

Santa Clara University

## Scholar Commons

---

Computer Science and Engineering Senior  
Theses

Engineering Senior Theses

---

5-29-2024

### ProtectNIC - SmartNIC Ransomware Detection

Arnav Choudhury

Eason Liu

Anson Xu

Follow this and additional works at: [https://scholarcommons.scu.edu/cseng\\_senior](https://scholarcommons.scu.edu/cseng_senior)



Part of the [Computer Engineering Commons](#)

---

**SANTA CLARA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Date: May 29, 2024

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

**Arnav Choudhury**  
**Eason Liu**  
**Anson Xu**

ENTITLED

**ProtectNIC - SmartNIC Ransomware Detection**


BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING



---

Thesis Advisor



---

Department Chair

# **ProtectNIC - SmartNIC Ransomware Detection**

by

Arnav Choudhury  
Eason Liu  
Anson Xu

Submitted in partial fulfillment of the requirements  
for the degree of  
Bachelor of Science in Computer Science and Engineering  
School of Engineering  
Santa Clara University

Santa Clara, California  
May 29, 2024

# **ProtectNIC - SmartNIC Ransomware Detection**

Arnav Choudhury

Eason Liu

Anson Xu

Department of Computer Science and Engineering  
Santa Clara University  
May 29, 2024

## **ABSTRACT**

Ransomware, a form of malware that restricts access to data until a ransom is paid, accounts for 20% of all cyber crimes. Although companies and organizations often require their personnel to take training for awareness of such bad actors, social engineering is constantly evolving and ransomware slips through the cracks every year. In this paper, we suggest a system that would help detect ransomware using a Smart Network Interface Card (SmartNIC) which runs machine learning algorithms to detect ransomware before it enters the system. This relieves computers in the network of the burden of detecting malware, freeing CPU capacity to do other work. Using previous network data captured while running ransomware binaries, we trained models that accurately predict whether network traffic contains ransomware using only packet payloads. Our results suggest that payload analysis could be a valid in-network solution to malware detection.

# Table of Contents

<b>Table of Contents</b>	<b>3</b>
<b>List of Figures</b>	<b>5</b>
<b>Chapter 1: Introduction</b>	<b>6</b>
1.1 Motivation	6
1.2 Background and Current Innovations	7
1.2.1 SmartNICs	7
1.2.2 In-Network Devices and Machine Learning	8
1.2.3 Ransomware Detection	8
1.3 Proposed Solution	9
<b>Chapter 2: Use Cases</b>	<b>10</b>
2.1 Stakeholders	10
2.2 Use Case Diagram	10
<b>Chapter 3: Design and Rationale</b>	<b>12</b>
3.1 Proposed Requirements	12
3.1.1 Functional Requirements	12
3.1.2 Non-Functional Requirements	12
3.2 Datasets	13
3.3 Model	14
<b>Chapter 4: Technologies</b>	<b>15</b>
4.1 Hardware	15
4.2 Software	15
<b>Chapter 5: Implementation</b>	<b>16</b>
5.1 Development Timeline and Risks	16
5.1.1 Development Phases	16
5.1.2 Project Timeline	16
5.1.3 Risk Analysis	17
5.2 Prototyping	17
5.2.1 Operation Classification	17
5.2.2 Per-Packet Classification	19
5.2.3 Cosine Similarity of Packet Flows	21
5.3 System Design	23
5.4 Data Preprocessing	23
5.4.1 Class Imbalance	24
5.5 Model Training	25
5.5.1 XGBoost	25
5.5.2 Random Forest	26

<b>Chapter 6: Results and System Evaluation</b>	<b>28</b>
6.1 Metrics	28
6.2 Confusion Matrices	29
6.3 Inference Times	29
<b>Chapter 7: Constraints and Standards</b>	<b>31</b>
7.1 Constraints	31
7.2 Standards	31
<b>Chapter 8: Societal Issues</b>	<b>33</b>
8.1 Ethical	33
8.2 Economic	33
8.3 Compassion	34
8.4 Sustainability	34
8.5 Lifelong Learning	34
<b>Chapter 9: Conclusions</b>	<b>36</b>
<b>Chapter 10: Acknowledgements</b>	<b>37</b>
<b>References</b>	<b>38</b>
<b>Appendices</b>	<b>40</b>
Appendix A: Final Hyperparameters for Resulting Models	40
Appendix B: Table of .pcap files used and their Sources	42

# List of Figures

2.1	Use Case Diagram for ProtectNIC System . . . . .	10
5.1	Project Timeline . . . . .	16
5.2	XGBoost Classifier confusion matrix for operation classification . . . . .	18
5.3	Random Forest Feature Importances (Per-packet Classification) . . . . .	20
5.4	Cosine similarity between representative flows of each .pcap file . . . . .	22
5.5	System Architecture for Model Training . . . . .	23
5.6	Pie chart illustrating class imbalance of .pcap dataset . . . . .	24
5.7	Accuracy scores of XGBoost Classifier after every incremental training session. . . . .	26
5.8	Visualized metrics for final XGBoost and random forest classifiers. . . . .	28
5.9	Confusion matrices for final models for packet flow classification. . . . .	29
5.10	Inference times on testing dataset for final XGBoost classifier . . . . .	30

# Chapter 1: Introduction

## 1.1 Motivation

Ransomware is a form of malware that threatens to expose an individual's personal data or permanently restricts access to it unless a ransom is provided. While basic ransomware may merely immobilize the system without harming any files, more sophisticated malware employs a method known as cryptoviral extortion. This method encrypts the victim's files, rendering them unattainable, and requires a ransom in exchange for decryption. There are 2 types of ransomware attacks that are popular. The first type is the crypto ransomware that encrypts files in a system and has the user pay a ransom to unlock their files. The second type of ransomware is locker ransomware which locks the user out of the system. The mouse and keyboard are left enabled so that the user can pay to unlock the system. This malware does not delete files, but instead locks the system until a payment is made. Ransomware is most often spread as a Trojan, a virus that is disguised as another program. Phishing emails, scareware, and other forms of social engineering are common ways to install ransomware on a victim's computer. As of 2022, Ransomware accounts for 20% of all cyber crimes [1].

Existing solutions to ransomware like training personnel are always susceptible to human error. Frequent backups can be used to prevent data loss, but it may be expensive to maintain such storage. We use recent developments in the field of machine learning to combine machine learning classification with SmartNIC technology to improve network security against ransomware attacks. ProtectNIC implements a machine learning model that classifies incoming ransomware traffic while performing well enough to run on a SmartNIC without producing significant network latency.



## 1.2 Background and Current Innovations

### 1.2.1 SmartNICs

Smart Network Interface Chips (SmartNICs) are programmable accelerators, often used to offload network-related tasks off of a host server. SmartNICs are also referred to as Data Processing Units (DPUs) and Infrastructure Processing Units (IPUs). SmartNICs are increasingly being used in data centers to reduce large overhead tasks like network virtualization, security and storage services that do not directly serve clients [2, 3]. In cloud architectures, these overhead tasks take away processing power from CPUs, increasing the cost of cloud computing [3]. There are three main types of SmartNICs.

**1. ASIC based NICs** are built with custom ASIC designs, which often have embedded CPU cores to handle new functionality [3].

**2. Multicore SoC based NICs** provide much better programmability than ASICs at the cost of performance. However it often has higher latency and poorer scalability. The Nvidia BlueField (DPU) that ProtectNIC uses is an example of an SoC based NIC.

**3. Field Programmable Gate Arrays (FPGAs)** are integrated circuits with generic logic blocks that can be reprogrammed after manufacturing. FPGAs can be programmed to work as SmartNICs. FPGAs are often described as a balance between ASIC based NICs and SoC based NICs because they combine the flexibility of SoC programmability with the performance of custom ASIC hardware [3].

Because our project focuses on the software side of ransomware detection, we use the Nvidia BlueField, an SoC based NIC which gives the most software flexibility.

### 1.2.2 In-Network Devices and Machine Learning

There are many papers demonstrating machine learning on programmable in-network devices. For example, Planter [5] and MAP4 [6] are frameworks that allow developers to map machine learning models onto programmable network devices. In-network machine learning has also been used to detect network attacks by embedding random forest classification in a programmable switch [7]. It has also been demonstrated that high accuracy packet classification can be done on SmartNICs with “minor performance degradation” [8].

