



Department of Computer Science

BSc (Hons) Computer Science

Academic Year 2019 - 2020

Storage Covert Channels Over Network Traffic

Anish Limbu

1618834

A report submitted in partial fulfilment of the requirements for the degree of
Bachelor of Science

Brunel University
Department of Computer Science
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

Hosts on the Internet use Internet protocols to communicate with each other. By exploiting the design weaknesses of Internet protocols, one could hide data in the header fields of different Internet protocols to communicate securely, launch cyberattacks, exfiltrate data, etc., in a stealth manner.

A lot of research has been done to detect covert storage channels using Artificial Intelligence. This is very difficult due to the enormous number of Internet protocols, the number of header fields that can be used to hide data, differentiating between normal and covert packets, etc. Since it would be impossible to detect all types of covert storage channels in the time allocated of my project, this project will focus on detecting the techniques used by NCovert, and Covert_tcp tools using a small number of supervised machine learning algorithms on labelled data. Then the trained models will be evaluated to identify which algorithms are better in detecting the techniques used by the two covert storage channel tools.

Acknowledgements

I certify that the work presented in the dissertation is my own unless referenced.

Signature: Anish

Date: 27/03/2020

Total Words: 10,593
(including references, titles, captions for figures, etc.)

Table of Contents

Abstract.....	i
Acknowledgements	ii
Table of Contents.....	iii
List of Tables	vi
List of Figures.....	vii
1 Chapter 1: Introduction	9
1.1 Introduction	9
1.2 Aims and Objectives.....	9
1.3 Research Approach	10
1.4 Dissertation Outline	10
2 Chapter 2: Theory and Motivation.....	11
2.1 Covert Channel Types	11
2.1.1 Covert Storage Channels.....	11
2.1.2 Covert Timing Channels	11
2.2 Potential Users of Covert Channels	11
2.2.1 Scenario 1: An Insider Threat.....	12
2.2.2 Scenario 2: Controlling Compromised Systems	12
2.3 Summary	12
3 Chapter 3: Background	12
3.1 The TCP/IP Model	13
3.1.1 Transport Layer and TCP	13
3.1.2 Internet Layer and IP.....	13
3.2 Covert Storage Channels in the TCP/IP Protocol Suite	14
3.2.1 IP Identification Field	14
3.2.2 TCP Sequence and Acknowledgement Fields.....	14
3.3 Detecting Covert Storage Channels Using AI	15
3.4 Summary	16
4 Chapter 4: Methodology.....	16
4.1 Research Aims.....	16
4.2 Research Execution	17
4.2.1 Requirements and Data Collection	17
4.2.2 Data Preparation	18

4.2.2.1	Extracting Features from PCAP Using tshark and Data Cleaning	18
4.2.3	Feature Selection and Engineering	18
4.2.4	Feature Scaling.....	19
4.2.5	Model Training and Testing Sets	19
4.2.6	Model Evaluation	19
4.2.6.1	Confusion Matrix.....	19
4.2.6.2	Classification Report: Precision, Recall, and F1 Scores	20
4.3	Summary	20
5	Design	20
5.1	Introduction	20
5.2	Requirements and Data Collection	21
5.2.1	Creating an Experimental Lab Environment	21
5.2.2	Data Generation and Collection.....	22
5.3	Data Preparation.....	23
5.3.1	Converting Wireshark PCAP to CSV.....	23
5.3.2	Cleaning and Labelling Data	24
5.3.2.1	Removing Blank Observations	24
5.3.2.2	Combining Covert and Normal Packets and Labelling Them	24
5.4	Feature Selection and Engineering	25
5.4.1	Feature Engineering	25
5.4.2	Selecting Features: Filter and Wrapper Methods	25
5.5	Designing the Passive Warden System	26
5.5.1	Functional Requirements	26
5.5.2	Passive Warden Design	27
5.6	Summary	27
6	Implementation	28
6.1	Pre-processing Stages	28
6.1.1	Reading in Excel File	28
6.1.2	Dropping Features.....	28
6.1.3	Initialising Independent and Dependent Variables.....	29
6.1.4	Splitting Data into Training and Testing Sets	30
6.1.5	Feature Scaling.....	30
6.2	Initialising Logistic Regression Classifier	30
6.3	Initialising Naïve Bayes Classifier	31

6.4	Initialising SVM Gaussian Kernel Classifier.....	31
6.5	Initialising Voting Ensemble Learning Classifier	32
6.6	Implementing the Passive Warden System	32
6.6.1	Initialising the Passive Warden	33
6.6.2	Loading the Trained Models	33
6.6.3	Example: Initialising the IP ID Trained Model	34
6.6.4	Detection Example: IP ID Covert Storage Channels	34
6.7	Summary	35
7	Evaluation	36
7.1	Evaluating Covert_tcp IP ID Detection Models	36
7.2	Evaluating Covert_tcp TCP Sequence Detection Models	37
7.3	Evaluating Covert_tcp Bounce Acknowledgement Models	38
7.4	Evaluating NCovert TCP Sequence Detection Models	39
7.5	Evaluating NCovert Bounce Acknowledgement Detection Models	40
8	Conclusions	40
8.1	Future Work.....	42
8.2	Personal Reflection	42
8.2.1	Reflection on Project.....	42
8.2.2	Personal Reflection	42
	References	43

List of Tables

Table 1. Datasets Required for Training and Testing Models.	17
Table 2. Feature Selection and Engineering.	19
Table 3. Confusion Matrix for Covert Channel Detection.	20
Table 4. Collected Data.	23
Table 5. Commands Entered to Generate Covert Packets.....	23
Table 6. Commands Entered to Extract Fields to Build Data Sets.....	24
Table 7. Best Performing Detection Models.	41

List of Figures

Figure 1. The TCP/IP Model.	13
Figure 2. A TCP Segment Header.	13
Figure 3. An IPv4 Datagram Header.....	14
Figure 4. Hiding 't' in ID Field of Datagram Using Rowland's Covert_tcp.	14
Figure 5. Hiding 'Hi:)' Inside Sequence Number Field Using NCovert.	15
Figure 6. Bounce Acknowledgement Method Used by NCovert and Covert_tcp to Hide Data.....	15
Figure 7. Data Mining Techniques.	16
Figure 8. Research Execution Methodology.	17
Figure 9. A Single Data Collection in Lab Environment.	18
Figure 10. Classification Report Formulas.....	20
Figure 11. Project Execution Stages.....	21
Figure 12. Created Lab Environment.	21
Figure 13. FreeBSD Connecting the Two NAT Networks.	22
Figure 14. Checking Host Connectivity Using Ping.....	22
Figure 15. Before and After Removing Blank Rows in a Data Set.	24
Figure 16. Labelling the Packets.	25
Figure 17. IP Checksum Feature Engineering.....	25
Figure 18. Filter Method Using Pearson Correlation.	26
Figure 19. Wrapper Method Using Recursive Feature Elimination.	26
Figure 20. Passive Warden System Activity Diagram.....	27
Figure 21. Reading in Excel File Using Pandas.	28
Figure 22. Dropping a Feature.	28
Figure 23. After Dropping 'tcp.checksum' Feature.....	29
Figure 24. Setting Independent and Dependent Variables.....	29
Figure 25. DataFrames of X and y variables.....	29
Figure 26. Splitting Data into Training and Testing Sets.	30
Figure 27. Scaling Features.	30
Figure 28. Logistic Regression Classifier Initialisation.....	31
Figure 29. Naïve Bayes Classifier Initialisation.....	31
Figure 30. SVM Gaussian Kernel Classifier Initialisation.	32
Figure 31. Voting Classifier Initialisation.....	32
Figure 32. Setting the Parameters of the Passive Warden.	33
Figure 33. Loading Trained Machine Learning Models.	33

Figure 34. Initialising the SVM IP ID Model.....	34
Figure 35. Detecting IP ID CSCs in a Wireshark Capture File.....	34
Figure 36. Detecting IP ID CSCs in Real Time.	34
Figure 37. Performance of Covert_tcp IP ID Detection Models.....	36
Figure 38. Prediction Results of Trained Covert_tcp IP ID Detection Models.....	36
Figure 39. Performance of Covert_tcp TCP Sequence Detection Models.	37
Figure 40. Prediction Results of Trained Covert_tcp TCP Sequence Detection Models.	37
Figure 41. Performance of Covert_tcp TCP ACK Bounce Detection Models.....	38
Figure 42. Prediction Results of Trained Covert_tcp TCP ACK Detection Models.	38
Figure 43. Performance of NCovert TCP Sequence Detection Models.....	39
Figure 44. Prediction Results of Trained NCovert TCP Sequence Detection Models.....	39
Figure 45. Performance of NCovert ACK Bounce Detection Models.	40
Figure 46. Prediction Results of Trained NCovert ACK Bounce Detection Models.....	40

1 Chapter 1: Introduction

This chapter highlights the purpose, project approach, the aims and objectives, and the scope of the project.

1.1 Introduction

The Internet runs on Internet protocols. These protocols define how hosts should communicate with each other by defining the structure of the messages, and rules for exchanging messages between hosts. Current Internet design focuses on the effectiveness of moving packets from source to the destination; malicious users can take advantage of this to exploit the weaknesses of existing Internet protocols to create communications channels which can be used to transfer data such as files. Due to the enormous number of methods that can be used to hide data in packet headers, this project will only look into detecting the techniques used by Covert_tcp [1], and NCovert [2] tools. Although these tools are old, it still uses techniques that exploit the current Internet design to create covert storage channels; therefore, the current Internet is still open to this type of attacks.

Machine learning algorithms will be used to detect the covert channel techniques used by the two tools in this project. By using a few supervised learning classification algorithms to detect covert storage channels, this project can help to understand their performance, which algorithms are suitable in detecting the techniques used by the two tools, etc. After the models are trained, a passive warden system will be built that can detect covert storage channels in real-time and in Wireshark capture files.

1.2 Aims and Objectives

The main aim of this project is to focus on detecting the covert storage channel techniques used by Covert_tcp [1], and NCovert [2] using a small number of supervised learning classification algorithms; therefore, their performance will be evaluated in this project to identify which algorithms are better in detecting covert storage channels. In order to meet the aims of this project, the following objectives listed below must be completed:

1. Start literature review. Different areas of covert channels must be reviewed, such as Internet protocols, how these techniques can be detected using machine learning algorithms, etc.
2. Identify an appropriate methodology which can be used to design and tackle the aims and objectives of this project.
3. Prepare requirements, such as which machine learning algorithms to use, from the research found during literature review.

4. Generate and collect covert data using the two tools. Normal Internet traffic must also be collected. Then start implementing the models that can detect the covert storage channels. Also, develop the passive warden system.
5. Evaluate and justify whether or not if the project aims and objectives were met. The performance of the models will be evaluated during this stage to identify which algorithms are better in detecting covert storage channels.

1.3 Research Approach

This project covers topics such as computer networking, algorithms, and cybersecurity. These areas will be researched to gain knowledge on the common techniques which can be used in this project to detect the methods used by the two covert storage channel tools. Then an appropriate methodology will be chosen to plan the overall project. The information gathered through background research will be analysed to formulate requirements, and the design will be created. Then the ML models will be implemented using a suitable programming language and APIs. Finally, the project will be evaluated.

1.4 Dissertation Outline

The table below outlines the chapters in this dissertation. Each chapter focuses on specific parts of this project.

Chapter 2: Theory and Motivation	This chapter will describe what covert channels are, and the potential users of covert channels.
Chapter 3: Background	This chapter will contain the background sources that are relevant to the project.
Chapter 4: Methodology	This chapter will discuss the chosen methodology that is appropriate for this project.
Chapter 5: Design	This chapter will discuss the steps required before implementing the different prediction models. The passive warden system will also be designed during this chapter.
Chapter 6: Implementation	This chapter will contain the pre-processing steps required before implementing the models. Then the different machine learning algorithms would be instantiated and trained in order to solve the aims of this project. Also, the passive warden system will be built.
Chapter 7: Testing and Evaluation	Evaluate the performance of the trained models and discuss whether or not the aims and objectives were met.

Chapter 8: Conclusions	Conclude the overall project. Discuss the areas for improvements and future work.
-------------------------------	---

2 Chapter 2: Theory and Motivation

This chapter will introduce the two types of covert channels and its potential users.

2.1 Covert Channel Types

There are two main types of covert channels: storage and timing channels. Both exploit the Internet protocols in order to transfer information in a manner that violates the system's security policy.

2.1.1 Covert Storage Channels

According to NCSC [4], a covert storage channel is known as:

“A covert channel that involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process. Covert storage channels typically involve a finite resource (e.g., sectors on a disk) that is shared by two subjects at different security levels.”

(NCSC, 1985)

To transfer data secretly, a covert storage channel changes storage attributes; for example, changing values of a packet header to hide data. Detecting CSCs are very complicated due to the number of header fields, protocols, etc. NCovert, and Covert_tcp tools use this technique in order to hide data inside the different fields of Internet protocols.

2.1.2 Covert Timing Channels

Covert timing channels are beyond the scope of this project. This technique does not hide data in packets; instead, packets are sent in timed intervals in order to conceal data. Timing channels are less attractive due to synchronisation issues and low bandwidth in comparison to storage channels.

According to NCSC [4], a covert timing channel is known as:

“A covert channel in which one process signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process.”

(NCSC, 1985)

2.2 Potential Users of Covert Channels

Covert channels allow individuals to keep their communications secret. Users such as criminals, insider threats, hackers, spies, etc, might seek to use these channels for various reasons. Two different scenarios will be used in this section to explain the potential users of covert channels.

2.2.1 Scenario 1: An Insider Threat

Suppose Edward is a system-administrator at a company and is dissatisfied about his pay deductions. He seeks revenge by stealing and selling company data to a third party. Edward and the third party come to agreement to use covert channels to exfiltrate data [9] by using the Exfiltration Over Alternative Protocol [14] technique by utilising Protocol Switching Covert Channels – where the data is exfiltrated in different protocols, instead of just one. Before the data is exfiltrated, Edward uses the Data Encrypted [15] technique by encrypting the data to hide the information that is being exfiltrated, thus making the exfiltration unnoticeable and preventing detection. To make the detection even more difficult, Edward uses the Scheduled Transfer [16] technique so that data exfiltration is only performed at certain intervals or at certain times of day to blend data exfiltration traffic with regular traffic.

