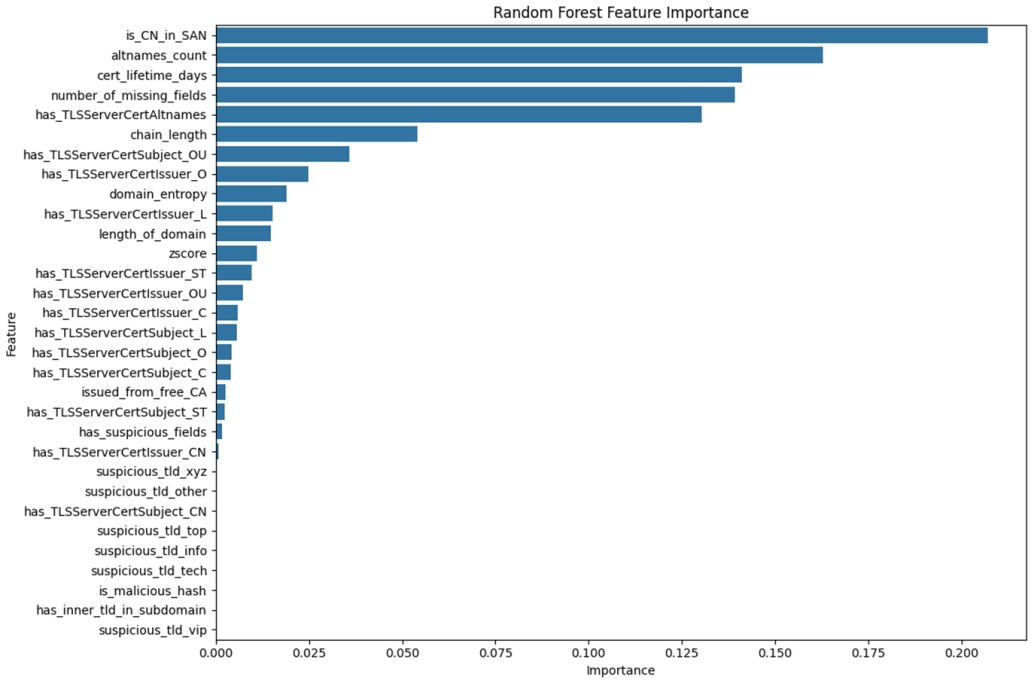

Figure 6.2: Feature Importance for Random Forest

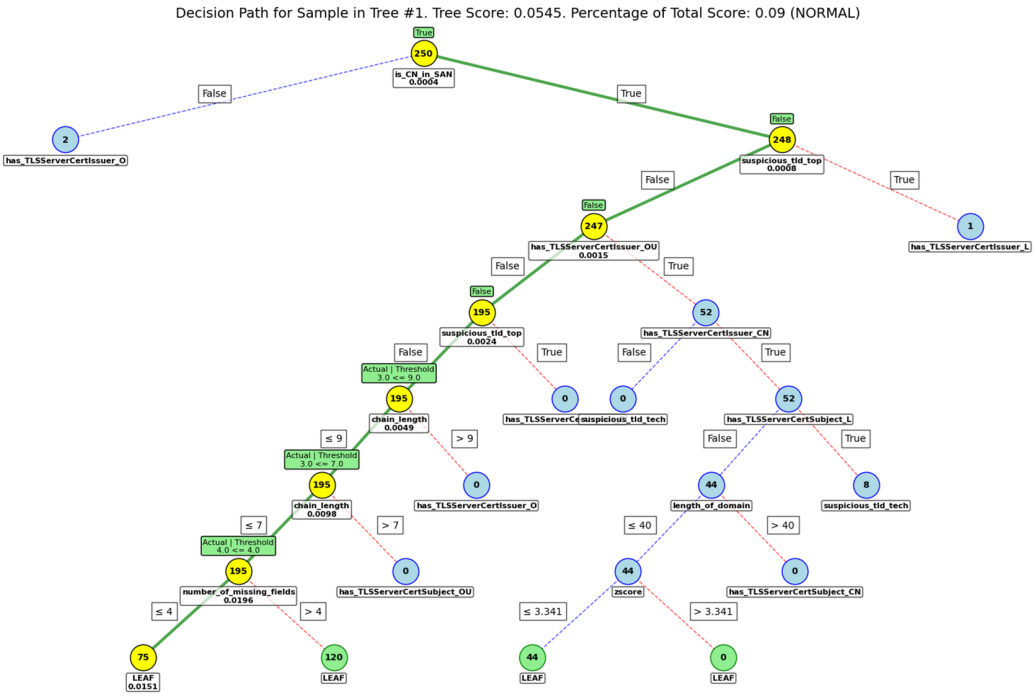

Figure 6.3: Half Space Trees Decision Path for a Benign Sample

Chapter 7

Reflections

Throughout this project, I faced a number of challenges that required both technical and problem-solving skills. One of the primary issues was the computational complexity, as the large dataset (approximately 9 million rows) required extensive processing time, often taking 2-3 hours per cycle, which I managed by running tasks overnight. Additionally, working with data presented a learning curve, as I had little prior experience with data analysis and had to quickly become proficient in using Pandas for data manipulation. My initial use of Python notebooks became disorganized, and I eventually refactored the code into modular classes, improving both the clarity and efficiency of my work. Another significant challenge was scaling the data correctly. Initially, I scaled the training and malicious datasets separately, leading to inflated F1 scores due to mismatched data distributions. After recognizing this, I ensured consistent scaling by applying the same transformation to both the training and validation sets. Feature selection also posed challenges, as some features were dropped due to skew or redundancy, which improved the model's performance. Lastly, the lack of labeled data for training and validation was a constraint, but by combining a small labeled dataset of malicious certificates with assumed benign data, I was able to proceed with model evaluation. However, the limited diversity of the malicious data meant that the model's generalization was constrained to only a few malware families. These challenges were valuable learning experiences that

contributed to the overall success of the project.