A big motivation for machine learning on programmable network devices is network security, and because many data centers already have such architecture in place, it could be a cheaper solution than having dedicated hardware to perform machine learning. ProtectNIC is similar in the way that it tries to offload ransomware detection, a security task, from the host server it is connected to.

### 1.2.3 Ransomware Detection

There is plenty of literature on ransomware detection, including papers that provide their datasets and pre-trained models online for anyone to download [9]. A big issue when training machine learning models to detect malware is the consideration of zero-day variants. Because new forms of malware may use exploits not seen before in the training dataset, they can be difficult to detect. Existing research has also been done to consider these risks [10]. ProtectNIC uses such existing research and datasets to create an accurate ransomware detection model that considers zero-day variants of ransomware.

## 1.3 Proposed Solution

ProtectNIC is a SmartNIC based ransomware detector using the Nvidia BlueField DPU, which involves training a model able to classify ransomware network traffic without significantly impacting network latency. This solution is only possible because of recent developments in Machine Learning and existing programmable in-network hardware. Building on top of previous research on ransomware detection using machine learning, ProtectNIC will allow a SmartNIC to detect anomalous activity in network traffic. This would offload work from the CPU, so it can use more computational power on the primary workload.

# Chapter 2: Use Cases

## 2.1 Stakeholders

**Developers:** Santa Clara University Senior Design students

**Customers:** Companies with existing in-network devices, e.g. Cloud providers interested in protecting customer data from ransomware attacks.

**Users:** Security Analysts and Network Administrators for interested companies.

## 2.2 Use Case Diagram

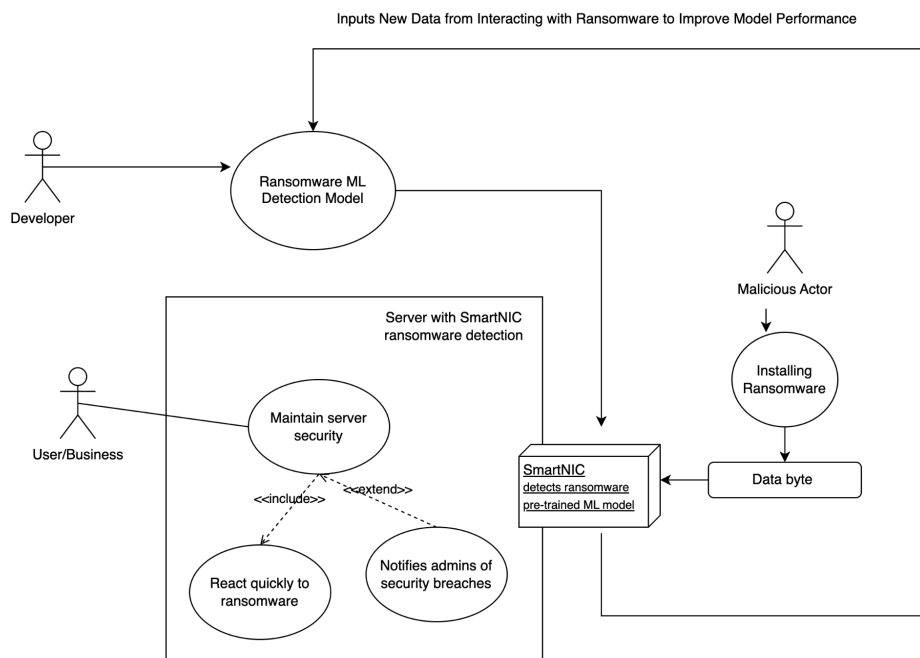


Figure 2.1: Use Case Diagram for ProtectNIC System

Figure 2.1 shows how ProtectNIC could be used in a medium to large software company that maintains their own data servers. The ransomware detection system would be installed on existing in-network devices such as the SmartNIC, and drop incoming packets that are

classified as ransomware. Recorded network traffic could be used to regularly improve the model with recent data.

## Chapter 3: Design and Rationale

ProtectNIC implements a machine learning model to detect ransomware. The trained model will be used in the BlueField DPU to detect malicious packets in a network stream. Packets deemed malicious by the model should be dropped while other packets should be forwarded to the host system. Performance of the network should be evaluated with and without the packet classification. Although increased latency is expected, ProtectNIC minimizes as much performance overhead as possible.

### 3.1 Proposed Requirements

#### 3.1.1 Functional Requirements

There are three major functional requirements for this project:

1. Ransomware detection model classifies ransomware network traffic and runs on a SmartNIC (Bluefield DPU).
2. System detects ransomware threats before they infiltrate the network, reducing the risk of system compromise.
3. Machine learning models are able to update its parameters to match new training data in order to adapt to evolving ransomware threats.

#### 3.1.2 Non-Functional Requirements

ProtectNIC has two major objectives: accurate ransomware detection, and minimal latency while doing so. Here are some metrics that would help us achieve these goals:

- **Network speed / Latency.** The slowdown caused by packet classification has to be negligible, or at least worth the security advantages. The network speed must at least be 80% of its original value while classifying ransomware.
- **Model performance.** The detection model should detect ransomware packets relatively accurately, meaning an f1-score of above 80%, given that previous research has achieved similar numbers [9].
- **Model Size.** Since ProtectNIC will work on programmable network devices, which will often have constrained resources, the model should be under 64 GB in size (memory limit for Bluefield DPU 3).

## 3.2 Datasets

The training dataset consists of open repositories and contemporary research to train the ransomware detection model. The first is the Ransomware PCAP repository [11], which provides samples of network activity recorded during the execution of binaries of different ransomware applications. There are a total of 39 families of ransomware in the repository, each with multiple PCAP files of sizes from a couple hundred megabytes to several gigabytes. The total size of the dataset approximated 120 gigabytes in size. The repository has recent data with packet captures from 2015 to 2021. In addition, the same research also provides pre-trained models [12] that can be used as a baseline for classification.

For goodware network traffic data, the UNSW-NB15 [13] dataset provides packet captures of a mixed set of applications from regular day to day activities. USTC-TFC2016 [14] provides packet captures recorded from applications like BitTorrent, Facetime, and World of Warcraft. We combine portions of both datasets to create network traffic data that resembles real-world network activity.

### 3.3 Model

The ProtectNIC model attempts to detect ransomware in network packets. There is previous work [12] that has used Neural Networks (NN) and Convolutional Neural Networks (CNN) that provides good baselines for how well our model should perform. Previous research also [15] suggests that decision tree models and random forests perform well to classify ransomware traffic. Considering their previous performance, we train XGBoost and random forest models which have been shown to have a good reputation for avoiding overfitting and producing models that make efficient use of computing resources. Setting depth limits also helps our models generalize to outliers like zero-day attacks, a major risk for malware detection models [10].



# Chapter 4: Technologies

## 4.1 Hardware

The ransomware detection models are intended to classify network traffic on a NVIDIA BlueField-3 Data Processing Unit (DPU). Model training and testing is performed on the Wiegand Advanced Visualization Environment (WAVE) High-Performance Computing (HPC) cluster. This cluster comprises powerful multi-core/multi-socket servers with high-performance storage, GPUs, and large memory, interconnected by a fast network. It is designed to support high computation and memory-intensive programs. Complementing these, a standard laptop, specifically the MacBook M1 Pro, serves as the primary computing platform.

## 4.2 Software

On the software side, Python serves as the core programming language, while machine learning libraries such as Scikit-learn, XGBoost, and Keras are used to train and evaluate models. Additionally, visual libraries including Seaborn and Matplotlib are employed to produce data visualizations, enhancing the interpretability of results and insights derived from the analyses.

# Chapter 5: Implementation

## 5.1 Development Timeline and Risks

### 5.1.1 Development Phases

**Phase 1: Research.** Research and understand machine learning development. Read previous work on malware and ransomware detection.

**Phase 3: Design.** Establish project guidelines and constraints. Decide major technologies to use throughout the project.