2.2.2 Scenario 2: Controlling Compromised Systems

Suppose Bob is a computer programming expert and his goal is to build a DDoS network to control the systems he has compromised to do the work for him. He wants to work on the three stages in a DDoS network [12]: Recruitment, Infection/Exploitation, and Use phases. To recruit agent machines, he uses the Spearphishing Attachment [17] technique by sending the malware attached to an email which requires user execution of the attached malware. He knows that direct communication with his agent machines will expose his IP address and his whole DDoS network; therefore, he looks into using an indirect communication method through the use of covert channels and IRC services to hide his identity. This will allow Bob and his agent machines to communicate indirectly without being detected [10]; therefore, he can now use covert channels to issue commands to coordinate the DDoS attacks.

2.3 Summary

This chapter looked into the theory of covert channels and its potential users. Firstly, the two types of covert channels were discussed. Then the potential users of covert channels were discussed using two different scenarios. The next chapter will look into the background of covert storage channels: the protocols and techniques used by the two tools, and how they can be detected using Machine Learning algorithms.

3 Chapter 3: Background

This chapter covers the research and relevant literature relating to this project. The main goal of this chapter is to introduce the literature associated with covert storage channels. It covers topics such as the protocols used by the two tools, techniques used to hide data, how they can be detected, etc.

3.1 The TCP/IP Model

This protocol suite contains a set of protocols that work together to provide network communication services. Hosts on the Internet use the TCP/IP model to communicate with other hosts on the Internet. There are four layers in this model: Application, Transport, Internet, and Network Access. The figure below illustrates the TCP/IP model.

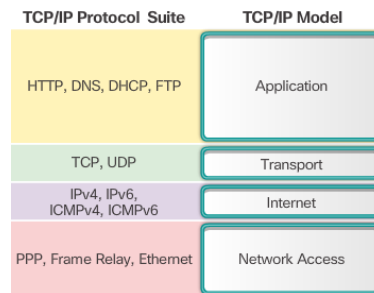


Figure 1. The TCP/IP Model.

3.1.1 Transport Layer and TCP

The transport layer is responsible for providing end-to-end transfer of application layer data. TCP [7] is a connection-oriented and reliable protocol. It recovers from data that is damaged, duplicated, lost, or delivered out of order using a sequence number assigned to each segment. If a segment is lost during transmission, then the sending host will retransmit it again after a timeout occurs; therefore, this protocol provides maximum reliability which is useful for covert channels. Before hosts communicate with each other, they will choose a random initial sequence number to avoid security attacks. Then the sequence number and acknowledgement fields are used to keep track of how much bytes have been sent and received.

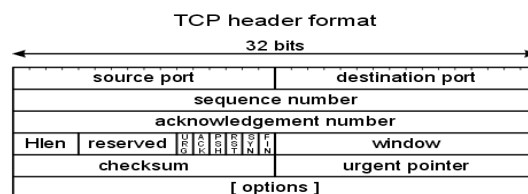


Figure 2. A TCP Segment Header.

3.1.2 Internet Layer and IP

This layer is responsible for routing and forwarding datagrams to provide host-to-host communication using a best-effort delivery service. It is responsible for encapsulating the upper layer segments by adding an IP header. Large datagrams that are bigger than the link's Maximum Transmission Unit (MTU) are fragmented into smaller datagrams. Then this layer is responsible for delivering datagrams from source to destination using the logical addressing scheme. The destination host is responsible for reassembling the fragmented datagrams in correct order using the identification field in each datagram. An example of an IP datagram header is shown below.

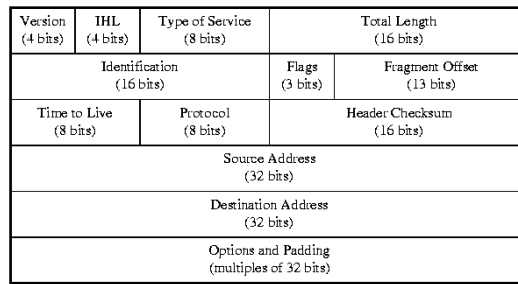


Figure 3. An IPv4 Datagram Header.

3.2 Covert Storage Channels in the TCP/IP Protocol Suite

The two tools this project focuses on use different techniques in order to hide data. These techniques are complicated and difficult to detect.

3.2.1 IP Identification Field

The 16-bit IP ID field is used to give a unique ID to each packet. This is used to identify and reassemble the fragmented packets. Rowland [1] uses this technique to hide data in the identification field of the IP protocol. He does this by replacing the IP identification field with the numerical ASCII value of the character to be encoded. For example, if one wants to transfer the message 'Nice' using Rowland's tool, the characters would be converted into its ASCII representation '78 105 99 101', then each ASCII value would be multiplied by 256; therefore, four packets would be created with each containing a hidden data (e.g. packet₁ would contain 19968 (N), packet₂ would contain 26880 (i), packet₃ would contain 25344 (c), and packet₄ would contain 25856 (e)).

```

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.3.5
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x7400 (29696)
  > Flags: 0x0000
    ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xedbc [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 10.0.3.5
  > Transmission Control Protocol, Src Port: 1234, Dst Port: 200, Seq: 0, Len: 0
    0000 00 00 27 dd 97 0a 08 00 27 7c 8e 8e 08 00 45 00  ....E
    0010 00 28 74 00 00 00 00 06 ed bc 0a 00 02 0f 0a 00  -[t]@.....
    0020 03 05 04 d2 00 c8 ab 12 00 00 00 00 00 50 02  ....P
    0030 02 00 e4 22 00 00
  
```

Figure 4. Hiding 't' in ID Field of Datagram Using Rowland's Covert_tcp.

The identification field of the IP datagram is 29696 in the figure above. When this number is divided by 256, its ASCII representation would be 't' (the hidden data).

3.2.2 TCP Sequence and Acknowledgement Fields

The sequence and acknowledgment fields can store up to 32-bits of information. Covert_tcp uses these fields to hide one byte of data (e.g. the 'P' character) without using any kind of byte-ordering algorithms. This leads to a very high covert rate because a segment is required for each data. NCovert

[2] builds on Covert_tcp [1] and uses a byte-ordering algorithm to hide four bytes (e.g. four characters such as 'Hell') of data inside the sequence and acknowledgement fields.

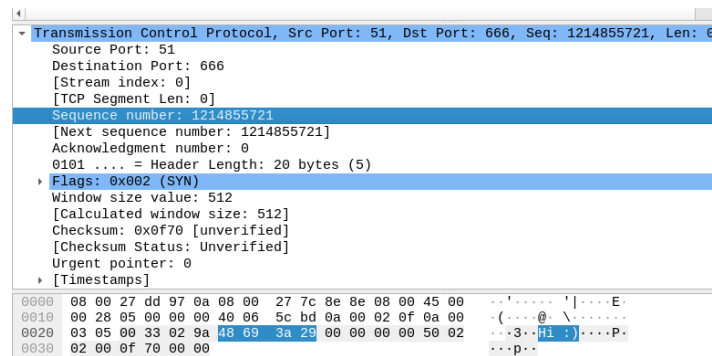


Figure 5. Hiding 'Hi:)' Inside Sequence Number Field Using NCovert.

In the figure above, the sequence number of the TCP segment is 1214855721 and conceals the message 'Hi:)' using a byte-ordering algorithm. The packet details window above shows the hexadecimal values of the four characters. The ACK Bounce technique is used by Covert_tcp [1] and NCovert [2] tools to provide maximum anonymity by using IP spoofing technique. It does this by hiding data inside the acknowledgement field and making sure there is no backward communication between the sender and receiver by bouncing covert packets off a server.

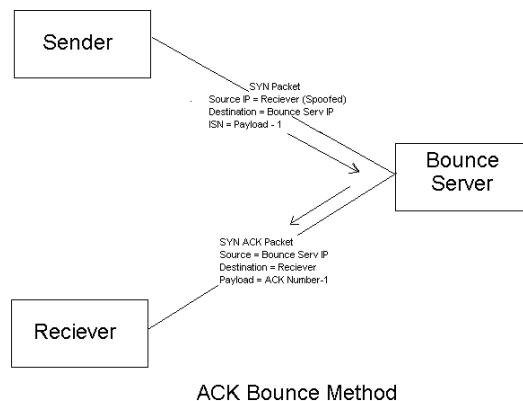


Figure 6. Bounce Acknowledgement Method Used by NCovert and Covert_tcp to Hide Data.

3.3 Detecting Covert Storage Channels Using AI

Over the years, a lot of research has been done [6]; however, the problem is a lot of research focus on using a specific algorithm only (e.g. the SVM algorithm to detect IP and TCP covert channels [18] [19]). The drawback of this is the 'No Free Lunch' [11] theorem. A lot of research also tend to focus on different aims compared to the aims of this project (e.g. detecting different techniques). In the past, unsupervised machine learning algorithms, such as K-Means Clustering, has also been used to detect covert channels [8].

The latest research, at the time of this project, on detecting covert storage channels is [13], where the researchers used multiple supervised learning classification algorithms to detect three types of covert storage channels. They found out SVM performed the best in detecting two out of three techniques. Detecting these two techniques were IP ID and TCP sequence covert storage channels generated using Covert_tcp tool – which are also the aims of this project. However, only accuracy metrics were used for evaluation. Since they did not use KFold cross validation, the models could have underperformed on the different parts of the dataset; for example, SVM performing well on the training and testing sets, but underperforming on the different parts of the data set due to overfitting. Therefore, KFold cross validation will be used in this project to evaluate the trained models.

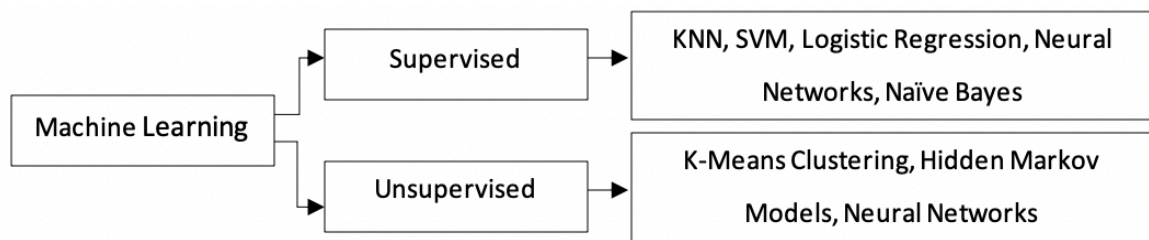


Figure 7. Data Mining Techniques.

3.4 Summary

In this chapter, the different areas of covert channels were covered. These topics include the commonly exploited protocols in the TCP/IP suite, data hiding techniques, how they can be detected, etc. The next chapter looks into the methodology used in the project to tackle the aims and objectives of this project.

4 Chapter 4: Methodology

In this chapter, the methodology used to tackle the project will be discussed along with other topics such as data collection techniques, evaluation methods, etc.

4.1 Research Aims

The main aim of this project is to use a small number of supervised learning classification algorithms to detect the techniques used by the two covert storage channel tools. The outcome of this project will indicate which classification algorithms are suitable in detecting the different techniques.

4.2 Research Execution

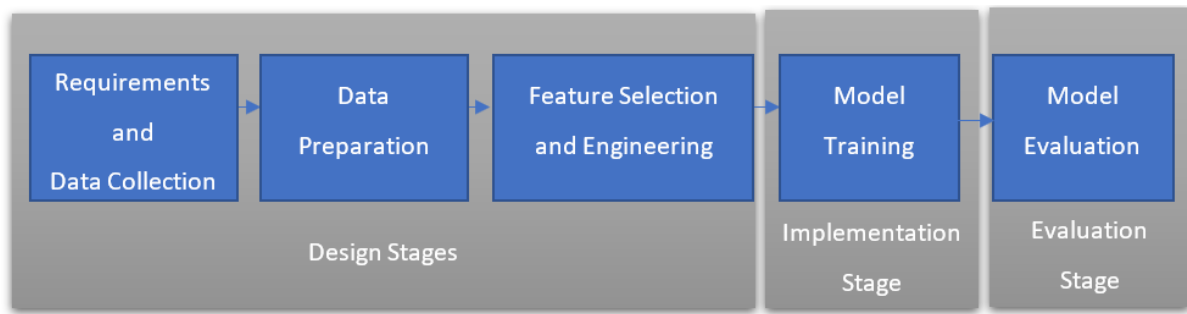


Figure 8. Research Execution Methodology.

The figure above contains the methodology that must be completed in order to build the models successfully. The subsections below elaborate more on the different stages of the methodology.

4.2.1 Requirements and Data Collection

The requirements are the aims of this project. These aims are detecting the techniques used by NCovert, and Covert_tcp tools. The table below specifies the requirements and required data sets in this project. Each row in the table below represents a technique this project is trying to detect; therefore, each technique requires two types of data sets: covert and normal traffic. ML algorithms will be trained on each technique only to identify which is better in detection, rather than training them on a single data set of all combined data sets.

Tools	Hidden Data Encoded in Header Field	Normal Data Sets	Covert Data Sets	Total Packets Count (Normal and Covert)
Covert_tcp	IP ID	Packets from normal communication	Packets from covert_tcp communication	≈100,000
Covert_tcp	TCP sequence	Packets from normal communication	Packets from covert_tcp communication	≈100,000
Covert_tcp	TCP acknowledge	Packets from normal communication	Packets from covert_tcp communication	≈100,000
NCovert	TCP sequence	Packets from normal communication	Packets from NCovert communication	≈100,000
NCovert	TCP acknowledge	Packets from normal communication	Packets from NCovert communication	≈100,000

Table 1. Datasets Required for Training and Testing Models.