The bulk of this project was built with Python. Libraries used include: scikit-learn, numpy, pandas, LIME, Matplotlib. I briefly used River, Seaborn, PyTorch, and Transformers. I gained familiarity with unsupervised learning models like Isolation Forest, Local Outlier Factor, One Class SVM, K Means Clustering, and supervised learning methods like K Nearest Neighbors and Random Forest. I also dipped my toes in Large Language Models such as Character BERT, learning how to embed text and store in a vector database like FAISS. I gained experience working with Graph Convolutional Networks, and tinkered with PyTorch Geometric for graph data.

Chapter 8

Conclusion

This project provided valuable insights into the application of machine learning techniques for detecting malicious TLS certificates. Through the use of various unsupervised and supervised learning models, I was able to assess the effectiveness of different algorithms in identifying anomalies in certificate data. The results highlighted the strengths of models like One-Class SVM and Random Forest, with the latter providing the best overall performance on the dataset.

Ultimately, I learnt that F1 score is not everything when it comes to evaluating a model. Constraints like implementation, time complexity, and interpretability are paramount in ensuring that the model fits into a processing pipeline, and that its predictions can be trusted.

Moving forward, I hope that the methodologies and lessons from this project will be valuable in tackling similar problems in the future, especially in the domain of cybersecurity.

References

1. Abuse.ch. (2021). *SSLBL — Malicious JA3 Fingerprints*. Retrieved from https://sslbl.abuse.ch/ja3-fingerprints/

2. Farshad, K. (2024, November 26). *Understanding Silhouette Score in Clustering*. Medium. https://farshadabdulazeez.medium.com/understanding-silhouette-score-in-clustering-8aedc06ce9c4

3. Fasllija, E., Enişer, H., & Prünster, B. (2019). *Phish-Hook: Detecting Phishing Certificates Using Certificate Transparency Logs*. In *Lecture Notes in Computer Science, Volume 11718* (pp. 318–329). Springer. https://pure.tugraz.at/ws/portalfiles/portal/25394076/156259641564590.pdf

4. Gómez, G., Dell'Amico, M., & Bilge, L. (2022). *Unsupervised Detection and Clustering of Malicious TLS Flows*. Retrieved from https://arxiv.org/pdf/2109.03878

