Configuration Manual

MSc Research Project

Master of Science in Cyber Security

Nikhil Nikhil

Student ID: 23161442

School of Computing

National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nikhil Nikhil		
Student ID:	23161442		
Programme:	Master of Science in Cyber Security	**Year:**	2023-24
Module:	Practicum Part 2		
Supervisor:	Michael Pantridge		
Submission Due Date:	12th August 2024		
Project Title:	Research Project Configuration manual		
Word Count:	783	**Page Count**	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Nikhil Nikhil

Date: 11-08-24

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nikhil Nikhil

Student ID: 23161442

Table of Contents

1 Introduction

The configuration manual outlines the tools and technologies utilized during the research implementation. Section 2 covers the experimental setup in detail. In Section 3, the technologies and software tools used are discussed. Section 4 provides a comprehensive, step-by-step guide for setting up the software tools, detailing the implementation process from installing and configuring Suricata, creating custom rules, generating attacks from Kali Linux, setting up email alerts, saving logs to Google Drive, and analysing and visualizing the attack patterns. Lastly, Section 5 contains the references for the software guide.

2 Experimental Setup

The experiment was carried out on a personal computer, which was set up specifically for this purpose.

- **Hardware Specifications:** The Windows system is equipped with a fifth-generation i5 processor, 8GB of RAM, and a 256GB SSD, while the Oracle Virtual Machine is configured with 4 CPU cores and 3000 MB of base memory for both the Ubuntu and Kali Linux virtual machines.
- **Operating System:** Windows 10, Ubuntu 18.04, Kali Linux
- **Experimental Setup:** Windows 10, Ubuntu 18.04, Kali Linux, Python 3.9, VS Code, Oracle VirtualBox

3 Technologies and Software Used for Implementation

- **Software Used:** VS Code, Oracle VirtualBox.
- Visual Studio Code (VS Code) and Oracle VirtualBox are two powerful tools that greatly enhance the coding and development experience. VS Code is a free, open-source code editor developed by Microsoft. It offers features such as debugging, syntax highlighting, intelligent code completion, and a vast library of extensions. These Features make it highly customizable and versatile for various programming languages and projects. Oracle VirtualBox is a free, open-source virtualization software that allows users to run multiple operating systems simultaneously on a single physical machine. This flexibility provides isolated environments for testing and development, supporting a wide range of guest

operating systems with features like snapshots, shared folders, and seamless mode (Wikipedia Contributors, 2019).

4 Implementation

Step 1: Install Ubuntu and Kali Linux on the virtual machine.

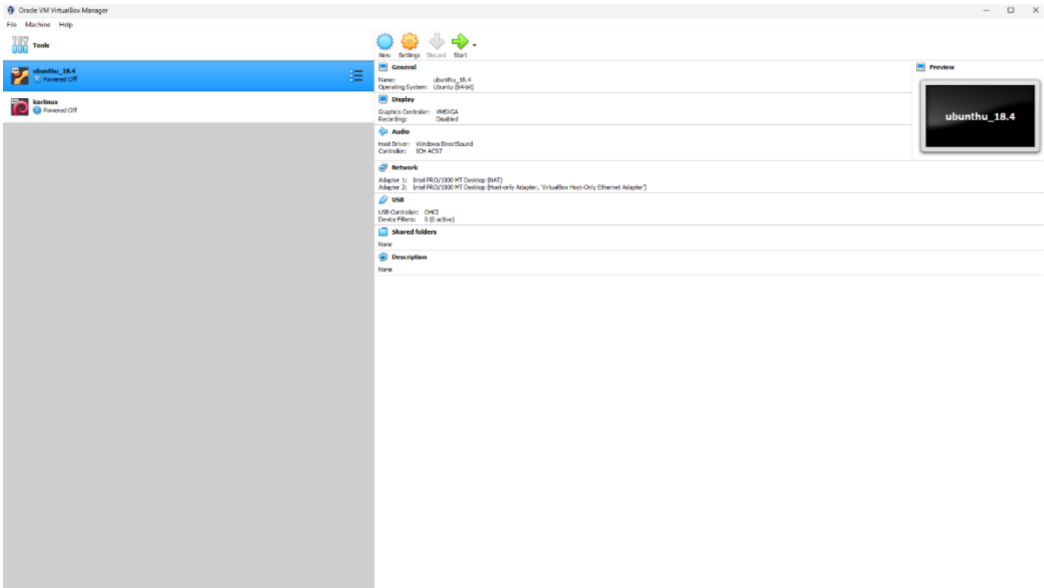


Figure (1):Successfully set up Ubuntu and Kali Linux

Step 2: Install Suricata on the Ubuntu system and configure its set up.