A normal data set will consist of 50,000 packets. A covert data set will also consist of 50,000 packets. Therefore, each dataset for each technique will consist of ≈100,000 packets (normal and covert). For example, 50000 IP ID covert packets generated using Covert_tcp would be collected, then 50000 regular packets would be collected, then these two data sets would be labelled and combined together to create a data set for training. An example of this is shown below in figure 9. Wireshark will be used to collect network packets.

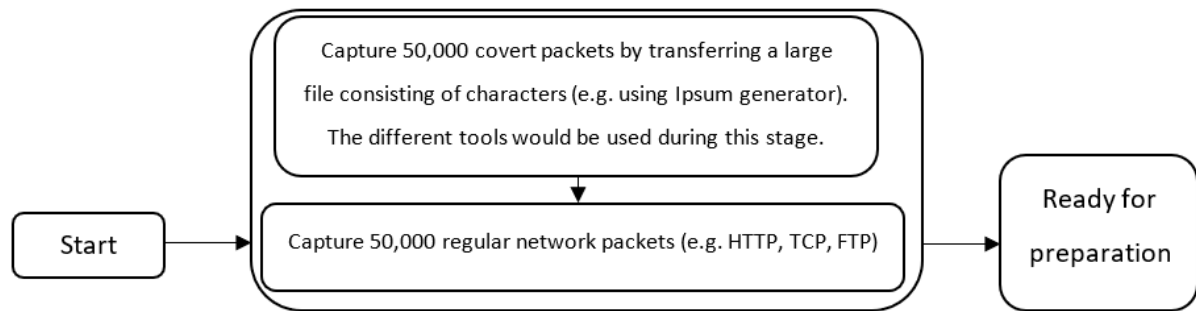


Figure 9. A Single Data Collection in Lab Environment.

4.2.2 Data Preparation

After the packets are captured using Wireshark, they must be prepared. The first step consists of extracting the features from the captured Wireshark files to build the data sets. Then the separate data sets must be cleaned to prevent model prediction errors. After this, the features must be selected and engineered. The subsections below elaborate more on this.

4.2.2.1 Extracting Features from PCAP Using tshark and Data Cleaning

After the packets are captured using Wireshark, they must be converted into a format which can be processed by Python. To achieve this, the Excel format will be used in this project. The tshark library will be used to extract the features from an input Wireshark file; for example, the command below extracts the ID and checksum values of all IP datagrams in a Wireshark capture file:

```
tshark -r input.pcapng -T fields -E separator=, -E header=y, -e ip.id -e ip.checksum >
output.csv
```

If a packet does not contain the IP ID and IP checksum values, then a blank row would be added in the CSV file, thus making it easy to clean data by just having to remove blank rows. Excel will be used to prepare the data sets. This process mainly consists of removing blank rows from unnecessary protocols, labelling the data, and combining the covert and normal packets.

4.2.3 Feature Selection and Engineering

Feature selection is vital in order to make quality and accurate predictions. The data sets contain features (header fields) that will be used to make predictions. If irrelevant features are added in the model then it could make the model inaccurate, also known as Garbage In, Garbage Out. In this project, filter and recursive feature elimination techniques will be used. Feature engineering is the process of using domain knowledge of the data to create features to make machine learning algorithms work. Feature engineering techniques will be used in this project. Each row in the table below is a technique this project is trying to detect; therefore, each technique contains features that require engineering, and the chosen features for predictions.

Tools	Data Hiding Technique	Features Selection	Feature(s) Engineering
Covert_tcp	IP ID	IP ID, IP Don't Fragment flag, IP Checksum	IP Checksum to decimal IP ID to decimal
Covert_tcp	TCP Sequence	TCP Sequence, TCP Next Sequence, TCP Acknowledgement, TCP Checksum, TCP SYN flag	TCP Checksum to decimal
Covert_tcp	TCP Acknowledgement	TCP Sequence, TCP Next Sequence, TCP Acknowledgement, TCP SYN flag, TCP ACK flag, TCP RESET flag, TCP Checksum	TCP Checksum to decimal
NCovert	TCP Sequence	TCP Sequence, TCP Next Sequence, TCP Acknowledgement, TCP Checksum, TCP SYN flag	TCP Checksum to decimal
NCovert	TCP Acknowledgement	TCP Sequence, TCP Next Sequence, TCP Acknowledgement, TCP SYN flag, TCP ACK flag, TCP RESET flag, TCP Checksum	TCP Checksum to decimal

Table 2. Feature Selection and Engineering.

4.2.4 Feature Scaling

After the features are selected and engineered, they must be scaled before training. Scaling the features will help to make sure the values are in a specific range, thus preventing model errors. The StandardScaler from sci-kit learn will be used to scale the features in this project.

4.2.5 Model Training and Testing Sets

Each data set must be split into training and testing sets before it is used by the models. The training set is used to train the model, and the testing set is used to test the overall performance of the trained model on unseen data. Each data set will use the 75/25 ratio; that is, 75% of the data for training, and 25% of the data for evaluating its performance on unseen data. The method 'train_test_split' from sci-kit learn library will be used to split the data sets into training and testing sets.

4.2.6 Model Evaluation

After the models are created and trained, they must be evaluated. Using the accuracy score alone is not the preferred way to evaluate classification models; therefore, the methods provided by sci-kit learn will be used to analyse the quality of predictions being made by the trained models. The subsections below contain the classification evaluation techniques that will be used in this project to evaluate the performance of the trained models.

4.2.6.1 Confusion Matrix

Confusion matrix is used to evaluate the performance of a classification model on a set of test data where the actual labels are known. It does this by comparing the predicted and actual classes of the test data. The confusion matrix is split into groups of positives and negatives to output the number of incorrect and correct predictions made on the test data. This metric is provided by the scikit-learn library; therefore, it will be used in this project to evaluate the performance of the trained models. The table below shows an example of a confusion matrix.

		Actual Result	
Predicted Result		Normal	Covert
	Normal	True Negative (TN)	False Positive (FP)
	Covert	False Negative (FN)	True Positive (TP)

Table 3. Confusion Matrix for Covert Channel Detection.

True Positive (TP): packet was predicted it has a covert channel, and it does.

False Positive (FP): packet was predicted it is normal, but it has a covert channel.

False Negative (FN): packet was predicted it has a covert channel, and it does not.

True Negative (TN): packet was predicted it is normal, and it is normal.

4.2.6.2 Classification Report: Precision, Recall, and F1 Scores

These metrics use the conditions described above in the confusion matrix section to build new scores. The precision score outputs the percentage of predictions that were correct. The recall score outputs the percentage of the positive instances that were correctly predicted. Finally, the F1 score outputs the harmonic mean of precision and recall scores. The higher the score, the better the model. These scores will be displayed for each trained model in this project.

$$recall = \frac{TP}{TP + FN} \quad precision = \frac{TP}{TP + FP}$$

$$F1\ Score = 2 * \frac{precision * recall}{precision + recall}$$

Figure 10. Classification Report Formulas.

4.3 Summary

This chapter looked into the methodology that will be used to execute this project. It discussed how data will be collected, how the performance of the models will be evaluated, etc. The next chapter will discuss the requirements and design stages of this project.

5 Design

5.1 Introduction

The previous chapter looked into the methodology that will be used to tackle the aims and objectives of this project. The figure below contains the overall project stages split into design, implementation, and evaluation stages. In this chapter, information from the previous chapters will be explored further to design the overall project, and this chapter will look into the first three design stages from the figure below to prepare data sets for implementation.

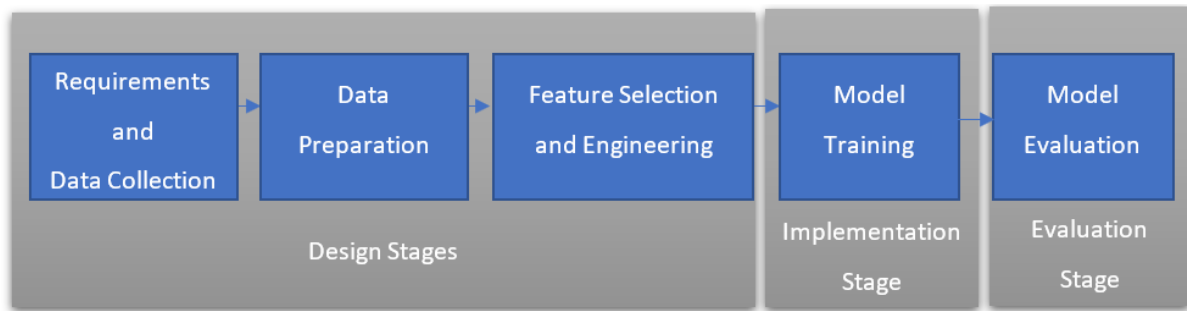


Figure 11. Project Execution Stages.

5.2 Requirements and Data Collection

This section will discuss the overall requirements and data collection stages in more depth. It will mainly focus on the techniques used for data collection.

5.2.1 Creating an Experimental Lab Environment

A lab environment consisting of virtual machines was created using VirtualBox. This lab environment consists of two Kali Linux hosts and one FreeBSD machine. Two NAT networks exist in this lab, and are joined together using the FreeBSD machine, thus allowing communications to take place between the two NAT networks (e.g. enabling Kali Linux #2 to send covert packets to Kali Linux #1 and vice versa). Kali Linux was chosen because it has built-in tools such as Wireshark, and it does not regularly check host connectivity by pinging the Ubuntu server which could introduce noise data. The two tools, Covert_tcp and NCovert, were also installed on Kali Linux #1 and Kali Linux #2. The figure below shows the created lab environment which will be used to generate and collect data.

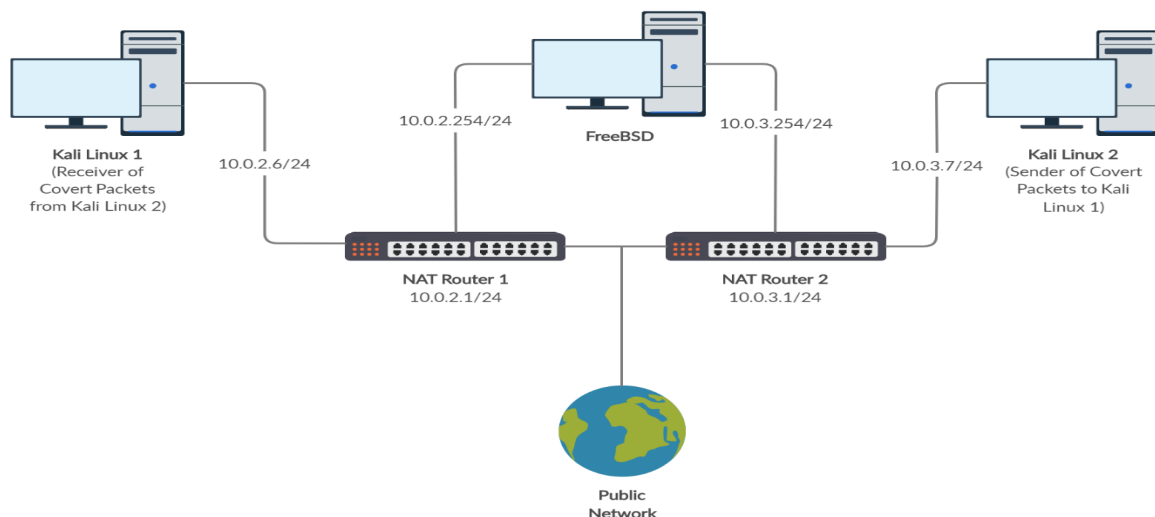


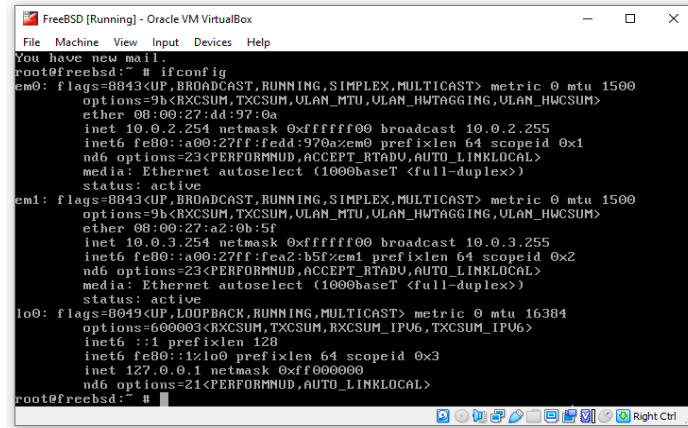
Figure 12. Created Lab Environment.

Kali Linux #2 will generate covert packets and send them to Kali Linux #1. Kali Linux #1 is responsible for receiving the covert packets sent by Kali Linux #2. These packets will be captured using Wireshark.