**Phase 4: Prototyping.** Create and test prototypes for ransomware detection models.

**Phase 5: Training.** Build ransomware detection system based on prototypes and previous research.

**Phase 6: Evaluation.** Evaluate model performance in terms of accuracy and speed.

### 5.1.2 Project Timeline

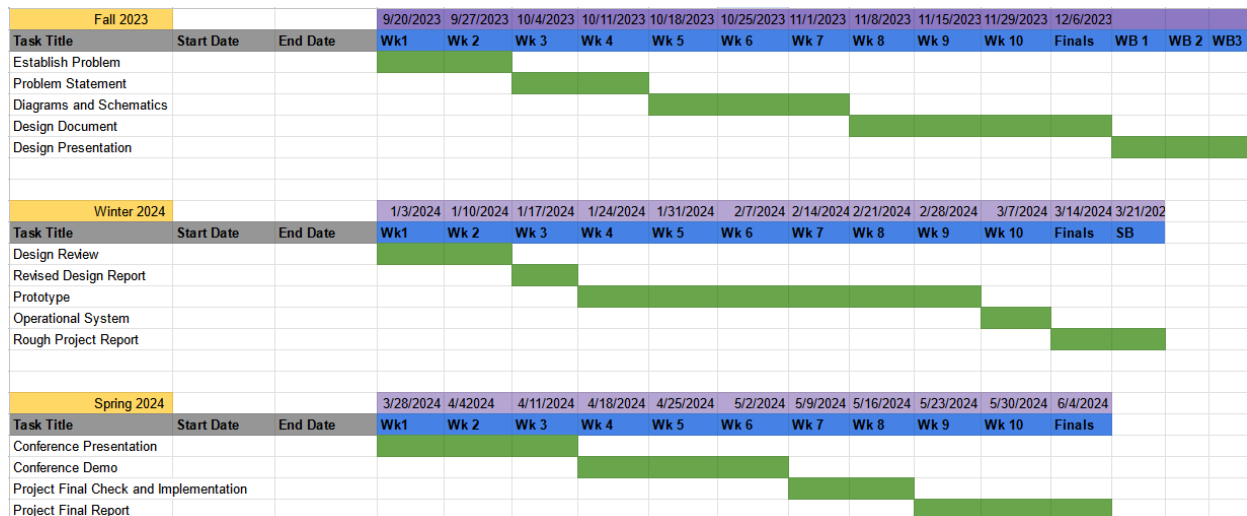


Figure 5.1: Project Timeline

### 5.1.3 Risk Analysis

Risk	Description	Mitigation
Delayed Arrival of Bluefield	Unable to test model on intended SmartNIC hardware.	N/A
Data Privacy Concerns	The ML model may process sensitive information, leading to privacy concerns.	Use data anonymization and encryption practices.
Model Drift	Ransomware attacks may change over time, leading to a decline in the model's performance.	Update model regularly with new data to adapt to changing patterns in ransomware attacks.
Resource Overhead	Running ML models can be resource-intensive, potentially impacting system performance.	Adjust model hyperparameters to improve model performance.
User Awareness	Users may undermine the effectiveness of the ransomware prevention system through risky behavior.	Conduct regular cybersecurity awareness training for users. Encourage best practices such as avoiding suspicious emails and keeping software up to date.
False Positives/Negatives:	The ML model may incorrectly classify benign activities as ransomware (false positives) or fail to detect actual ransomware attacks (false negatives)	Regularly update the ML model based on new data. Use a combination of machine learning and traditional rule-based methods to reduce false positives and negatives.

## 5.2 Prototyping

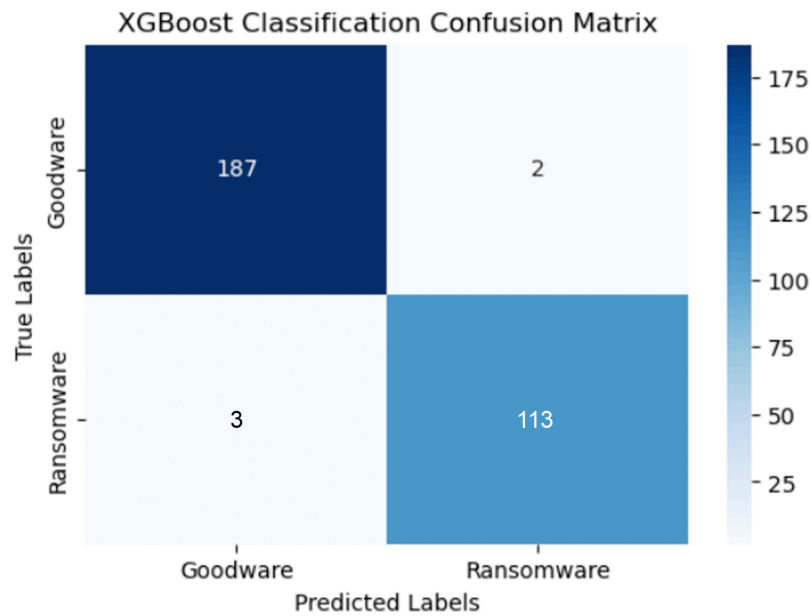
### 5.2.1 Operation Classification

This prototype attempts to reproduce the results in a previous paper [16] that trained a model to classify computer operations as either ransomware or goodware with high accuracy. The dataset

includes a log of computer operations, including API calls and file operations. Sample features were generated dynamically from data obtained from VirusShare.com. Using this dataset, we trained a model to classify ransomware operations using the default XGBClassifier from the XGBoost API. With a 80/20 split for training and testing data, this model was able to achieve similar accuracy to that of the paper. The table and confusion matrix below shows the resulting performance metrics:

Metric	Value
Accuracy	0.9836
Precision	0.9826
Recall	0.9741
F1-Score	0.9783

**Table 5.1:** Metrics for XGBoost model for operation classification



**Figure 5.2:** XGBoost Classifier confusion matrix for operation classification

XGBoost performs well on all metrics and boasts a high rate of true positives and true negatives. This helped us determine that XGBoost was a good choice for malware detection, and gave us a preliminary idea of whether ransomware detection through network traffic would be feasible.

### 5.2.2 Per-Packet Classification

As a first attempt to classify network traffic, we trained a random forest classifier that predicts whether a packet is part of a ransomware attack based on its fields. For the sake of practicality, a small set of packet capture (.pcap) files was used:

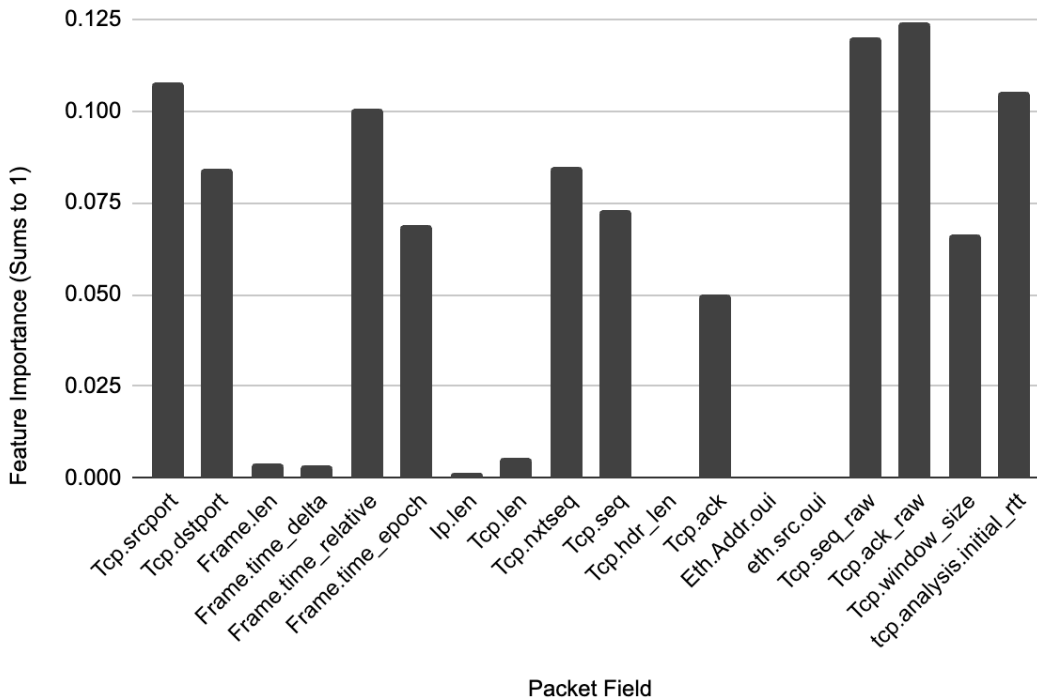
File Name	Source Dataset
Hive_06082021.pcap	Ransomware PCAP Repository
CryptoFortress_12032015.pcap	Ransomware PCAP Repository
client_normal_1_210110.pcap	UNSW-NB15
Gmail.pcap	USTC-TFC2016

**Table 5.2:** list of .pcap files used to train per-packet classifier

Because a majority of the network traffic followed the TCP protocol, we limited our dataset to TCP packets. Tshark was used to extract packet fields from the TCP packets, which includes fields from TCP protocol, Ethernet, and the IP protocol. Each sample of the resulting dataset represents a TCP packet with features representing packet fields. Each packet was also labeled as ransomware or goodware according to their source file.