Step 3: Develop custom rules in Suricata to detect attacks.

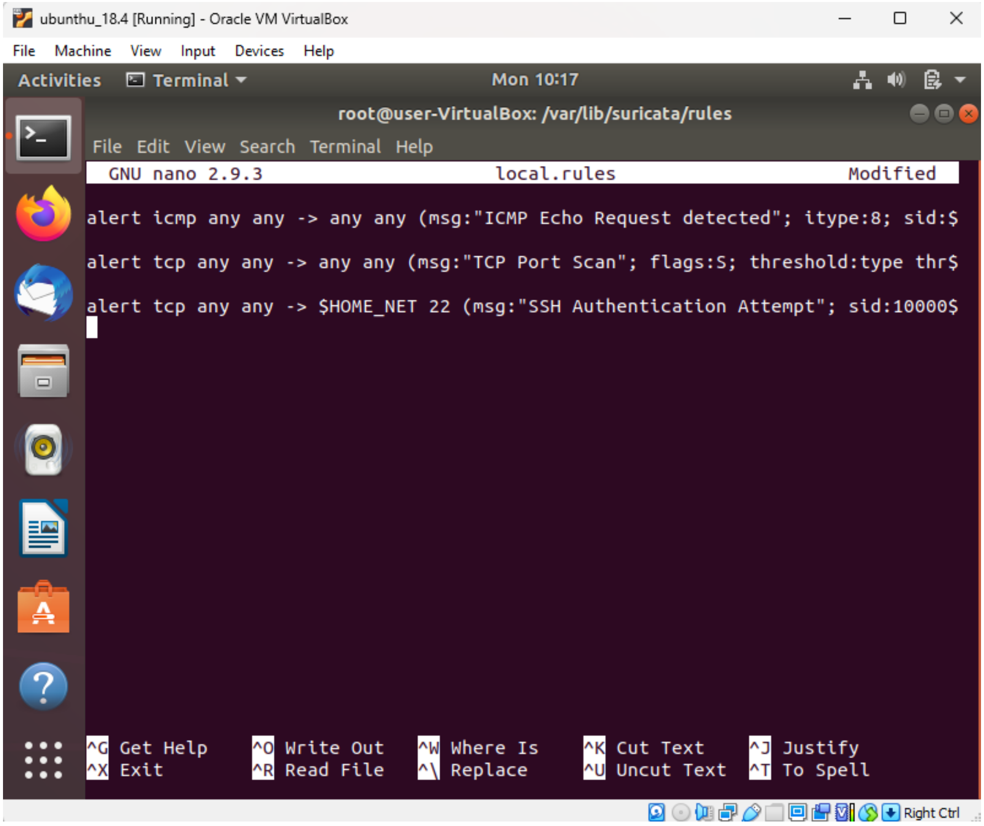
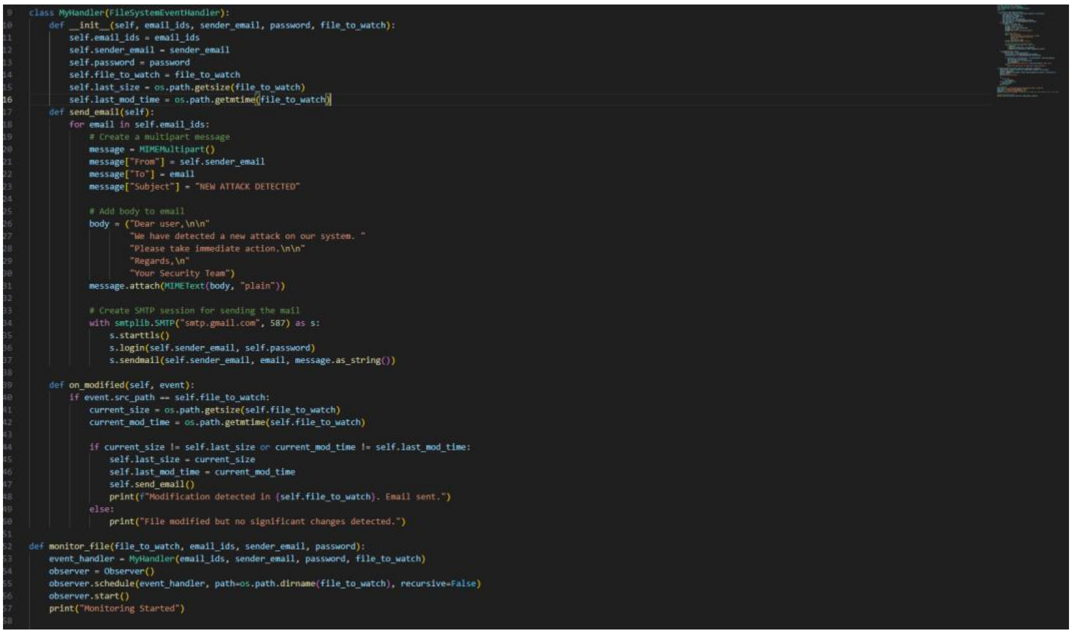