The following commands were used to setup the gateway addresses to allow communications between the hosts in the two NAT networks:

sudo route add -net 10.0.2.0/24 gw 10.0.3.254 in Kali Linux #2

sudo route add -net 10.0.3.0/24 gw 10.0.2.254 in Kali Linux #1



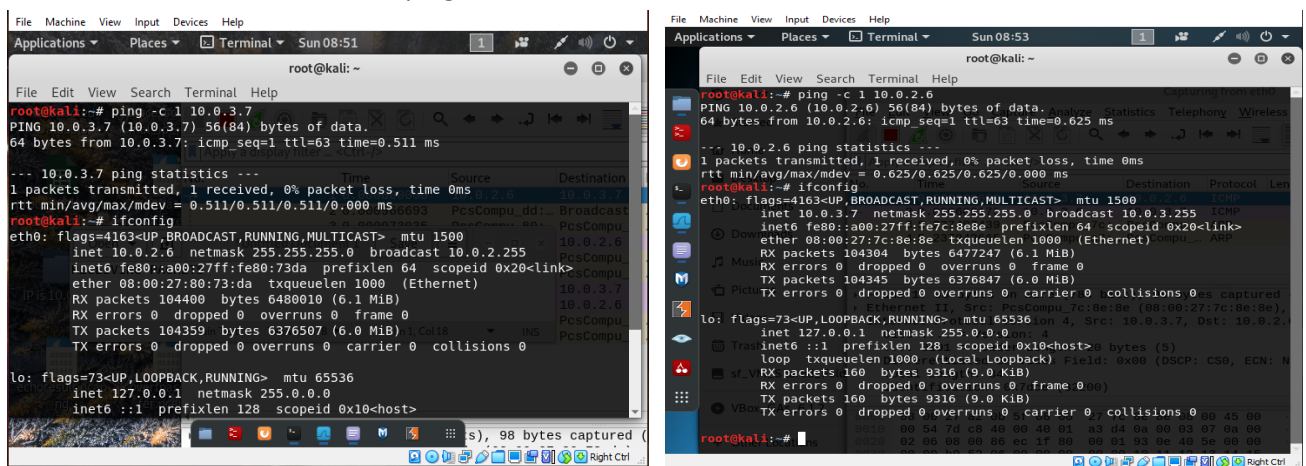
```
FreeBSD [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@freebsd:~ # ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
options=9b<RXCSUM, TXCSUM, ULAN_MTU, ULAN_HWTAGGING, ULAN_HWCSUM>
ether 08:00:27:dd:97:0a
inet 10.0.2.254 netmask 0xfffff00 broadcast 10.0.2.255
inet6 fe80::a00:27ff:fead:970a%em0 prefixlen 64 scopeid 0x1
nd6 options=23<PERFORMNUD, ACCEPT_RTADV, AUTO_LINKLOCAL>
media: Ethernet autoselect (1000baseT <full-duplex>)
status: active

em1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
options=9b<RXCSUM, TXCSUM, ULAN_MTU, ULAN_HWTAGGING, ULAN_HWCSUM>
ether 08:00:27:a2:0b:5f
inet 10.0.3.254 netmask 0xfffff00 broadcast 10.0.3.255
inet6 fe80::a00:27ff:fea2:b5f%em1 prefixlen 64 scopeid 0x2
nd6 options=23<PERFORMNUD, ACCEPT_RTADV, AUTO_LINKLOCAL>
media: Ethernet autoselect (1000baseT <full-duplex>)
status: active

lo0: flags=8049<UP, LOOPBACK, RUNNING, MULTICAST> metric 0 mtu 16384
options=600003<RXCSUM, TXCSUM, RXCSUM_IPV6, TXCSUM_IPV6>
inet ::1 prefixlen 128
inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
inet 127.0.0.1 netmask 0xff000000
nd6 options=21<PERFORMNUD, AUTO_LINKLOCAL>
root@freebsd:~ #
```

Figure 13. FreeBSD Connecting the Two NAT Networks.

In the figure above, the 'ifconfig' command was executed in FreeBSD machine to show the two network interfaces - 'em0' and 'em1' – that are responsible for connecting the two NAT networks; the IP addresses for these interfaces are 10.0.2.254 and 10.0.3.254. Then ping tests were carried out to show that Kali Linux #1 and Kali Linux #2 could communicate with each other. This was successful because both hosts were able to ping each other.



```
root@kali:~# ping -c 1 10.0.3.7
PING 10.0.3.7 (10.0.3.7) 56(84) bytes of data:
64 bytes from 10.0.3.7: icmp_seq=1 ttl=63 time=0.511 ms

--- 10.0.3.7 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.511/0.511/0.511/0.000 ms

root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
ether 08:00:27:80:73:da txqueuelen 1000 (Ethernet)
RX packets 104400 bytes 6480010 (6.1 MiB)
TX packets 104359 bytes 6376507 (6.0 MiB)

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>

root@kali:~# ping -c 1 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data:
64 bytes from 10.0.2.6: icmp_seq=1 ttl=63 time=0.625 ms

--- 10.0.2.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.625/0.625/0.625/0.000 ms

root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
ether 08:00:27:7c:8e:8e txqueuelen 1000 (Ethernet)
RX packets 104304 bytes 6477247 (6.1 MiB)
TX packets 104345 bytes 6376847 (6.0 MiB)

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 160 bytes 9316 (9.0 KiB)
TX packets 160 bytes 9316 (9.0 KiB)
```

Figure 14. Checking Host Connectivity Using Ping.

5.2.2 Data Generation and Collection

A Lorem Ipsum generator was used to generate a text file with over 50,000 ASCII characters. Then the tools Covert_tcp and NCovert were used to transfer this large text file to generate covert packets; Kali Linux #2 was responsible for sending covert packets to Kali Linux #1, and Kali Linux #1 was responsible for receiving the covert packets sent by Kali Linux #2. These covert packets were collected using Wireshark in the lab environment. The normal data sets consist of regular Internet traffic from TCP,

HTTP, FTP, and DNS protocols. Table 4 shows the data sets collected in this project. Table 5 contains the commands used to generate covert packets using the two tools.

Tool	Data Hidden in Header Field	Normal Data Sets	Covert Data Sets	Total Packets Count (Normal and Covert)	Obtained
Covert_tcp	IP ID	Packets from normal communication	Packets from covert_tcp communication	100,398	✓
Covert_tcp	TCP Sequence	Packets from normal communication	Packets from covert_tcp communication	100,398	✓
Covert_tcp	TCP Acknowledgement	Packets from normal communication	Packets from covert_tcp communication	100,398	✓
NCovert	TCP Sequence	Packets from normal communication	Packets from NCovert communication	100,398	✓
NCovert	TCP Acknowledgement	Packets from normal communication	Packets from NCovert communication	100,398	✓

Table 4. Collected Data.

Tool	Hiding Technique	Command(s) Entered to Generate Covert Packets
Covert_tcp	IP ID	Sender: <code>./covert_tcp -dest 10.0.2.6 -source 10.0.3.7 -dest_port 65535 -file file.txt -id</code>
Covert_tcp	TCP Sequence	Sender: <code>./covert_tcp -dest 10.0.2.6 -source 10.0.3.7 -dest_port 65535 -file file.txt -seq</code>
Covert_tcp	TCP Acknowledgement	Sender: <code>./covert_tcp -dest 10.0.2.6 -source 10.0.2.254 -dest_port 65535 -src_port 65535 -file file.txt</code> Receiver: <code>./covert_tcp -source 10.0.2.254 -server -src_port 65535 -dest_port 65535 -file file_captured.txt -ack</code>
Ncovert	TCP Sequence	Sender: <code>./ncovert -d 10.0.2.6 -s 10.0.3.7 -f file.txt -v -p 65535 -l 65535</code>
Ncovert	TCP Acknowledgement	Sender: <code>./ncovert -d 10.0.2.6 -s 10.0.2.254 -f file.txt -v -p 65535 -l 65535</code> Receiver: <code>./ncovert -s 10.0.2.254 -S -f file_captured.txt -a -p 65535 -v</code>

Table 5. Commands Entered to Generate Covert Packets.

5.3 Data Preparation

This section will elaborate more on the data preparation stage. It will focus on the tshark library used to convert Wireshark capture files into CSV files, feature selection and engineering processes, etc.

5.3.1 Converting Wireshark PCAP to CSV

The tshark library was used to extract the fields from the Wireshark capture files to create the data sets. Using this library, it is possible to extract specific fields from protocols in packets. The table below contains the commands used to extract the different fields from Wireshark capture files to build the data sets.

Technique	Command Entered to Extract Features
Covert_tcp IP ID	tshark -r input_wireshark_file.pcapng -T fields -E separator=, -E header=y, -e ip.id -e ip.flags.df -e ip.checksum -e ip.ttl > output_wireshark_file.csv
Covert_tcp TCP SEQ	tshark -r input_wireshark_file.pcapng -T fields -E separator=, -E header=y, -e tcp.seq -e tcp.nxtseq -e tcp.ack -e tcp.checksum -e tcp.flags.syn > output_wireshark_file.csv
Covert_tcp Bounce ACK	tshark -r input_wireshark_file.pcapng -T fields -E separator=, -E header=y, -e tcp.srcport -e tcp.dstport -e tcp.seq -e tcp.nxtseq -e tcp.ack -e tcp.flags.syn -e tcp.flags.ack -e tcp.flags.reset -e tcp.checksum > output_wireshark_file.csv
NCovert TCP SEQ	tshark -r input_wireshark_file.pcapng -T fields -E separator=, -E header=y, -e tcp.seq -e tcp.nxtseq -e tcp.ack -e tcp.checksum -e tcp.flags.syn > output_wireshark_file.csv
NCovert Bounce ACK	tshark -r input_wireshark_file.pcapng -T fields -E separator=, -E header=y, -e tcp.seq -e tcp.nxtseq -e tcp.ack -e tcp.flags.syn -e tcp.flags.ack -e tcp.flags.reset -e tcp.checksum > output_wireshark_file.csv

Table 6. Commands Entered to Extract Fields to Build Data Sets.

5.3.2 Cleaning and Labelling Data

After the CSV files were generated, it was time to prepare the data sets. This process mainly consisted of removing blank rows, combining the packets from regular and covert data sets to create a single data set for each technique and labelling them, etc.

5.3.2.1 Removing Blank Observations

Excel was used to clean the CSV data sets. As mentioned in the previous section, tshark will add a blank row if the specific fields are not found in a packet. Not removing blank rows can affect the machine learning algorithms. These blank rows were removed by using the following procedures in Excel: *Find & Select -> Go To Special... -> Select Blanks -> OK -> Delete -> Delete Sheet Rows*. This was done for all the data sets, normal and covert packets, in this project. The two figures below show the before and after removing the blank rows in the same data set.

Figure 15. Before and After Removing Blank Rows in a Data Set.

5.3.2.2 Combining Covert and Normal Packets and Labelling Them

Since this project focuses on using supervised learning algorithms, two classes exist: 1 (covert) and 0 (normal). The observations in the data sets are labelled according to this. Each packet in a data set will store its class type in a new column called 'class'; for example, the column 'K' in the figure below. This

process was completed for each data set after combining the normal and covert packets together to create a final data set that will be used for training.

A	B	C	D	E	F	G	H	I	J	K
65535	65535	0	0	544532655	0	1	1	0.000000e+0	17703	1
65535	65535	0	0	1970255278	0	1	1	1.000000e+0	45302	1
65535	65535	0	0	1942019544	0	1	1	1.000000e+0	46666	1
65535	65535	0	0	1668841859	0	1	1	1.000000e+0	45206	1
65535	65535	0	0	918324851	0	1	1	1.000000e+0	41864	1
65535	65535	0	0	170147681	0	1	1	1.000000e+0	46454	1
65535	65535	0	0	1857294449	0	1	1	1.000000e+0	41876	1
65535	65535	0	0	1656641278	0	1	1	1.000000e+0	43337	1
65535	65535	0	0	1657076282	0	1	1	1.000000e+0	45555	1
65535	65535	0	0	1994732533	0	1	1	1.000000e+0	48444	1
65535	65535	0	0	163505191	0	1	1	1.000000e+0	52259	1
65535	65535	0	0	1628810251	0	1	1	1.000000e+0	54329	1
65535	65535	0	0	1668623955	0	1	1	1.000000e+0	1225	1
65535	65535	0	0	1443203858	0	1	1	1.000000e+0	52737	1
65535	65535	0	0	1970478238	0	1	1	1.000000e+0	232	1
65535	65535	0	0	1983118031	0	1	1	1.000000e+0	46164	1
65535	65535	0	0	1903050795	0	1	1	1.000000e+0	48595	1
65535	65535	0	0	1903050909	0	1	1	1.000000e+0	45206	1
65535	65535	0	0	1789897973	0	1	1	1.000000e+0	3332	1
65535	65535	0	0	1797932659	0	1	1	1.000000e+0	43226	1
65535	65535	0	0	1988353035	0	1	1	1.000000e+0	46160	1
65535	65535	0	0	1903050962	0	1	1	1.000000e+0	52953	1
65535	65535	0	0	1643517788	0	1	1	1.000000e+0	2562	1
65535	65535	0	0	1635053357	0	1	1	1.000000e+0	41859	1
65535	65535	0	0	1970088001	0	1	1	1.000000e+0	2338	1
65535	65535	0	0	1970042474	0	1	1	1.000000e+0	41864	1
65535	65535	0	0	1651879859	0	1	1	1.000000e+0	46664	1
65535	65535	0	0	1702377970	0	1	1	1.000000e+0	51850	1
65535	65535	0	0	1637739462	0	1	1	1.000000e+0	2338	1
55192	80	2244834118	2244834185	0	1	0	0	0.000000e+0	20964	0
80	55192	259325	259325	2244834200	1	1	0	0.000000e+0	9077	0
55192	80	2244834200	2244834200	259326	0	1	0	0.000000e+0	20964	0
55192	80	2244834200	2244834468	259326	0	1	0	0.000000e+0	20432	0
80	55192	259326	259326	2244834468	0	1	0	0.000000e+0	44462	0
36942	443	2244834468	2244834468	259326	0	1	0	0.000000e+0	20964	0
36942	443	2244834468	2244834468	259326	0	1	0	0.000000e+0	98177	0
36942	443	259326	259326	505906298	1	1	0	0.000000e+0	19464	0
36942	443	505906298	505906298	259326	0	1	0	0.000000e+0	90797	0
36942	443	505906298	505906298	259326	0	1	0	0.000000e+0	51194	0
36942	443	259326	259326	505906298	0	1	0	0.000000e+0	6220	0
36942	443	505906298	505906298	259326	0	1	0	0.000000e+0	90797	0
36942	443	505906298	505906298	259326	0	1	0	0.000000e+0	49105	0
36942	443	505906298	505906298	259326	0	1	0	0.000000e+0	90797	0
36942	443	259326	259326	505906298	0	1	0	0.000000e+0	13685	0
36942	443	505906298	505906298	259326	0	1	0	0.000000e+0	90797	0
45266	80	1784326418	1784326418	259326	0	1	0	0.000000e+0	50833	0

Figure 16. Labelling the Packets.