Using default parameters, Scikit-learn was used to train a random forest classifier on a 80/20 train test split. The graph below shows the feature importances of each packet field. Packet fields are labeled by their filter name in Wireshark.

### Random Forest Feature Importances (Per-packet Classification)



**Figure 5.3:** Random Forest Feature Importances (Per-packet Classification)

Although the model achieves both an accuracy and f1-score of 1.0, the feature importances shown in Figure 5.3 show us that the packet fields with the greatest importances are usually client specific. The model mainly uses differences in network speed (frame.time\_delta and tcp.analysis.initial\_rtt), port numbers (tcp.srcport and tcp.dstport) and raw sequence numbers (tcp.seq\_raw) to make predictions. Since we only have a few clients in our dataset, this makes it easier for the model to rely on client specific features. Per-packet classification does not provide the generalized approach needed to classify real-world network traffic.

### 5.2.3 Cosine Similarity of Packet Flows

Instead of fields of individual packets, much previous work [17-18] classifies network traffic by looking at the characteristics of packet flows. The first advantage of this is that packet flows give the bigger picture of network activity and have features that can be more easily generalized. The second advantage is that classification needs to be done only once per packet flow, meaning less time is spent on prediction for each packet.

In this prototype, we take inspiration from both packet flows and tokenization used in natural language processing to vectorize network traffic data. For the same reasons as mentioned before, we only use packets following the TCP protocol. First, we define a packet flow as a set of packets with the same source IP address (`ip.src`), destination IP address (`ip.dst`), source TCP port (`tcp.srcport`), and destination TCP port (`tcp.dstport`). In addition, the relative sequence numbers (`tcp.seq`) must be sequential. If relative sequence numbers are not sequential, we assume a new packet flow is started.

Once packets are split into flows, 4 bytes are taken from the end of the TCP payload of each packet. We count the frequency of each unique 4 byte sequence and place these values into a vector representing a single packet flow. The order of each vector is kept consistent, and each unique 4 byte chunk's frequency is represented by a unique index in the vector.

Finally, the cosine similarity is calculated between the vectors representing the longest flows from each .pcap file. Given the vectors are represented by  $A$  and  $B$ , the cosine similarity is calculated between each pair of vectors as follows:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

The cosine similarity between every pair of vectors for the files in our dataset is shown in Figure 5.4.

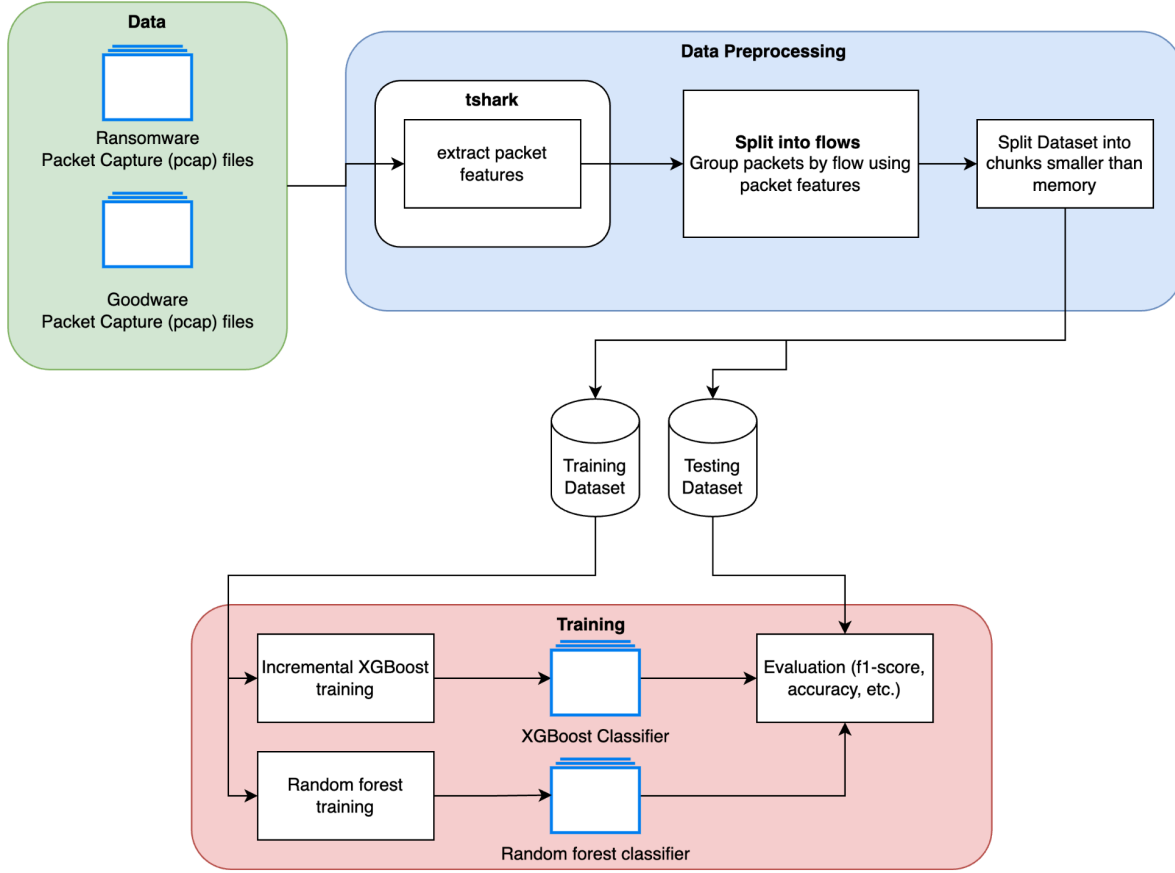
filename	cerber_1008201	CryLock_240120	Cryptofortress_06082021	CryptoShield_31	Hive_06082021	maktub_120420	MRCR_1501201	netwalker_2101	RansomX_2806	Scorab_181020	Spora_1705201	BitTorrent.pcap.c	client_normal_1	client_normal_2	client_normal_3	Gmail.pcap.csv	normal_1	pcap.c	normal_2	pcap.c	normal_3	pc	WorldOWWarcraft
cerber_1008201	1	0.9410560448	0.9362239117	0.4039280545	0.9243817728	0.7965842865	0.9364289211	0.9443218006	0.938677856	0.9339313986	0.9850786023	0.000368386711	0.00012279557	0.00012279557	0.00012279557	0	6.43E-06	1.21E-05	0.000175347011	0.00177240142			
CryLock_240120	0.9410560448	1	0.9982485892	0.4248850364	0.9585169225	0.8375686443	0.9585684894	0.995352006	0.9988974463	0.9885642214	0.9401564999	0	2.61E-06	2.61E-06	2.61E-06	0	2.79E-06	2.89E-06	3.39E-05	9.04E-06			
Cryptofortress_06082021	0.9362239117	0.9982485892	1	0.4216799502	0.9579561336	0.8315650017	0.9998639953	0.9903796118	0.9999426328	0.9840528574	0.9363658951	0	1.85E-06	1.85E-06	1.85E-06	0	1.18E-07	3.25E-07	4.72E-05	5.33E-06			
CryptoShield_31	0.4039280545	0.4248850364	0.4216799502	1	0.4054849742	0.3915149691	0.4218195298	0.4258911727	0.4222003836	0.4229112381	0.4023999568	0	0	0	0	0	3.24E-06	1.20E-07	0	0			
Hive_06082021	0.9243817728	0.9585169225	0.9579561336	0.4054849742	1	0.7992723641	0.9572302605	0.9592350263	0.9577740927	0.9756128979	0.9475947138	0	0	0	0	0	7.98E-06	3.73E-05	3.74E-05	4.15E-05	0.000241175552		
maktub_120420	0.7965842865	0.8375686443	0.8315650017	0.3915149691	0.7992723641	1	0.8318446364	0.8397947364	0.832636695	0.8341631177	0.7934539581	0	0.00834566244	0.00834566244	0.00834566244	0	2.84E-05	5.20E-05	7.61E-08	0			
MRCR_1501201	0.9364289211	0.9585684894	0.9998639953	0.4218195298	0.9572302605	0.8318446364	1	0.9907180031	0.9998962188	0.9838047996	0.9362195086	0	2.45E-06	2.45E-06	2.45E-06	0	9.50E-07	1.08E-06	5.12E-05	1.84E-05			
netwalker_2101	0.9443218006	0.995352006	0.9903796118	0.4258911727	0.9592350263	0.8397947364	0.9907180031	1	0.9917290553	0.9915895957	0.9441552341	0	0	0	0	0	4.49E-06	4.29E-06	4.30E-07	1.51E-05			
RansomX_2806	0.938677856	0.988974463	0.9999426328	0.4222003836	0.9577740927	0.832636695	0.9998962188	0.9917290553	1	0.9946333999	0.9364691952	0	0	0	0	0	1.30E-07	3.10E-07	5.09E-05	4.99E-06			
Scorab_181020	0.9339313986	0.9885642214	0.9840528574	0.4229112381	0.9756128979	0.8341631177	0.9838047996	0.9910885957	0.9848333999	1	0.940166385	0	0	0	0	0	5.32E-10	1.36E-09	1.58E-06	1.15E-05			
Spora_1705201	0.9850786023	0.9401564999	0.9363658951	0.4023999568	0.9475947138	0.7934539581	0.9362195086	0.9441552341	0.9364691952	0.940166385	1	3.08E-05	0.00012333119	0.00012333119	0.00012333119	6.54E-05	6.56E-05	8.94E-05	9.77E-05	0.000712052981			
BitTorrent.pcap.c	0.000368386711	0	0	0	0	0	0	0	0	0	3.08E-05	1	0	0	0	0	0	0	0	0			
client_normal_1	0.00012279557	2.61E-06	1.85E-06	0	0.00834566244	2.45E-06	0	0	0	0.00012333119	0	0	1	0	0	0	0.00021660949	0	0	0			
client_normal_2	0.00012279557	2.61E-06	1.85E-06	0	0.00834566244	2.45E-06	0	0	0.00012333119	0	0	0	0	1	0	0	0.00021660949	0	0	0			
client_normal_3	0.00012279557	2.61E-06	1.85E-06	0	0.00834566244	2.45E-06	0	0	0.00012333119	0	0	0	0	0	1	0	0.00021660949	0	0	0			
Gmail.pcap.csv	0	0	0	0	7.98E-06	0	0	0	0	0	6.54E-05	0	0	0	0	0	0	0	0	0			
normal_1.pcap.c	6.43E-06	2.79E-06	1.18E-07	3.24E-06	3.73E-05	2.84E-05	9.50E-07	4.49E-06	1.30E-07	5.32E-10	6.56E-05	0	0	0	0	0	0	0	0	0			
normal_2.pcap.c	1.21E-05	2.89E-06	3.25E-07	1.20E-07	3.74E-05	5.20E-05	1.08E-06	4.29E-06	3.10E-07	1.36E-09	8.94E-05	0	0.00021660949	0.00021660949	0.00021660949	0	0	0	0	0	0.9672933275		
normal_3.pc	0.000175347011	3.39E-05	4.72E-05	0	4.15E-05	7.61E-08	5.12E-05	4.93E-07	5.09E-05	1.58E-06	9.77E-05	0	0	0	0	0	0	0	0	0	0	0	
WorldOWWarcraft	0.00177240142	9.04E-06	5.33E-06	0	0.000241175552	0	1.84E-05	1.51E-05	4.99E-06	1.15E-05	0.000712052981	0	0	0	0	0	0	0	0	0	0	0	-1