5. Han, X., Cui, S., Qin, J., Liu, S., Jiang, B., Dong, C., Lu, Z., & Liu, B. (2024). *ContraMTD: An Unsupervised Malicious Network Traffic Detection Method based on Contrastive Learning*. In *Proceedings of the ACM Web Conference 2022*, 1680–1689. https://doi.org/10.1145/3589334.3645479

6. Liu, F. T., Ting, K. M., & Zhou, Z. (2008). *Isolation Forest*. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining* (pp. 413–422). IEEE. https://ieeexplore.ieee.org/document/4781136

7. Mounish V. (2024, March 29). *A Comprehensive Guide for SVM One-Class Classifier for Anomaly Detection*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2024/03/one-class-svm-for-anomaly-detection/

8. NCC Group. (2021). *Encryption Does Not Equal Invisibility – Detecting Anomalous TLS Certificates with the Half-Space-Trees Algorithm*. Retrieved from https://www.nccgroup.com/us/research-blog/encryption-does-not-equal-invisibility-detecting-anomalous-tls-certificates-with-the-half-space-trees-algorithm/

9. Ondara, W. (2024, February 6). *11 Best Free SSL Certificate Providers in 2024*. 11 Best Free and Low-Cost SSL Certificate Authorities. https://www.tecmint.com/best-ssl-certificate-authorities/

10. Ostertág, R., & Stanek, M. (2024). *Anomaly Detection in Certificate Transparency Logs*.

11. Splunk. (2023). *Trust Unearned? Evaluating CA Trustworthiness Across 5 Billion Certificates*. Splunk. https://www.splunk.com/en_us/blog/security/trust-unearned-evaluating-ca-trustworthiness-across-5-billion-certificates.html

12. The Spamhaus Project. (2025). *Strengthening Trust and Safety Across the Internet*. Retrieved from https://www.spamhaus.org/reputation-statistics/cctlds/domains/

13. Trust Unearned? Evaluating CA Trustworthiness Across 5 Billion Certificates — Splunk. (2023). Splunk. https://www.splunk.com/en_us/blog/security/trust-unearned-evaluating-ca-trustworthiness-across-5-billion-certificates.html

14. What is an x.509 certificate and how does it work? (2024). CheapSSLShop. https://www.cheapsslshop.com/blog/what-is-an-x509-certificate

Appendix A

Initial Data Columns

1. TLSServerCertIssuer_C
2. TLSServerCertIssuer_L
3. TLSServerCertIssuer_O
4. TLSServerCertIssuer_CN
5. TLSServerCertIssuer_OU
6. TLSServerCertIssuer_ST
7. TLSServerCertSubject_C
8. TLSServerCertSubject_L
9. TLSServerCertSubject_O
10. TLSServerCertSubject_CN
11. TLSServerCertSubject_ST
12. TLSServerCertSubject_OU
13. TLSServerCertIssuedDate
14. TLSServerCertExpiryDate
15. TLSServerCertAltnames
16. SSL/TLS Version
17. Cipher Suites
18. Extensions
19. Elliptic Curves (Groups)
20. Elliptic Curve Point Formats
21. chain_index
22. chain_length

Appendix B

Full List of 31 Features

B.1 Engineered Features (n = 12)

- chain_length
- zscore
- cert_lifetime_days
- altnames_count
- is_CN_in_SAN
- number_of_missing_fields
- issued_from_free_CA
- length_of_domain
- domain_entropy
- is_malicious_hash
- has_suspicious_fields
- has_inner_tld_in_subdomain

B.2 Suspicious TLD One-Hot Encodings (n = 6)

- suspicious_tld_tech
- suspicious_tld_info
- suspicious_tld_xyz
- suspicious_tld_top
- suspicious_tld_vip
- suspicious_tld_other

B.3 Certificate Field Presence Flags (n = 13)

- has_TLSServerCertIssuer_L
- has_TLSServerCertIssuer_OU
- has_TLSServerCertIssuer_ST
- has_TLSServerCertSubject_C
- has_TLSServerCertSubject_L
- has_TLSServerCertSubject_ST
- has_TLSServerCertSubject_O
- has_TLSServerCertIssuer_C
- has_TLSServerCertIssuer_O
- has_TLSServerCertIssuer_CN
- has_TLSServerCertSubject_OU
- has_TLSServerCertSubject_CN
- has_TLSServerCertAltnames