5.4 Feature Selection and Engineering

This section focuses on selecting and engineering features to improve the performance of the machine learning algorithms. Since the aim of this project is to detect the techniques used by NCovert and Covert_tcp only, they use specific traffic patterns that can be distinguished from normal traffic; for example, sequence field is set to 0 and reset flag is 1 when Covert_tcp hides data inside the acknowledgement field – this pattern is not used by normal traffic. These patterns/features will be selected to build reliable machine learning models that can detect the five techniques used by the two tools.

5.4.1 Feature Engineering

All data sets contain features that must be engineered before the training process. Certain hexadecimal features, such as TCP checksum values and IP ID values, are converted into its decimal format, and stored in a new column. Excel was used to engineer the features specified in table 2. The command `=HEX2DEC(RIGHT(cr,(LEN(cr)-2)))` is used to convert a hexadecimal value into its decimal format; the value 'cr' specifies the location (column and row) of the hexadecimal value. During the training process, the hexadecimal features are dropped, and the decimal features are used instead.

E2						
=HEX2DEC(RIGHT(D2,(LEN(D2)-2)))						
	A	B	C	D	E	F
1	ip.id	ip.id.dec	ip.flags.df	ip.checksum	ip.checksum.dec	ip.ttl
2	0x00004c00	19456	0	0x000030ce	12494	64

Figure 17. IP Checksum Feature Engineering.

5.4.2 Selecting Features: Filter and Wrapper Methods

The filter technique relies on identifying features that are highly correlated with the 'class' feature using Pearson Correlation. The wrapper method relies on identifying the best features by selecting all

the features first, and then removing the irrelevant features continuously – a process also known as Recursive Feature Elimination. By using these two methods, it was possible to select the most important features in each data set to build accurate and reliable models. Algorithms such as Logistic Regression support this wrapper method, but not Naïve Bayes. The table 2 specifies the features that were chosen for each technique.

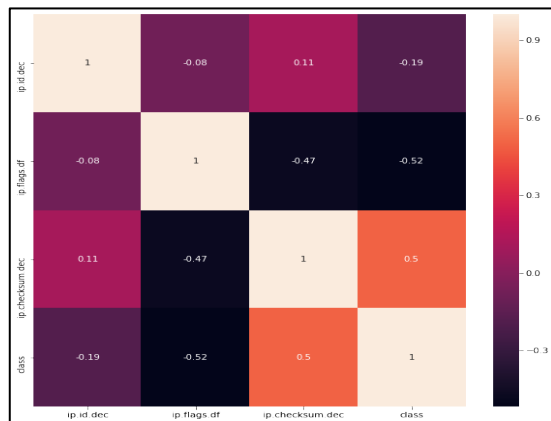


Figure 18. Filter Method Using Pearson Correlation.

The image on the left shows the filter method using the Pearson Correlation to identify the features that are correlated with the 'class' feature. The image on the left specifies the features 'ip.flags.df' and 'ip.checksum.dec' are highly correlated with the 'class' feature.

```
RFE(estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
    fit_intercept=True, intercept_scaling=1,
    l1_ratio=None, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2',
    random_state=None, solver='lbfgs', tol=0.0001,
    verbose=0, warm_start=False),
    n_features_to_select=4, step=1, verbose=0)

print(rfe.support_)
print(rfe.ranking_)

[ True  True  True False False False  True]
[1 1 1 4 3 2 1]

X.head()

tcp.seq tcp.nextseq tcp.ack tcp.flags.syn tcp.flags.ack tcp.flags.reset tcp.checksum.dec
```

Figure 19. Wrapper Method Using Recursive Feature Elimination.

The image on the left shows an RFE in action for the Logistic Regression algorithm with the number of features to keep set to four. 'rfe.support_' and 'rfe.ranking_' properties are used to output the best features to keep.

5.5 Designing the Passive Warden System

This section will describe the passive warden system that will be capable of detecting the different covert storage channels in real-time and in saved Wireshark capture files with the help of the best trained models.

5.5.1 Functional Requirements

The passive warden system has its own functional requirements. The requirements of the Passive Warden system are listed below:

1. The warden should be able to detect the five different covert storage channel techniques, used by Covert_tcp and NCovert using the best trained models, in real-time and in Wireshark capture files.

2. The user should be able to choose which technique to detect, and whether to detect covert storage channels in real-time or in a Wireshark capture file.
3. The warden should be able to locate the hidden data inside the covert packets, decode it, and print it to the screen for the user to see what was transferred/received in a covert manner.

5.5.2 Passive Warden Design

Figure 20 contains the activity diagram that describes the passive warden system. First, the user will select which technique to detect; for example, IP ID or TCP sequence covert storage channels. Then the user will enter the detection method, that is, whether to detect covert packets in real time or in a Wireshark capture file. If the user enters 1, then covert packets are detected in real time using a packet sniffer. If the user enters 2, then covert packets are detected in a Wireshark capture file. Each packet will be passed to a function called 'check_packet' that has the job of identifying whether the packet has a covert storage channel or not. This function/process is responsible for building the features list, predicting whether the packet has a covert channel or not, and outputting a message.

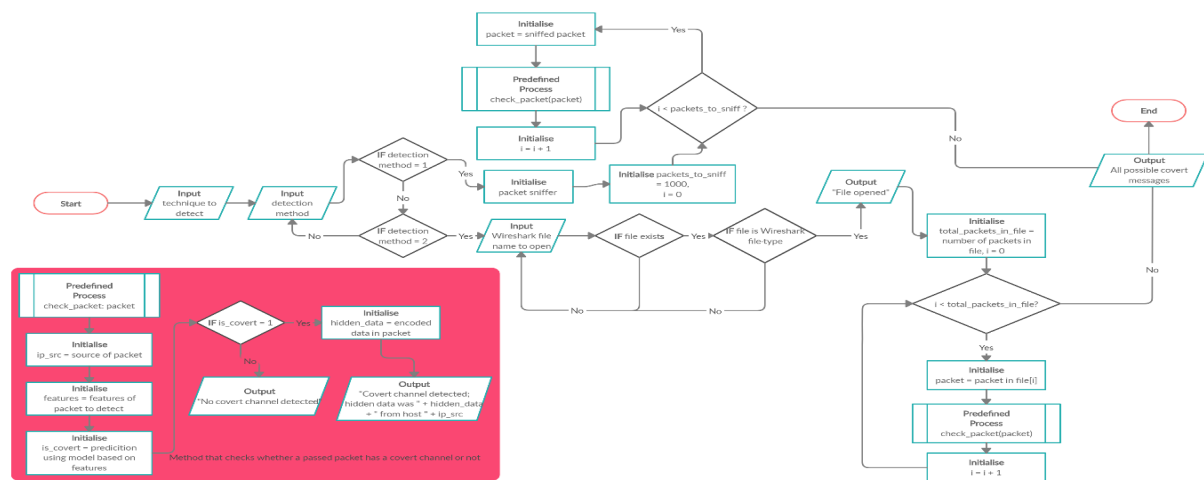


Figure 20. Passive Warden System Activity Diagram.

5.6 Summary

This chapter looked into the design stages of this project. This chapter mainly consisted of designing and creating the lab environment, turning Wireshark data sets into CSV files, selecting and engineering important features for training, preparing the data sets for training. The next chapter will focus on the implementation of the passive warden, and the different machine learning models using the scikit learn library and training them on the data sets created in this chapter.

6 Implementation

This chapter will focus on implementing the passive warden system and creating the machine learning models to detect the covert storage channels. The pre-processing steps are completed before Logistic Regression, Naïve Bayes, Support Vector Machine, and Voting Classifier algorithms are instantiated and trained.

6.1 Pre-processing Stages

The data sets will require pre-processing before being used by the machine learning models. This stage consists of reading in the data, dropping irrelevant features, splitting data into training and testing sets, scaling the variables, etc. These stages are very similar to each other; therefore, examples would be shown to demonstrate the different pre-processing stages using a single data set.

6.1.1 Reading in Excel File

```
# Two classes exist in this dataset: 1 and 0
# 1 = Covert packet
# 0 = Normal packet
dataset = pd.read_excel('Updated Final CovertTcp TCP SEQ Dataset.xlsx')
dataset.head(5)
```

	tcp.seq	tcp.nextseq	tcp.ack	tcp.checksum	tcp.checksum.dec	tcp.flags.syn	class
0	1275068416	1275068416	0	0x0000e296	58006	1	1
1	1862270976	1862270976	0	0x0000bf96	49046	1	1
2	1912602624	1912602624	0	0x0000bc96	48278	1	1
3	1694498816	1694498816	0	0x0000c996	51606	1	1
4	1828716544	1828716544	0	0x0000c196	49558	1	1

Figure 21. Reading in Excel File Using Pandas.

The figure above shows the 'read_excel' method used to read in an Excel file using the Pandas library. This method returns a Pandas DataFrame from the Excel file. The name of the Excel file to be opened is passed as a parameter. The next method 'head' is used to show the observations in the data set. In the figure above, the number five has been passed to show the first five observations in the DataFrame.

6.1.2 Dropping Features

```
# The checksum hexadecimal values are dropped
# Instead, the decimal values are used
dataset = dataset.drop(['tcp.checksum'], axis=1)
```

Figure 22. Dropping a Feature.

The figure above contains the code used to drop the 'tcp.checksum' feature using the 'drop' function. This feature was dropped because machine learning algorithms do not work with hexadecimal values. Two parameters have been passed in this function: the feature to drop inside a list, and the axis to specify the column.

```
dataset.head(5)
```

	tcp.seq	tcp.nxtseq	tcp.ack	tcp.checksum.dec	tcp.flags.syn	class
0	1275068416	1275068416	0	58006	1	1
1	1862270976	1862270976	0	49046	1	1
2	1912602624	1912602624	0	48278	1	1
3	1694498816	1694498816	0	51606	1	1
4	1828716544	1828716544	0	49558	1	1

Figure 23. After Dropping 'tcp.checksum' Feature.

The figure above shows the Pandas DataFrame after dropping the 'tcp.checksum' feature. After dropping the feature, the decimal version ('tcp.checksum.dec') of checksum feature will be used instead.

6.1.3 Initialising Independent and Dependent Variables

```
# Set the independent and dependent variables
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
```

Figure 24. Setting Independent and Dependent Variables.

The figure above contains the code used to initialise the independent and dependent variables. The method 'iloc' is used to extract the observations and features from the data set. The 'X' variable contains all the observations and their features, excluding the 'class' feature. The 'y' variable contains only 'class' feature for all the observations in the data set. The contents of 'X' and 'y' variables can be seen in the figure below.

```
In [14]: # View the independent variables
X.head()

Out[14]:
```

	tcp.seq	tcp.nxtseq	tcp.ack	tcp.checksum.dec	tcp.flags.syn
0	1275068416	1275068416	0	58006	1
1	1862270976	1862270976	0	49046	1
2	1912602624	1912602624	0	48278	1
3	1694498816	1694498816	0	51606	1
4	1828716544	1828716544	0	49558	1

```

In [15]: # Class
y.head()

Out[15]:
```

0	1
1	1
2	1
3	1
4	1

```
Name: class, dtype: int64
```

Figure 25. DataFrames of X and y variables.

6.1.4 Splitting Data into Training and Testing Sets

```
In [16]: # Split the dataset into training and testing sets
# Stratify it so there is equal number of packets in the two classes
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y)

In [17]: # Check the size the datasets
print(X_train.shape)
print(X_test.shape)
print()
print(y_train.shape)
print(y_test.shape)

(75298, 5)
(25100, 5)

(75298,)
(25100,)
```

Figure 26. Splitting Data into Training and Testing Sets.

The figure above shows how the data set is split into training and testing sets using the 'train_test_split' method provided by the sklearn library. This function takes in a few parameters. The 'X' and 'y' parameters are the observations that is split. The 'test_size' parameter specifies the size of the testing data set. In this project, 0.25 will be used for all the data sets; therefore, 25100 observations will be used for testing and evaluating the performance. The 'stratify' parameter specifies the data must be split into equal number of observations for each class in the data set.

6.1.5 Feature Scaling

```
In [18]: # Scale the variables for faster computation
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 27. Scaling Features.

The figure above shows the scaling process. This process is used to make computations faster and to prevent the values from dominating each other. It does this by making sure the values are in a specific range. After all these stages are carried out, then the next step is to instantiate and train the machine learning algorithms.

6.2 Initialising Logistic Regression Classifier

This figure below contains the code used to create the Logistic Regression model using the sklearn library. Firstly, the Logistic Regression model is imported from the sklearn library. Then the model is instantiated and initialised to the variable 'classifier'. After this, the training data 'X_train' and 'y_train' are used to train the classifier model. Lastly, the classifier is used to predict the outcome of the unseen observations stored in the 'X_test' data set.

```

In [18]: # Use Logistic regression to fit
         from sklearn.linear_model import LogisticRegression

In [19]: classifier = LogisticRegression(solver='lbfgs')
         classifier.fit(X_train, y_train)

Out[19]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=100,
                             multi_class='warn', n_jobs=None, penalty='l2',
                             random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                             warm_start=False)

In [20]: # Predict the test data
         y_predict = classifier.predict(X_test)
         print(y_predict)

[0 0 0 ... 1 1 1]

```

Figure 28. Logistic Regression Classifier Initialisation.

6.3 Initialising Naïve Bayes Classifier

The figure below contains the code used to create the Naïve Bayes classifier. To accomplish this, the 'GaussianNB' is used from the sklearn library. After initialising the Naïve Bayes classifier to the variable 'classifier', the training data is used to train the model. In the code below, no parameter has been passed inside the constructor of the GaussianNB object; therefore, the default parameters are used. Then the test data was predicted using the classifier.