**Figure 5.4:** Cosine similarity between representative flows of each .pcap file

Filenames of ransomware and goodware are highlighted in blue and yellow respectively. Similarities closer to 1 are highlighted in green while similarities closer to 0 are highlighted in red. The average similarity between any two ransomware applications was 0.84, while the average similarity between any two goodware and ransomware applications was 0.0003. From these results, it is clear that a representative packet flow of ransomware applications is likely to be more similar to that of other ransomware applications. One outlier is Cryptoshield, which has relatively lower similarities with other ransomware. We also see some similarities between goodware, but this should not affect our model performance. We use the results from this prototype as the basis for our final ransomware detection models.



## 5.3 System Design



**Figure 5.5:** System Architecture for Model Training

Figure 5.5 is a high level visual representation of how the final XGBoost classifier and random forest classifier are trained. More detail on each step is provided in the following sections.

## 5.4 Data Preprocessing

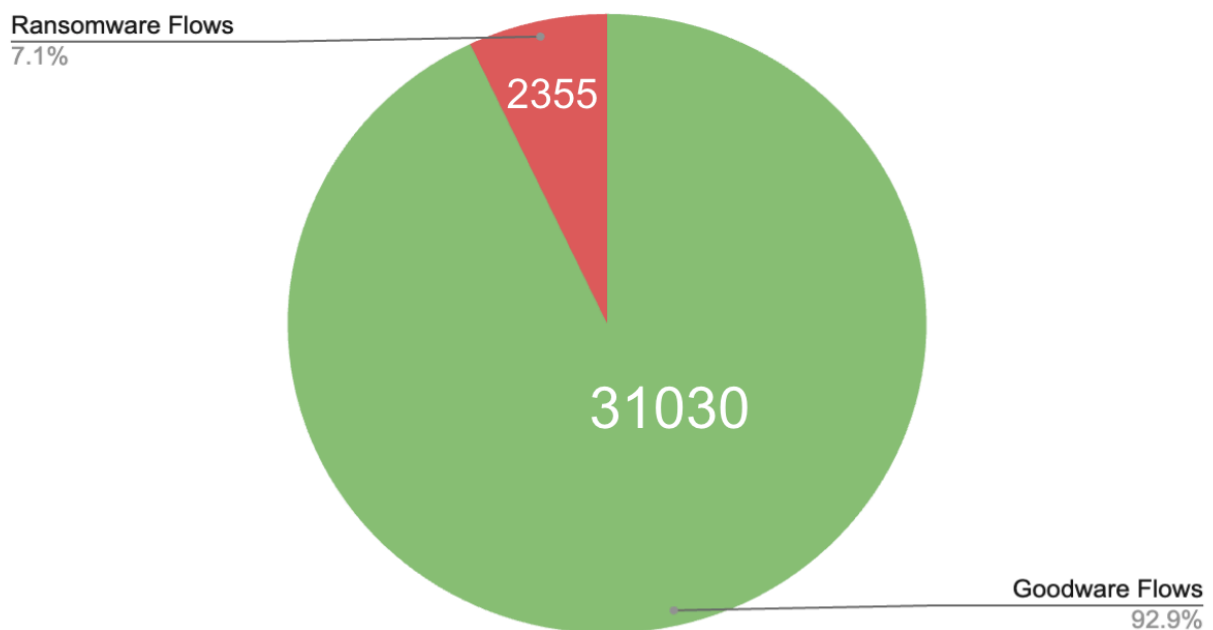
We use 30 Gigabytes of packet capture files from public datasets [11], [13] and [14] as our input data. A list of the files we use and their sources can be found in Appendix B.

During the data preprocessing step, all .pcap files are first fed into Tshark, which extracts the source IP address (ip.src), destination IP address (ip.dst), TCP source port (tcp.srcport), TCP

destination port (tcp.dstport), raw sequence number (tcp.seq\_raw), relative sequence number (tcp.seq), and TCP payload (tcp.payload) from each TCP packet. These features are then used to split the packets into flows as defined in the cosine similarity section, and duplicate packet flows are removed to prevent data leakage. Finally, the payloads of each packet flow is placed on separate rows of a .csv file. Because some files may be too large to load into memory, the dataset is split into chunks smaller than system memory. 80% of dataset chunks are placed in the training dataset, and 20% is used for the testing dataset.

#### 5.4.1 Class Imbalance

##### Ransomware/Goodware Flows in Dataset



**Figure 5.6:** Pie chart illustrating class imbalance of .pcap dataset

Despite having larger file sizes, the .pcap files for ransomware binaries often had fewer packet flows with more packets in a single flow. On the other hand, goodware tended to have a large number of shorter packet flows. In addition to ransomware network traffic being more difficult to

find, this resulted in a class imbalance of mostly goodwill flows. Out of a total of 33385 packet flows, only 2355 were ransomware flows.

## 5.5 Model Training

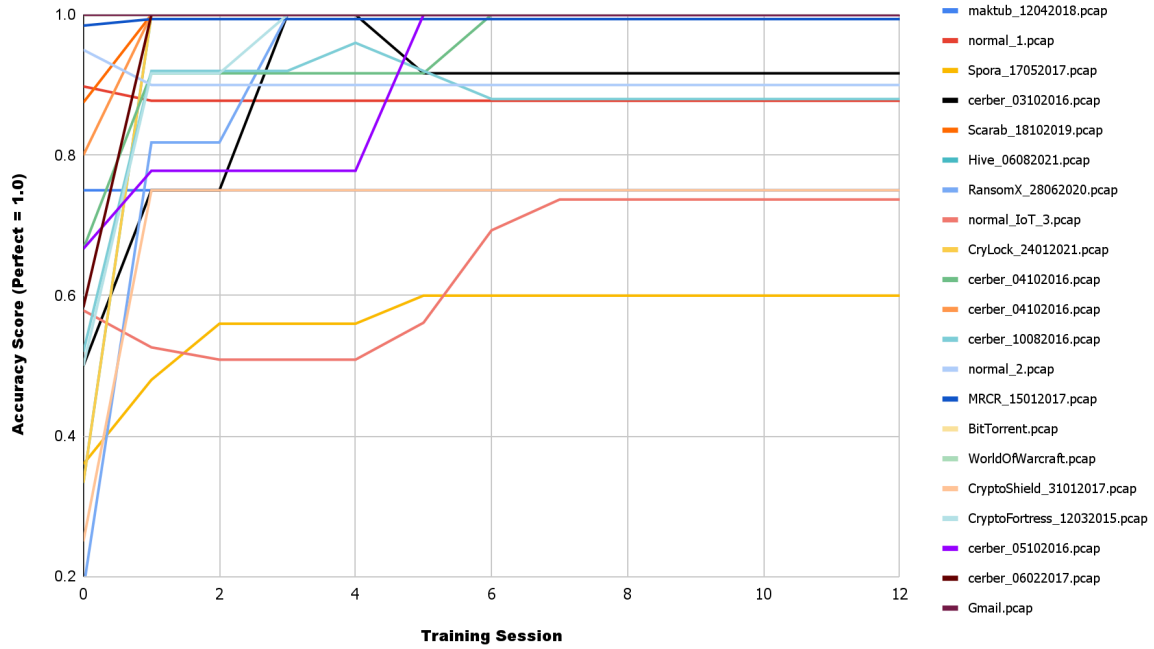
Both the XGBoost and random forest classifiers were trained on the same dataset produced by the dataset preprocessing step.

### 5.5.1 XGBoost

Since the built-in XGBoostClassifier does not support incremental learning, a custom XGBoost model was built using the XGBoost API. The hyperparameters are listed in Appendix A. Our model uses hinge loss for binary classification, and incrementally trains on chunked dataset files from the training dataset. 13 separate training sessions are performed, each adding 10 estimators to the previous model (resulting in a total of 130 estimators). To prevent overfitting, a max depth of 6 is also provided.

Figure 5.7 shows the accuracy improvement after each incremental training session. Accuracy is calculated for each .pcap file in the testing dataset.

**Accuracy Score per XGBoost Incremental Training Iteration (Separated by file)**



**Figure 5.7:** Accuracy scores of XGBoost Classifier after every incremental training session

After the seventh training iteration, model predictions do not change even with additional data. We also see outliers like Spora, that perform poorly even after additional training. A possible reason for this result is both a lack of data, and shorter packet flows in Spora having similarities with goodware. Nevertheless, the increasing accuracy shows that this is a promising approach to updating existing models with contemporary data.

### 5.5.2 Random Forest

With the WAVE system, we were able to fit the entire dataset into memory and train a random forest model on the entire dataset. Scikit-learn's implementation of random forest does not support incremental training, and workarounds like Dask Dataframes must be used to chunk larger datasets. However, this still means that the model needs to be retrained on the entire

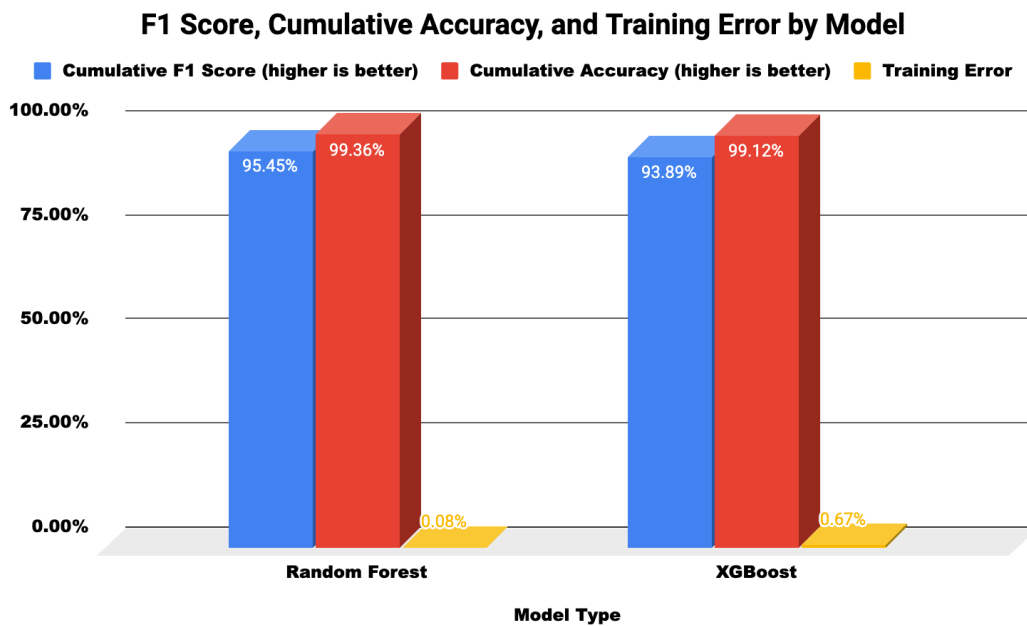
dataset when the model needs to be updated. Our model uses 100 estimators, and model hyperparameters are provided in the Appendix A.

## Chapter 6: Results and System Evaluation

The final models are evaluated against the testing dataset, which contains a total of 6677 packet flows. More specifically, the testing dataset contains 6206 goodwill and 471 ransomware packet flows.

### 6.1 Metrics

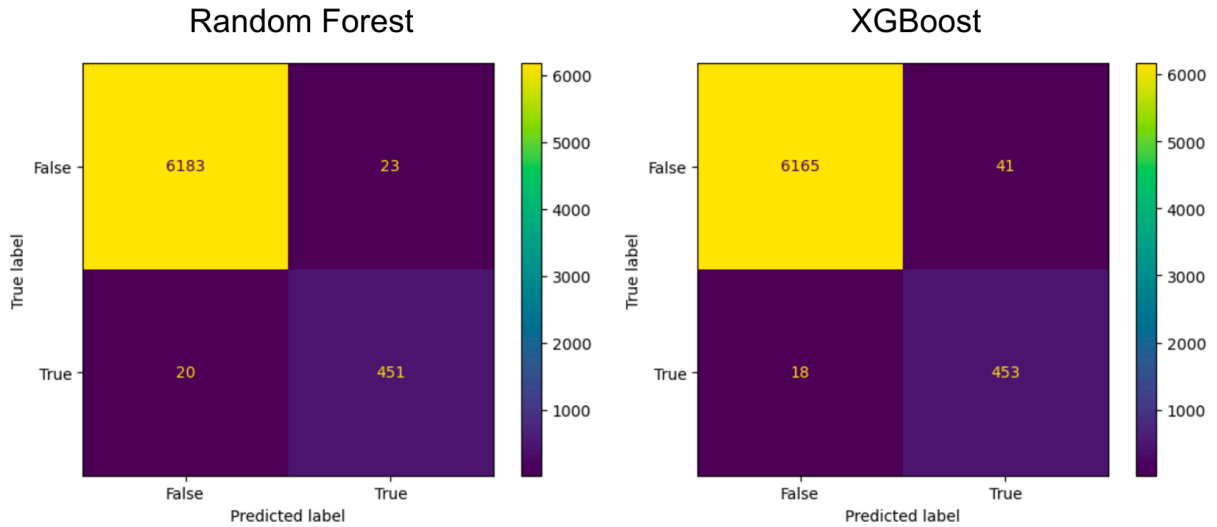
We evaluate the classification performance of both models using standard metrics provided by scikit-learn. Cumulative f1-score and accuracy are calculated using the testing dataset, and training error is calculated using the training dataset. Metrics for both models are shown in figure 5.8.