```

In [18]: # Use naive bayes to fit
         from sklearn.naive_bayes import GaussianNB

In [19]: classifier = GaussianNB()
         classifier.fit(X_train, y_train)

Out[19]: GaussianNB(priors=None, var_smoothing=1e-09)

In [20]: # Predict the test data
         y_predict = classifier.predict(X_test)
         print(y_predict)

[0 1 0 ... 0 1 0]

```

Figure 29. Naïve Bayes Classifier Initialisation.

6.4 Initialising SVM Gaussian Kernel Classifier

Another algorithm that will be used in this project is the Support Vector Machine (SVM) using the Radial Basis Function kernel. The figure below contains the code used to create the SVM classifier. The gamma parameter is set to 'auto', so the algorithm can automatically adjust the gamma value by itself. Then the classifier is trained using the training data set. The test data set is also predicted using the trained model.


```

In [18]: # Use SVM rbf kernel to fit
         from sklearn.svm import SVC

In [19]: classifier = SVC(kernel='rbf', gamma='auto')
         classifier.fit(X_train, y_train)

Out[19]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
            max_iter=-1, probability=False, random_state=None, shrinking=True,
            tol=0.001, verbose=False)

In [20]: # Predict the test data
         y_predict = classifier.predict(X_test)
         print(y_predict)

[0 1 0 ... 0 0 1]

```

Figure 30. SVM Gaussian Kernel Classifier Initialisation.

6.5 Initialising Voting Ensemble Learning Classifier

```

In [18]: # Import the necessary modules
         from sklearn.linear_model import LogisticRegression
         from sklearn.naive_bayes import GaussianNB
         from sklearn.svm import SVC
         from sklearn.ensemble import VotingClassifier

         lr = LogisticRegression(solver='lbfgs')
         nb = GaussianNB()
         svm = SVC(kernel='rbf', gamma='auto')
         classifier = VotingClassifier(estimators=[('lr', lr), ('nb', nb), ('svm', svm)], voting='hard')
         classifier.fit(X_train, y_train)

```

Figure 31. Voting Classifier Initialisation.

The final algorithm that will be used in this project is the Voting Classifier. This algorithm is different compared to others because it combines different models to create a single model. In the figure above, a Voting Classifier is instantiated and set to the 'classifier' variable. Logistic Regression, Naïve Bayes, and SVM are the estimator algorithms. The voting parameter specifies how the estimators should vote. The 'hard' parameter specifies the prediction is done using a majority rule voting, that is the estimators will vote whether the packet has a covert channel or not.

6.6 Implementing the Passive Warden System

This section builds on the previous chapter to build the passive warden system. Only the important parts that focus on the detection of covert channels will be shown in this section; therefore, irrelevant parts such as reading user input, printing formatted messages, etc., will not be shown. Also, since the code for initialising the trained models are very similar to each other, only one example would be shown in this section.

6.6.1 Initialising the Passive Warden

```
def initialise_sniffer(self):
    # The number of packets to sniff is controlled by the constant at the beginning of the file
    print(f'Sniffing the first {total_packets_to_sniff} packets only')
    sleep(1)
    # No iface (interface) has been set here; therefore, it will capture packets from every interface
    # However, iface can be initialised to listen to a specific interface (e.g. lo for loopback address)
    sniff(prn=self.sniff_packet, count=total_packets_to_sniff)

def open_capture_file(self, name):
    file_exists = os.path.exists(name)
    if file_exists:
        # Read the Wireshark capture file and the packets inside it
        return rdpcap(name)
    print(f'File \'{name}\' does not exist')
    print('Terminating program now')
    exit(0)
```

Figure 32. Setting the Parameters of the Passive Warden.

The figure above contains the code used to initialise the passive warden. The first function initialises the sniffer to allow the warden to sniff packets in real time. Inside the 'sniff' function, two parameters have been set; the 'prn' parameter specifies how the sniffed packets should be processed, and the 'count' parameter specifies the total number of packets to sniff in real-time. The second function 'open_capture_file' is used to open a Wireshark capture file using the 'scapy' library; therefore, the passive warden can detect covert storage channels in Wireshark capture files.

6.6.2 Loading the Trained Models

```
#####
# The Trained Models
def load_ip_id_model():
    return pickle.load(open('Models//Covert_tcp IP ID//covert_tcp_ip_id_svm_model', 'rb'))

def load_covert_tcp_seq_model():
    return pickle.load(open('Models//Covert_tcp TCP SEQ//covert_tcp_seq_nb_model', 'rb'))

def load_covert_tcp_ack_bounce_model():
    return pickle.load(open('Models//Covert_tcp ACK Bounce//covert_tcp_ack_bounce_lr_model', 'rb'))

def load_ncovert_tcp_seq_model():
    return pickle.load(open('Models//NCovert TCP SEQ//ncovert_tcp_seq_lr_model', 'rb'))

def load_ncovert_tcp_ack_bounce_model():
    return pickle.load(open('Models//NCovert ACK Bounce//ncovert_ack_bounce_nb_model', 'rb'))
#####
```

Figure 33. Loading Trained Machine Learning Models.

The figure above shows how the trained models are loaded using the 'load' function. Inside the 'open' function, the location of the trained model and 'rb' have been passed, where the 'rb' parameter specifies the file can only be read in binary format. Each function above returns a specific trained model which will be used for prediction. The Voting Classifier performed better in certain occasions, but it was not used in this project due to runtime exceptions thrown by the object; therefore, the second-best performing algorithm was used instead when Voting Classifier performed the best in detecting a specific technique.

6.6.3 Example: Initialising the IP ID Trained Model