**Figure 5.8:** Visualized metrics for final XGBoost and random forest classifiers

Random Forest performs slightly better than XGBoost in both f1-score (0.95) and cumulative accuracy (0.99). We hypothesize that this could partially be attributed to the class imbalance of the dataset.

## 6.2 Confusion Matrices



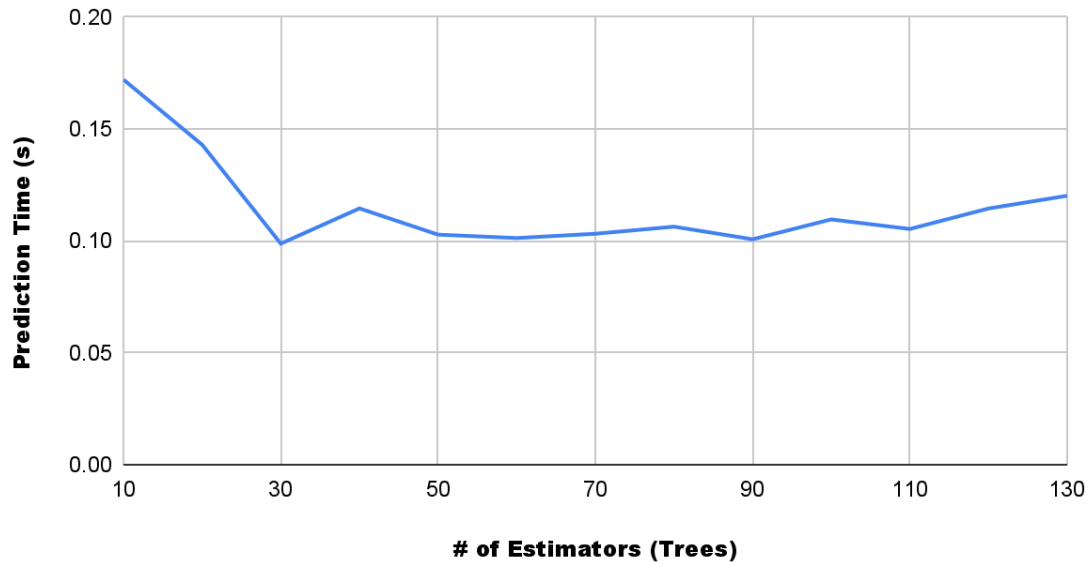
**Figure 5.9:** Confusion matrices for final models for packet flow classification

Goodware and ransomware samples are labeled “False” and “True” respectively. Figure 5.9 illustrates that ransom forest and XGBoost both boast a high rate of true positives and true negatives, demonstrating that the high accuracy of the models is not a result of class imbalance.

## 6.3 Inference Times

Other than accurate classification of ransomware packet flows, our model needs to perform well on in-network devices without significantly affecting network latency. Inference times of random forest and XGBoost were measured on a Macbook M1 Pro. These times were measured for making predictions on the testing dataset, a total of 6677 packet flows.

### XGBoost Inference Times per Estimators



**Figure 5.10:** Inference times on testing dataset for final XGBoost classifier

Figure 5.10 shows the inference times for the XGBoost model at different stages of incremental training. Where XGBoost performs most accurately (80 to 130 estimators), the models have an average of 0.109 second inference times. Random forest makes the same predictions in 3.127 seconds, about 31 times slower.



# Chapter 7: Constraints and Standards

## 7.1 Constraints

Our project faces several constraints that shape our development process. Firstly, time is a significant constraint, with approximately 30 weeks allocated for project development. This duration must be balanced alongside the academic commitments and job search activities of three student team members. Moreover, due to data sensitivity concerns, access to normal packet captures of medium to large corporations is restricted, which limits us to public datasets for development.

Hardware limitations constrain our model training, as we are restricted to personal computers and WAVE, limiting our ability to train on larger datasets. Additionally, issues with purchasing and the 16-week estimated shipping time of the Bluefield 3 DPU prevent us from testing the model on our own SmartNIC. A significant constraint is the necessity to preprocess data to ensure compatibility with machine learning libraries and frameworks, primarily those available in Python, which we have chosen as our development environment.

## 7.2 Standards

Code for training and testing the ransomware detection model is written and interpreted in Python 3.12.0 [19] because of personal familiarity and Python's popularity in machine learning. Our code generally follows the PEP (Python Enhancement Proposals) 8 style guide [20], which provides conventions to keep our code readable and consistent. In addition, we use standard machine learning libraries like Scikit-learn [21] and XGBoost [22] which implement APIs in multiple programming languages so that trained models can be easily ported to other systems

Our project uses Conda [23], a command line tool to manage Python dependencies and environments. We use Jupyter Notebook (.ipynb) [24] files that combine rich text with code and execution output, allowing us to record results and document model training and evaluation. Standard metrics like accuracy and f1-score were imported from Sci-kit learn to ensure fair evaluation of models, and the recommended 80% training 20% testing split was used to partition the network traffic dataset.

# Chapter 8: Societal Issues

## 8.1 Ethical

In the development of the ransomware detection model, several ethical issues arise, predominantly concerning data privacy and security. One significant ethical concern is the potential violation of personal and corporate privacy. During the process of training the model, normal network traffic is recorded, which inherently contains sensitive information. Although the analysis is restricted to network headers within PCAP files, the mere act of collecting this data can still pose privacy risks. These ethical dilemmas highlight the need for careful consideration and management of data to ensure that privacy and security are not compromised during the development and deployment of the detection model.

## 8.2 Economic

The primary economic consideration for this project is the cost of the SmartNIC. Apart from this significant expense, the remainder of the project relies entirely on software, incurring no additional costs. During our research, we explored alternative options for the SmartNIC, such as FPGAs and ASICs. However, because we wanted to focus on the software side of development, SoC emerged as the most suitable option. While our models are trained with the Bluefield DPU in mind, ProtectNIC hopes to reduce the cost of businesses like cloud providers by offloading security tasks onto a range of existing in-network devices. This could also help reduce the cost of cloud services to end users and the general public.

## 8.3 Compassion

Compassion is central to our project, as we are acutely aware of the suffering caused by ransomware attacks, particularly in healthcare settings. In 2023 alone, 46 hospital systems were targeted, affecting 141 hospitals and disrupting access to vital IT systems and patient records [25]. This disruption forces emergency departments to redirect patients to other facilities, straining regional healthcare resources and negatively impacting the treatment of time-sensitive conditions such as acute strokes. Tragically, it is estimated that ransomware attacks led to the deaths of 42 to 67 Medicare patients between 2016 and 2021 [25]. By developing a robust ransomware detection model, we aim to alleviate this suffering by enhancing the security of healthcare systems, ensuring continuous access to critical patient data, and ultimately safeguarding human lives. This project is driven by a desire to relieve the suffering caused by cyberattacks, reflecting our commitment to compassion in engineering.

## 8.4 Sustainability

A major sustainability issue with malware detection models is whether they can keep up with evolving cyberattacks. In order for machine learning to keep up with the latest ransomware, models need to be updated with the newest data. Recording ransomware traffic has to be an ongoing effort to prevent models from becoming obsolete when faced with new ransomware threats.

## 8.5 Lifelong Learning

Machine learning development is often a process of trial and error. However, we can guide our prototyping with the help of the insights of more experienced engineers. Building the final

models for ProtectNIC required inspiration from previous prototypes, advice from our faculty advisor, and knowledge of previous research. We needed to explore XGBoost's API ourselves in order to build a classifier that could perform incremental training. This project has helped us explore outside of what coursework has taught us, and has been a reminder that there is often no "correct" answer when creating real world solutions. We hope this serves as a first step of converting theory into practice, from students to engineers.