```
def process_ip_id(packet):
    # Load the scaler used to train the SVM model
    ip_id_scaler = load('Scalers/Covert_tcp_IP_ID/scaler_covert_tcp_ip_id_svm_model.bin')
    # Load the IP ID SVM model
    ip_id_model = load_ip_id_model()
    # Extract the necessary features from the passed packet
    packet_id = packet['IP'].id
    packet_checksum = packet['IP'].chksum
    packet_df_flag = packet['IP'].flags.DF
    source = packet['IP'].src
    destination = packet['IP'].dst
    # Build a list containing the features
    features = [packet_id, packet_df_flag, packet_checksum]
    # Scale the features and predict whether the packet has a covert channel or not
    # If result is not 0, then it is predicted to have a covert channel
    result = ip_id_model.predict(ip_id_scaler.transform(features))
    if result:
        print('Covert channel detected in ', end=' ')
        print(features, end=' ')
        # Extract the hidden data inside the IP ID field
        hidden_data = chr(packet_id // 256)
        print(f'Hidden data was {hidden_data} \'; Sent by {source} to {destination}')
        return [source, destination, hidden_data]
    else:
        print('No covert channel detected in ', end=' ')
        print(features, end='\n')
```

Figure 34. Initialising the SVM IP ID Model.

The figure above shows how the IP ID SVM model is initialised. Firstly, the scaler used to train the SVM model is loaded using the 'load' function. Then, the IP ID SVM model is loaded and initialised to the 'ip_id_model' variable. Afterwards, the necessary features are extracted from the packet that has been passed as an argument and stored in a list for prediction. Before the model predicts whether the packet has a covert channel or not, the features are scaled using the scaler object that was used to train the SVM IP ID model. If the model detects there is a covert channel present, then it will extract the hidden data from the IP ID, decode it, and print to the screen.

6.6.4 Detection Example: IP ID Covert Storage Channels

Now the passive warden system is complete, it is time to deploy the system. Figure 35 shows that the passive warden can detect IP ID covert storage channels in a Wireshark capture file. The Wireshark file 'firstmethodCHR.pcapng' contains communications between hosts 10.0.2.15 and 10.0.3.5. Each datagram sent by 10.0.2.15 contains a hidden character inside the IP ID field. The host 10.0.2.15 sent the message 'Ghost in the wire' to 10.0.3.5, and the warden detected this successfully. Figure 36 shows that the passive warden can detect IP ID covert storage channels in real time using its packet sniffing functionality. The passive warden can detect the covert packets sent by host 10.0.3.7 to 10.0.2.6, and decode the hidden data inside the covert IP ID datagrams.

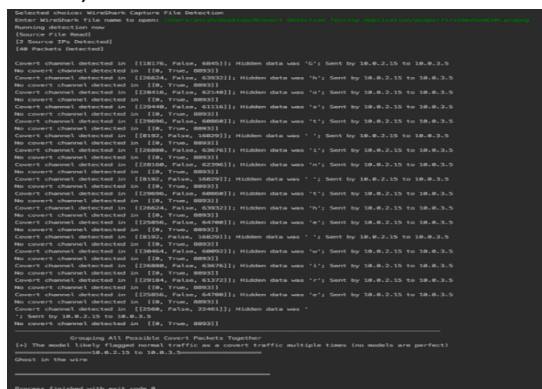


Figure 35. Detecting IP ID CSCs in a Wireshark Capture File.

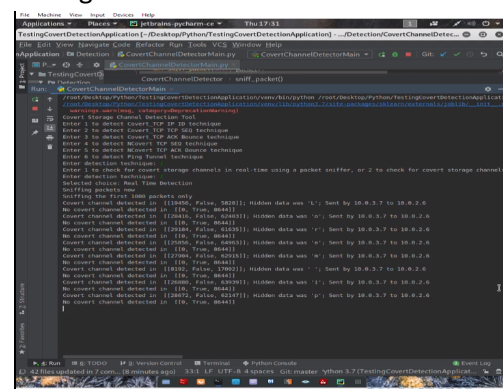


Figure 36. Detecting IP ID CSCs in Real Time.

6.7 Summary

This chapter looked into the pre-processing stages required before training the different machine learning algorithms. These stages consisted of reading in data, dropping irrelevant features, initialising the dependent and independent variables, splitting data into training and testing sets, and scaling the variables. Then the machine learning algorithms were instantiated and trained. Also, the passive warden system was developed. The next chapter will focus on the evaluation of the different models.

7 Evaluation

This chapter will focus on the evaluation of the trained models using different classification evaluation techniques.

7.1 Evaluating Covert_tcp IP ID Detection Models

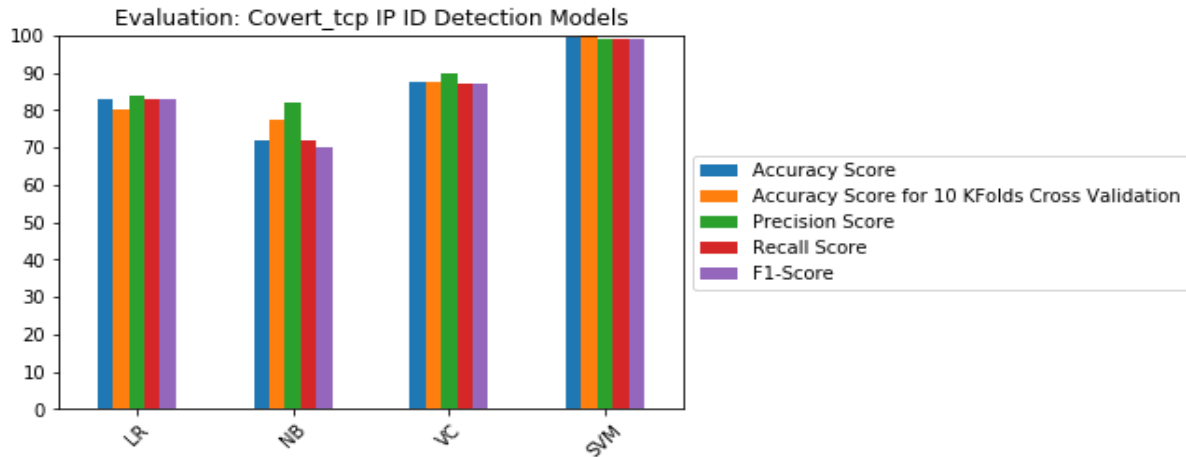


Figure 37. Performance of Covert_tcp IP ID Detection Models.

$\begin{bmatrix} 9408 & 3142 \\ 1142 & 11408 \end{bmatrix}$	$\begin{bmatrix} 5534 & 7016 \\ 0 & 12550 \end{bmatrix}$	$\begin{bmatrix} 9376 & 3174 \\ 0 & 12550 \end{bmatrix}$	$\begin{bmatrix} 12412 & 138 \\ 0 & 12550 \end{bmatrix}$
LR	NB	VC	SVM

Figure 38. Prediction Results of Trained Covert_tcp IP ID Detection Models.

The Naïve Bayes algorithm performed the worst in detecting IP ID covert storage channels. It had an accuracy score of 72% on the testing data set, and an average score of 77.21% for the ten k-folds cross validation. The scores for precision, recall, and F1 were 82%, 72%, and 70% on the testing data set. This model made 7016 incorrect false positive predictions on the testing data set, and 0 false negative predictions. Logistic Regression performed better than the Naïve Bayes algorithm. It had an accuracy score of 82% on the testing data set, and an average score of 80% for the ten k-folds cross-validation. The scores for precision, recall, and F1 were 84%, 83%, and 83% on the testing data set. This model made 3142 false positive predictions, and 1142 false negative predictions. The Voting Classifier model performed very well in detecting IP ID covert storage channels. It had an accuracy of 87.35% on the testing data set, and an average score of 87.77% for the ten k-folds cross validation. The scores for precision, recall, and F1 were 90%, 87%, and 87% on the testing data set. The Voting Classifier model made 3174 incorrect false positive predictions on the testing data set, and 0 false negative predictions. However, Support Vector Machine performed the best in detecting IP ID covert storage channels due to its non-linear behaviour. It had an accuracy score of 99.45% on the testing data set, and an average score of 100% for ten k-folds cross-validation. Its score for the precision, recall, and F1 were 99%, 99%,

and 99% on the testing data set. Although this model was very accurate, it still made 138 false positive predictions on the testing data set with 0 false negatives. Therefore, SVM algorithm performed the best in detecting IP ID covert storage channels. This trained model will be used by the passive warden system to detect IP ID covert storage channels.

7.2 Evaluating Covert_tcp TCP Sequence Detection Models

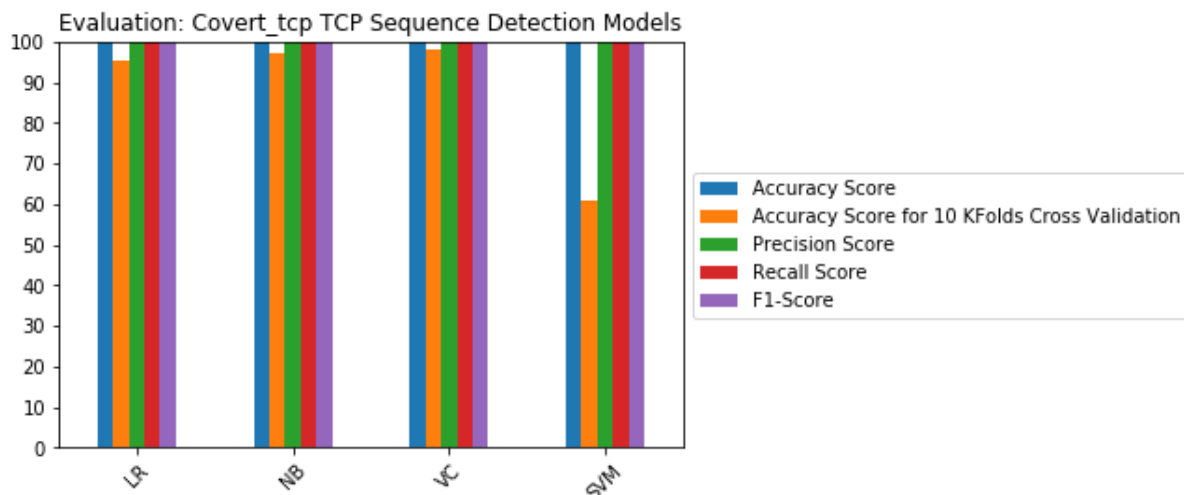


Figure 39. Performance of Covert_tcp TCP Sequence Detection Models.

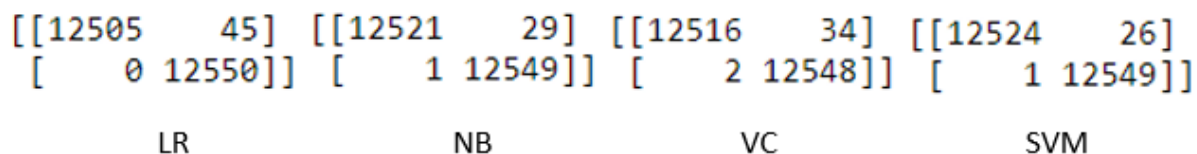


Figure 40. Prediction Results of Trained Covert_tcp TCP Sequence Detection Models.

The Support Vector Machine performed the worst in detecting this technique. It had an accuracy of 99.89% on the testing data set, and an average score of 61.09% for the ten k-folds cross validation. This algorithm suffered from over-fitting and was not able to generalise well on the different parts of the data set. The other three algorithms performed exceptionally well in detecting this technique. The Logistic Regression had an accuracy score of 99.82% on the testing data set, and an average score of 95.38% for the ten k-folds cross validation. This model made 45 false positive predictions, and 0 false negative predictions. Its precision, recall, and F1 scores were 100%, 100%, and 100% respectively on the testing data set. The Naïve Bayes had an accuracy score of 99.88% on the testing data set, and an average score of 97.02% for the ten k-folds cross validation. This model made 29 false positive predictions, and 1 false negative prediction. Its precision, recall, and F1 scores were 100%, 100%, and 100% on the testing data set. The Voting Classifier model performed the best in detecting this technique. It had an accuracy of 99.86% on the testing data set, and an average score of 97.89% for the ten k-folds cross validation. Its precision, recall, and F1 scores were 100% on the testing data set. Naïve Bayes and Voting Classifier algorithms performed the best in detecting this technique because

this technique relies on setting the acknowledgement field to 0 and the synchronise flag to 1; therefore, the next sequence field is also the same as the current sequence field because the TCP segment does not carry a payload data.

7.3 Evaluating Covert_tcp Bounce Acknowledgement Models

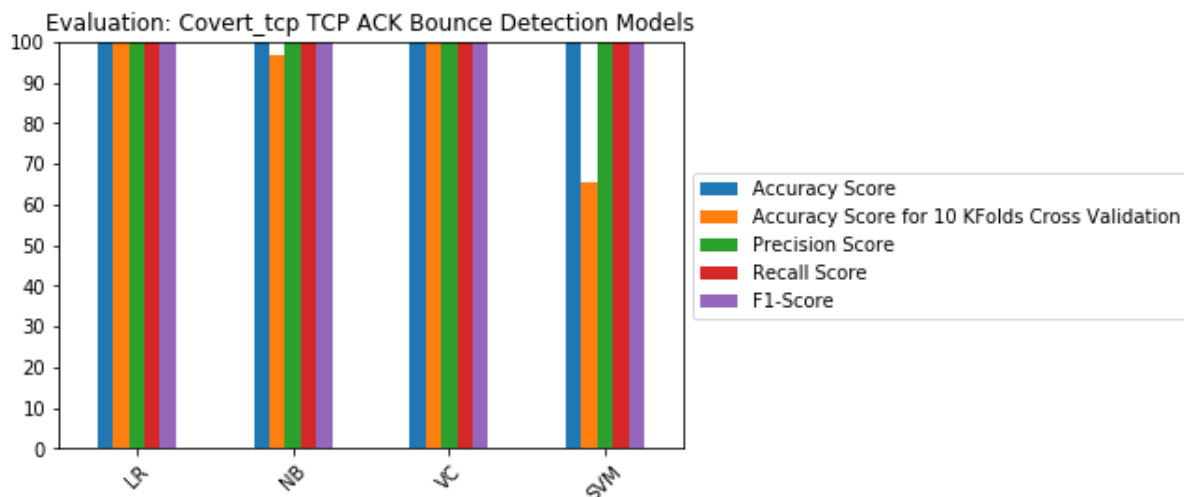


Figure 41. Performance of Covert_tcp TCP ACK Bounce Detection Models.

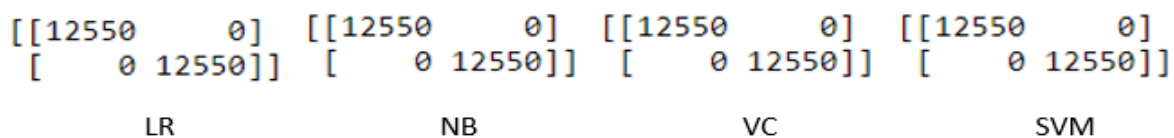


Figure 42. Prediction Results of Trained Covert_tcp TCP ACK Detection Models.

The Support Vector Machine performed the worst again as it could not generalise well on the different parts of the data set. It had an accuracy of 100% on the testing data set, but an average score of 65.45% for the ten k-folds cross validation. The Naïve Bayes model performed very well and had an accuracy of 100% on the testing data set, and an average score of 96.73% for the ten k-folds cross validation. This model has zero false positives and zero false negatives on the testing data set. The Logistic Regression and Voting Classifier models performed the best. Both of the models had an accuracy of 100% on the testing data set, and an average score of 100% for the ten k-folds cross validation. The precision, recall, and F1 scores were 100% for both of these models on the testing data set. These models also achieved very high true positives and true negatives. Logistic Regression and Voting Classifier algorithms performed the best in detecting this technique. This technique was easy to detect because of its unusual network traffic patterns; for example, it sets the sequence and next sequence fields to 0, reset and acknowledgment flags to 1 because the covert packet is bounced off a server.

7.4 Evaluating NCovert TCP Sequence Detection Models

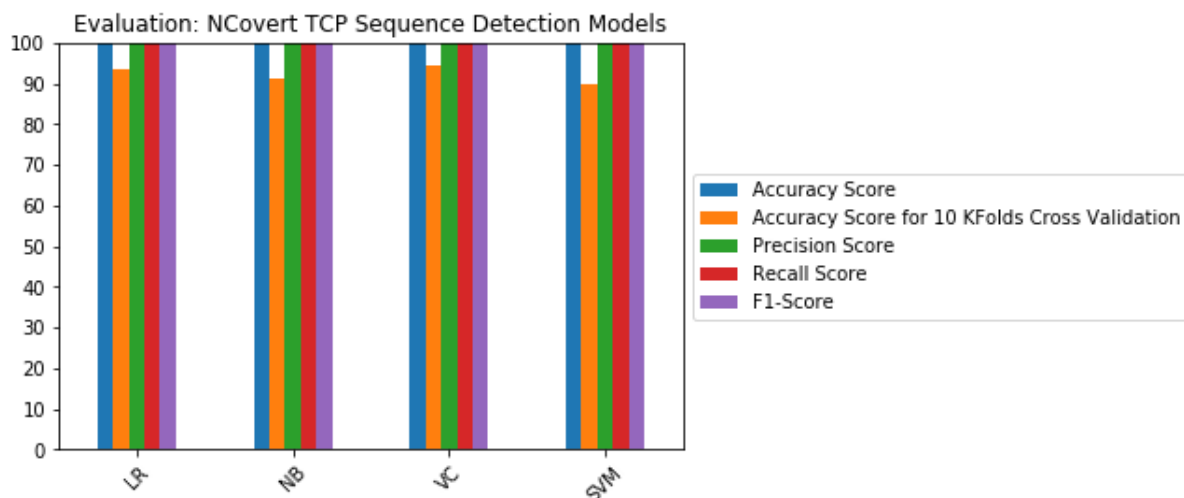


Figure 43. Performance of NCovert TCP Sequence Detection Models.

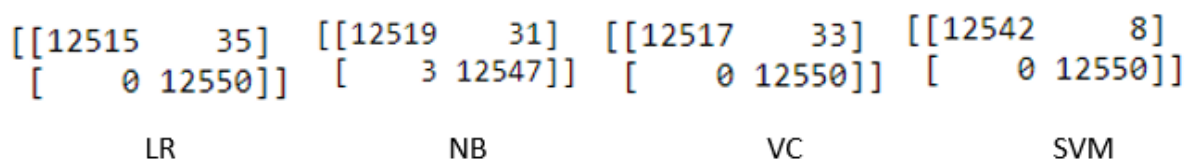


Figure 44. Prediction Results of Trained NCovert TCP Sequence Detection Models.

The Support Vector Machine performed the worst in detecting this technique. It achieved an accuracy of 99.87% on the testing data set, and an average score of 89.96% for the ten k-folds cross validation. This algorithm is unable to generalise well on the different parts of the data set due to many input features. Naïve Bayes model performed slightly better than the SVM model. It has an accuracy of 99.86% on the testing data set, and an average score of 91.14% for the ten k-folds cross validation. This model made 31 false positive predictions, and 3 false negative predictions on the testing data set. The Logistic Regression model performed better. It achieved an accuracy of 99.86% on the testing data set, and an average score of 93.69% for the ten k-folds cross validation. This model made 35 false positive predictions and 0 false negative predictions. The Voting Classifier model performed the best with an accuracy of 99.87% on the testing data set and an average score of 94.43% for the ten k-folds cross validation. This model made 33 false positive predictions on the testing data set. This technique is difficult to detect because it hides four bytes of data in the sequence field of the TCP segment. The precision, recall, and F1 scores for the four models were 100% on the testing data set.

7.5 Evaluating NCovert Bounce Acknowledgement Detection Models

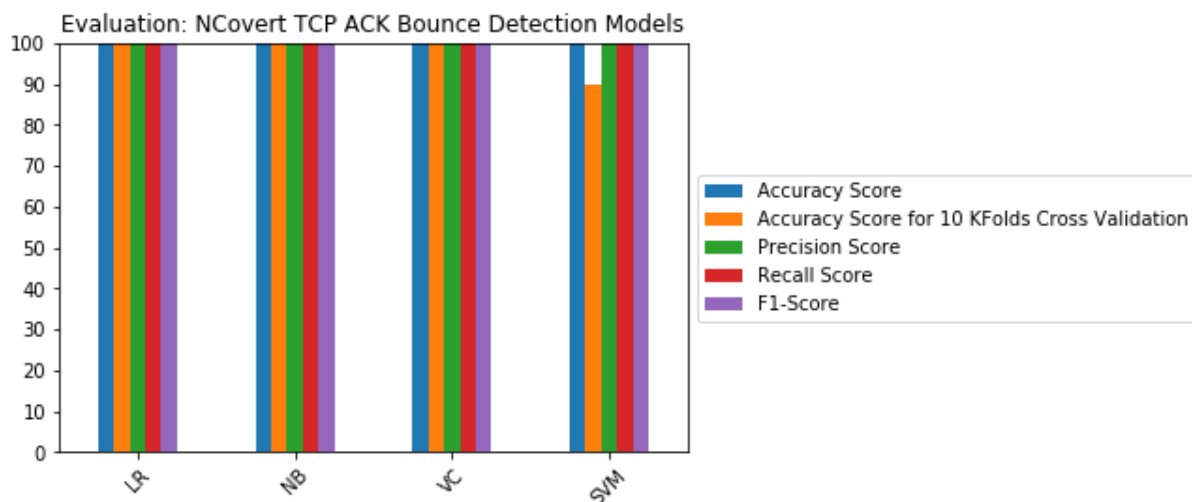


Figure 45. Performance of NCovert ACK Bounce Detection Models.

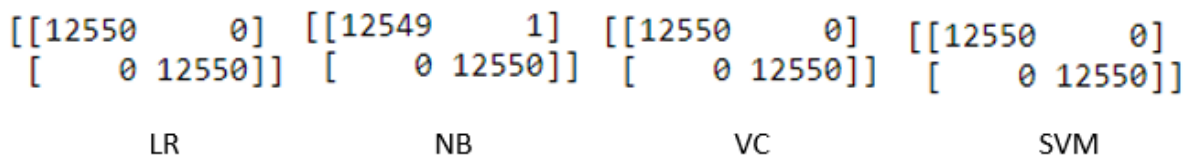


Figure 46. Prediction Results of Trained NCovert ACK Bounce Detection Models.

The Support Vector Machine algorithm performed well on the testing data set but achieved an average score of 89.96% for the ten k-folds cross validation. Logistic Regression, Naïve Bayes, and Voting Classifier models performed very well in detecting this technique. This technique uses the same technique as Covert_tcp (e.g. setting the reset and acknowledgement flags to 1), but only one thing is different, that is, it hides four bytes of data inside the acknowledgement field of the TCP segment instead of one byte. The Logistic Regression, Naïve Bayes, and Voting Classifier models achieved an accuracy score of over 99.99% on the testing data set. The precision, recall, and F1 scores for the three models were 100% on the testing data set.

8 Conclusions

The main aim of this project was to use Machine Learning algorithms to detect the five covert storage channel techniques used by Covert_tcp and NCovert. This was successful because ML algorithms were able to detect the five covert storage channel techniques with very high detection rates. This project also proved that No Free Lunch theorem exists in detection of covert storage channels because different algorithms performed better in different scenarios; for example, SVM performed the best in detecting IP ID covert storage channels, but this algorithm suffered when the technique to detect required many input features, such as TCP covert storage channels. The table below specifies the best

performing models, and the trained models used by the passive warden system to detect the different covert storage channels in real-time and in Wireshark capture files.

Tool	Data Hiding Technique	Best Performing Model	Second Best Performing Model	Model Used by Passive Warden
Covert_tcp	IP ID	SVM	Voting Classifier	SVM
Covert_tcp	TCP Sequence	Voting Classifier	Naïve Bayes	Naïve Bayes
Covert_tcp	TCP Acknowledgement	Logistic Regression	Voting Classifier	Logistic Regression
NCovert	TCP Sequence	Voting Classifier	Logistic Regression	Logistic Regression
NCovert	TCP Acknowledgement	Naïve Bayes	Voting Classifier	Naïve Bayes

Table 7. Best Performing Detection Models.

The SVM suffered from The Curse of Dimensionality because it was not able to generalise well on data with many input features, such as TCP covert storage channels. SVM only performed well when the technique to detect required a small number of input features, such as Covert_tcp IP ID covert storage channels. This result is same compared to the result produced by [13] because SVM performed the best in detecting Covert_tcp IP ID covert storage channels (see figures 37 and 38). This model achieved a very high detection rate; therefore, SVM is suitable in detecting IP ID CSCs.

Detecting Covert_tcp and NCovert covert storage channels in TCP sequence field was easier because these techniques use unusual network traffic patterns compared to normal network traffic; for example, setting the reset and ack flags to 1 at the same time. The Voting Classifier algorithm performed the best in detecting the two types (one byte, and four bytes of data) of TCP sequence covert storage channels (see figures 39 and 40, and 41 and 42) by combining multiple algorithms together to create a single model. This algorithm achieved very high detection rates on both of the techniques with very high true positives and true negatives.

Detecting TCP acknowledgement covert channels was successful. This technique also used unusual network traffic patterns compared to normal network traffic, thus making it easier to detect. Logistic Regression performed the best in detecting Covert_tcp acknowledgement covert channels (see figures 41 and 42), where only one byte of data is hidden inside the covert packet. Naïve Bayes performed the best when detecting NCovert TCP acknowledgement covert channels (four bytes of data inside ack field). This model achieved a very high detection rate (see figures 45 and 46).

8.1 Future Work

An interesting area to explore is Protocol Switching Covert Storage Channels – where the data is hidden in different Internet protocols instead of just one in a two-way communication. The passive warden system can detect the five different techniques in real-time and in Wireshark capture files, but this system can be expanded further; for example, being able to detect more techniques, or even using the code to create an active warden system that can modify and remove hidden data inside covert packets.

8.2 Personal Reflection

This last section focuses on personal reflection, and reflection on the project.

8.2.1 Reflection on Project

This project covered a wide range of areas such as Computer Networking, AI, Cybersecurity, Computer Programming, etc. The AI field was completely new to me and I had zero experience in AI when I started this project. Not choosing the AI module offered by university meant I had to rely on other sources to learn AI. During the Christmas break, I purchased multiple Udemy courses to learn AI. During March, my supervisor selected me to present this project at Made In Brunel Software Innovation event. My project fell under the AI category with other final year students who used AI in their project. I was able to successfully present my project to a woman - who was judging all final year AI projects - from Lloyds Bank. Out of all the final year AI projects, she picked me as the winner, and I won the Level 3 AI Innovation Prize. This meant I had the best AI final year project.

8.2.2 Personal Reflection

Time management was my biggest weakness. Even though I followed the methodology of this project, I still lost a lot of time during the design stage. This was because collecting normal and covert data, and preparing it took more than a month for the five techniques. I could have done better if I developed a time plan and followed it from the start. This would have allowed me to tackle more techniques used by existing tools such as Ping Tunnel.

References

- [1] Rowland, C.H. (1997). Covert channels in the TCP/IP protocol suite. *First Monday*, [online] 2(5). Available at: <https://firstmonday.org/ojs/index.php/fm/article/view/528/449> [Accessed 7 Dec. 2019].
- [2] Sourceforge.net. (2004). *NMRC Projects - NCovert*. [online] Available at: <http://ncovert.sourceforge.net/> [Accessed 7 Dec. 2019].
- [3] Cisco. (2017). *IPv6 Tunnel through an IPv4 Network*. [online] Available at: <https://www.cisco.com/c/en/us/support/docs/ip/ip-version-6/25156-ipv6tunnel.html> [Accessed 7 Dec. 2019].
- [4] National Computer Security Center (NCSC). (1985). *DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA*. [online] Available at: <https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/dod85.pdf> [Accessed 7 Dec. 2019]. pp. 107.
- [5] Mileva, A. and Panajotov, B. (2014). Covert channels in TCP/IP protocol stack - extended version-. *Open Computer Science*, 4(2).
- [6] Zander, Sebastian & Armitage, Grenville & Branch, Philip. (2007). *Covert channels and countermeasures in computer network protocols*. IEEE Communications Surveys and Tutorials. 9. 44-57. 10.1109/COMST.2007.4317620.
- [7] IETF. (1981). *RFC 793 - Transmission Control Protocol*. [online] Available at: <https://tools.ietf.org/html/rfc793> [Accessed 21 Oct. 2019].
- [8] Qian, Yuwen & Huaju, Song & Chao, Song & Xi, Wang & Linjie, Leng. (2012). *Network Covert Channel Detection with Cluster based on Hierarchy and Density*. Procedia Engineering. 29. 4175-4180. 10.1016/j.proeng.2012.01.639.
- [9] Giani, Annarita & Berk, Vincent & Cybenko, George. (2006). Data exfiltration and covert channels. Proceedings of SPIE - The International Society for Optical Engineering. 10.1117/12.670123.
- [10] Sbrusch, R. and Sbrusch, R. (2006). *SANS Institute Information Security Reading Room Network Covert Channels: Subversive Secrecy* [online] Available at: <https://www.sans.org/reading-room/whitepapers/covert/network-covert-channels-subversive-secrecy-1660> [Accessed 8 Jan. 2020].

- [11] Sewell, M. (2020). *No Free Lunch Theorems*. [online] No-free-lunch.org. Available at: <http://www.no-free-lunch.org/> [Accessed 31 Jan. 2020].
- [12] Mirkovic, Jelena & Reiher, Peter. (2004). A taxonomy of DDoS attack and DDoS Defense mechanisms. *ACM SIGCOMM Computer Communication Review*. 34. 10.1145/997150.997156.
- [13] M. A. Ayub, S. Smith and A. Siraj, "A Protocol Independent Approach in Network Covert Channel Detection," *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, New York, NY, USA, 2019, pp. 165-170.
- [14] Mitre.org. (2019). *Exfiltration Over Alternative Protocol, Technique T1048 - Enterprise | MITRE ATT&CK™*. [online] Available at: <https://attack.mitre.org/techniques/T1048/> [Accessed 11 Mar. 2020].
- [15] Mitre.org. (2019). *Data Encrypted - Enterprise | MITRE ATT&CK™*. [online] Available at: <https://attack.mitre.org/techniques/T1022/> [Accessed 11 Mar. 2020].
- [16] Mitre.org. (2019). *Scheduled Transfer, Technique T1029 - Enterprise | MITRE ATT&CK™*. [online] Available at: <https://attack.mitre.org/techniques/T1029/> [Accessed 11 Mar. 2020].
- [17] Mitre.org. (2019). *Spearphishing Attachment, Technique T1193 - Enterprise | MITRE ATT&CK®*. [online] Available at: <https://attack.mitre.org/techniques/T1193/> [Accessed 11 Mar. 2020].
- [18] Shon, Taeshik & Moon, Jongsub & Lee, Sangjin & Lim, Jongin. (2003). *Covert Channel Detection in the ICMP Payload Using Support Vector Machine*. 828-835. 10.1007/978-3-540-39737-3_103.
- [19] Shen, Yao & Huang, Liusheng & Lu, Xiaorong & Yang, Wei. (2014). *A novel comprehensive steganalysis of transmission control protocol/Internet protocol covert channels based on protocol behaviors and support vector machine*. *Security and Communication Networks*. 8. 10.1002/sec.1081.

BREO Project Submission

Project Details

A1 Proposed Project Title

Storage Covert Channels Over Network Traffic

Applicant Details

A2 Applicant Details

Title

First Name

Surname

Mr

Anish

Limbu

Brunel Email address

1618834@brunel.ac.uk

A3 Are there other researcher(s) who will work on the research project?

☐ Yes

☒ No

A5 Applicant Status: Please select the level in which you are carrying out the research:

Undergraduate Student

Please select your College (Only if you are professional staff, please select 'No College'):

College of Engineering, Design and Physical Sciences

Please select your Department:

Computer Science

Please select your Institute (if you do not belong to an Institute, select 'No Institute'):

No Institute

Student Details

A6 Student Number

1618834

A7 Module Name and Number

CS3072-CS3605 Computer Science-Business Computing Final Year Project

A8 Project Submission Deadline Date

27/03/2020

Please use this box **only** if you are resubmitting an application and you need to explain any discrepancies between your Proposed Start Date and your Project Submission Deadline Date:

A9 Supervisor Details

Title	First Name	Surname
<input type="text" value="Mr"/>	<input type="text" value="Panos"/>	<input type="text" value="Louvieris"/>
College	<input type="text" value="College of Engineering, Design and Physical Sciences"/>	
Department	<input type="text" value="Computer Science"/>	
Brunel Email address	<input type="text" value="panos.louvieris@brunel.ac.uk"/>	

Risk Factors

A10 Does your research involve the recruitment of NHS patients and/or staff, HM Prison and Probation Services, or requires approval from the Social Care Research Ethics Committee?

☐ Yes☒ No**Risk Factors**

A11 Does your research fit into any of the following security-sensitive categories?

- ☐ Concerns the military;
- ☐ Set up under an EU security call;
- ☐ Involves seeking a security clearance;
- ☐ Concerns terrorist or extremist groups;

☒ None of the above

A12 Does your research involve the use of human participants, their data and/or their tissue? Please tick yes if your research includes sensitive financial data, data obtained through questionnaires/surveys, face-face/phone interviews, or focus groups
See the information (i) icon for guidance.

- ☐ Yes
☒ No

Please state how your research does not involve human participants or their data and/or tissue. (For example, your project uses only secondary data, data that is already publically available, or data that is already anonymised)

Research only requires network packets using an application called Wireshark.

Research Integrity

A19 Have you completed the Research Integrity Online Training relevant to your field of research via Blackboard Learn?

- ☒ Yes
☐ No

If yes, please provide details:

Yes, I attended the lecture and read through the handbook.

Start and End Date

J1 Proposed Start Date

14/11/2019

NOTE: You are not able to select a start date within a week of your application. If you are using human participants, please select a suitable start date that allows time for the review and approval of your application. **Please see the information icon (i) for further guidance.**

J2 Proposed End Date:

27/03/2020

Researcher/Applicant

Researcher/Applicant Signature

- I understand that I cannot commence my research until full research ethics approval has been granted by the relevant research ethics committee.
- I understand that it can take up to 15 working days to receive feedback on my application.
- I confirm that the research will be undertaken in accordance with the [Brunel University London Ethical Framework](#), [Brunel University London Code of Research Ethics](#), and [Brunel University London Research Integrity Code](#).
- I shall ensure that any changes in approved research protocols are reported promptly for approval by the relevant University Research Ethics Committee.
- I shall ensure that the research study complies with the law and Brunel University London policies on the use of human material (if applicable) and health and safety.
- I am satisfied that the research study is compliant with the Data Protection Act 2018, and that necessary arrangements will be made with regard to the storage and processing of participants' personal information and generally, to ensure confidentiality of such data supplied and generated in the course of the research.

(Note: Where relevant, further advice is available from the Data Protection Officer, e-mail data-protection@brunel.ac.uk).

- I will ensure that all adverse or unforeseen problems arising from the research project are reported in a timely fashion to the Chair of the relevant Research Ethics Committee.
- (For members of staff and PhD students) I will undertake to provide notification to the Chair of the relevant Research Ethics Committee when the study is complete, or if it fails to start or is abandoned.

Signed: This form was signed by Anish Limbu (1618834@brunel.ac.uk) on 07/11/2019 21:03

Supervisor

Signature of Supervisor

- I confirm that I have met and advised the student on the ethical aspects of the study design and his/her responsibilities in relation to the submission of this application and the research.
- I confirm that the student has been advised to read the University's Code of Research Ethics and other relevant documentation.
- I confirm that the student has the skills to carry out the research.
- I confirm that if the research involves human participants, a Participant Information Sheet (PIS) is included and has been completed appropriately.
- I confirm that if the research involves human participants, the procedures for recruitment and obtaining informed consent are appropriate.
- I confirm that if there are issues of risk in the research, a full risk assessment has been undertaken and is attached.
- I confirm that A DBS check has been obtained (where appropriate).

Signed: This form was signed by Prof Panos Louvieris (Panos.Louvieris@brunel.ac.uk) on 03/02/2020 13:56