## Chapter 9: Conclusions

In this paper, we presented a novel approach to ransomware detection that may help offload security tasks to in-network devices like the SmartNIC. The models presented use the TCP payload to make inferences at high accuracies, and show that payload analysis of packet flows can be a promising approach to detecting ransomware. While we saw that random forest outperforms XGBoost in terms of metrics, XGBoost provides a resource-efficient solution to ransomware detection and a convenient way to improve an existing model with contemporary data. Our results provide insights on how machine learning models might be used for lower latency network security and we demonstrate how small, lightweight models can be trained for packet flow analysis.

Both our models achieve our goals in terms of f1-score and model size, but we have not performed performance testing on their intended hardware due to issues with purchasing the Bluefield DPU. In addition, while we evaluate the model against the testing dataset, the model needs to be generalized to work against unseen threats like zero-day attacks. Finally, a big disadvantage of the random forest model is that it must be retrained for every update, which is a lengthy process when working with large datasets. Even the XGBoost model, which supports incremental training, must be adjusted so it can scale more effectively without significantly increasing the number of estimators.

Much work needs to be done to test and improve model performance on zero-day attacks and outliers like Spora. Possible solutions include increasing training data from the Ransomware PCAP repository to fix class imbalance and adjusting model hyperparameters to improve model scaling and accuracy. We also hope to build a working system on a SmartNIC so network latency can be measured in real-world situations.

## Chapter 10: Acknowledgements

We would like to express our heartfelt gratitude to Professor Choi for his guidance and help throughout our project. As a team, we deeply appreciate his dedication and the valuable time he invested to provide advice and direction throughout the project.

We would also like to extend our deepest gratitude to Professor Daniel Morato Oses of the Public University of Navarre for his invaluable assistance in providing us access to the dataset that was essential for our project.

## References

- [1] C. Griffiths, “The latest ransomware statistics (updated april 2024): Aag it support,” Jan 2024. [Online]. Available: <https://aag-it.com/the-latest-ransomware-statistics/>
- [2] B. Davie, “SmartNICs, IPU, DPUs de-hyped: Why and how cloud giants are offloading work from server cpus,” Nov 2024. [Online]. Available: [https://www.theregister.com/AMP/2021/11/24/infrastructure\\_processing\\_units/](https://www.theregister.com/AMP/2021/11/24/infrastructure_processing_units/).
- [3] D. Firestone et al., ‘Azure Accelerated Networking: SmartNICs in the Public Cloud’, in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 51–66.
- [4] C. Zheng et al., ‘Automating In-Network Machine Learning’, *arXiv [cs.NI]*. 2022.
- [5] C. Zheng, M. Zang, X. Hong, L. Perreault, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, “Planter: Rapid Prototyping of In-Network Machine Learning Inference,” *ACM SIGCOMM Computer Communication Review*, 2024.
- [6] B. M. Xavier, R. Silva Guimaraes, G. Comarela, and M. Martinello, “Map4: A pragmatic framework for in-network machine learning traffic classification,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4176–4188, 2022.
- [7] J.-H. Lee and K. Singh, ‘SwitchTree: In-network Computing and Traffic Analyses with Random Forests’, *Neural Computing and Applications*, 2020.
- [8] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, ‘Programmable Switches for in-Networking Classification’, in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.
- [9] E. Berrueta, D. Morato, E. Magaña, and M. Izal, ‘Crypto-ransomware detection using machine learning models in file-sharing network scenarios with encrypted traffic’, *Expert Systems with Applications*, vol. 209, p. 118299, 2022.
- [10] U. Zahoora, A. Khan, M. Rajarajan, S. H. Khan, M. Asam, and T. Jamal, “Ransomware detection using deep learning based unsupervised feature extraction and a cost sensitive pareto ensemble classifier,” *Scientific Reports*, vol. 12, no. 1, September 2022. [Online]. Available: <https://openaccess.city.ac.uk/id/eprint/28937/>
- [11] E. Berrueta, D. Morato, E. Magaña, and M. Izal, “Open repository for the evaluation of ransomware detection tools,” 2020. [Online]. Available: <https://dx.doi.org/10.21227/qnyn-q136>
- [12] D. Morató, E. Berrueta, E. Magaña, and M. Izal, “A Chronological Evolution Model for Crypto-Ransomware Detection Based on Encrypted File-Sharing Traffic,” Jan. 2022, doi: 10.2139/ssrn.4074557.



- [13] N. Moustafa and J. Slay, “Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set),” in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.
- [14] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, “Malware traffic classification using convolutional neural network for representation learning,” in *2017 International conference on information networking (ICOIN)*, pp. 712–717, IEEE, 2017.
- [15] M. Almousa, J. Osawere, and M. Anwar, ‘Identification of Ransomware families by Analyzing Network Traffic Using Machine Learning Techniques’, in *2021 Third International Conference on Transdisciplinary AI (TransAI)*, 2021, pp. 19–24.
- [16] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, ‘Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection’, *arXiv [cs.CR]*. 2016.
- [17] G. Fox and R. V. Boppana, ‘Detection of Malicious Network Flows with Low Preprocessing Overhead’, *Network*, vol. 2, no. 4, pp. 628–642, 2022.
- [18] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, ‘Malware traffic classification using convolutional neural network for representation learning’, in *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 712–717.
- [19] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.
- [20] “PEP 8 – style guide for python code,” Python Enhancement Proposals (PEPs). [Online]. Available: <https://peps.python.org/pep-0008/#code-lay-out>.
- [21] F. Pedregosa et al., ‘Scikit-learn: Machine Learning in Python’, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [22] T. Chen and C. Guestrin, ‘XGBoost: A Scalable Tree Boosting System’, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA, 2016, pp. 785–794.
- [23] ‘Anaconda Software Distribution’, *Anaconda Documentation*. Anaconda Inc., 2020.
- [24] T. Kluyver et al., ‘Jupyter Notebooks -- a publishing format for reproducible computational workflows’, in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016, pp. 87–90.
- [25] S. Alder, “At Least 141 Were Hospitals Directly Affected by Ransomware Attacks in 2023,” *HIPAA Journal*, Jan. 04, 2024.  
<https://www.hipaajournal.com/2023-healthcare-ransomware-attacks>.

# Appendices

## Appendix A: Final Hyperparameters for Resulting Models

### **XGBoost Hyperparameters**

```
booster='gbtree'  
device=cpu  
validate_parameters=True  
nthread=None  
disable_default_eval_metric=False  
eta=0.3  
gamma=0  
max_depth=6  
min_child_weight=1  
max_delta_step=0  
subsample=1  
sampling_method='uniform'  
colsample_bytree=1  
colsample_bylevel=1  
colsample_bynode=1  
lambda=1  
alpha=0  
tree_method='auto'  
scale_pos_weight=1  
refresh_leaf=1  
process_type='default'  
grow_policy='depthwise'  
max_leaves=0  
max_bin=256  
num_parallel_tree=1  
multi_strategy=one_output_per_tree  
max_cached_hist_node=65536  
objective=binary:hinge  
seed=0  
seed_per_iteration=False
```

### **Scikit-learn Random Forest Hyperparameters**

```
n_estimators=100  
criterion="gini"  
max_depth=None  
min_samples_split=2  
min_samples_leaf=1  
min_weight_fraction_leaf=0.0  
max_features="sqrt"  
max_leaf_nodes=None  
min_impurity_decrease=0.0  
bootstrap=True  
oob_score=False  
n_jobs=None
```

```
random_state=0  
warm_start=False  
class_weight=None  
ccp_alpha=0.0  
max_samples=None  
monotonic_cst=None
```

## Appendix B: Table of .pcap files used and their Sources

Filename	Source
cerber_04102016.pcap cerber_03102016.pcap cerber_05102016.pcap cerber_06022017.pcap cerber_10082016.pcap CryLock_24012021.pcap CryptoFortress_12032015.pcap CryptoShield_31012017.pcap Hive_06082021.pcap maktub_12042018.pcap MRCR_15012017.pcap netwalker_21012021.pcap RansomX_28062020.pcap Scarab_18102019.pcap Spora_17052017.pcap	Ransomware PCAP Repository [11]
Gmail.pcap BitTorrent.pcap WorldOfWarcraft.pcap	USTC-TFC2016 [14]
client_normal_1_210110.pcap client_normal_2_210111.pcap client_normal_3_210111.pcap normal_IoT_3.pcap	UNSW-NB15 [13]