Figure (2): Custom Rules

Step 4: Generate an attack from Kali Linux to the Ubuntu system.

Figure (3): Attack Generation using Kali Linux

Step 5: Send an alert to the authority when an attack is detected.



```python
class MyHandler(FileSystemEventHandler):
    def __init__(self, email_ids, sender_email, password, file_to_watch):
        self.email_ids = email_ids
        self.sender_email = sender_email
        self.password = password
        self.file_to_watch = file_to_watch
        self.last_size = os.path.getsize(file_to_watch)
        self.last_mod_time = os.path.getmtime(file_to_watch)
    def send_email(self):
        for email in self.email_ids:
            # Create a multipart message
            message = MIMEMultipart()
            message["From"] = self.sender_email
            message["To"] = email
            message["Subject"] = "NEW ATTACK DETECTED"

            # Add body to email
            body = ("Dear user,\n\n"
                    "We have detected a new attack on our system. "
                    "Please take immediate action.\n\n"
                    "Regards,\n"
                    "Your Security Team")
            message.attach(MIMEText(body, "plain"))

            # Create SMTP session for sending the mail
            with smtplib.SMTP("smtp.gmail.com", 587) as s:
                s.starttls()
                s.login(self.sender_email, self.password)
                s.sendmail(self.sender_email, email, message.as_string())

    def on_modified(self, event):
        if event.src_path == self.file_to_watch:
            current_size = os.path.getsize(self.file_to_watch)
            current_mod_time = os.path.getmtime(self.file_to_watch)

            if current_size != self.last_size or current_mod_time != self.last_mod_time:
                self.last_size = current_size
                self.last_mod_time = current_mod_time
                self.send_email()
                print(f"Modification detected in {self.file_to_watch}. Email sent.")
            else:
                print("File modified but no significant changes detected.")

def monitor_file(file_to_watch, email_ids, sender_email, password):
    event_handler = MyHandler(email_ids, sender_email, password, file_to_watch)
    observer = Observer()
    observer.schedule(event_handler, path=os.path.dirname(file_to_watch), recursive=False)
    observer.start()
    print("Monitoring Started")
```

Figure (4): Monitor attack detection and Sending mail alerts to Authority

Step 6: Upload the details of the attack to the google drive to store.

```python
from __future__ import print_function
import os.path
import io
from google.oauth2 import service_account
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow

SCOPES = ['https://www.googleapis.com/auth/drive.file']

def main():
    creds = None
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    service = build('drive', 'v3', credentials=creds)

    # Update the file name and MIME type for a text file
    file_metadata = {'name': 'attack.txt'}
    media = MediaFileUpload('attack.txt',
                            mimetype='text/plain')
    file = service.files().create(body=file_metadata,
                                  media_body=media,
                                  fields='id').execute()
    print('File ID: %s' % file.get('id'))

if __name__ == '__main__':
    main()
```

Figure (5): Storing attack Logs in Google Drive

Step 7: Analyse and visualize the attack logs to study the attack patterns.

```python
import re
import matplotlib.pyplot as plt
from collections import defaultdict
from datetime import datetime
import pandas as pd

# Define a function to parse the log file
def parse_log_file(file_path):
    with open(file_path, 'r') as file:
        logs = file.read()

    # Regular expression to match the attack type and details
    pattern = re.compile(r'(\d{2}/\d{2}/\d{4}-\d{2}:\d{2}:\d{2}\.\d+)\s+\[\*\*\] \[(\d+):(\d+):(\d+)\] (.+?) \[\*\*\] \[Classification: (.*?)\] \[Priority: (\d+)\] \{(\w+)\} (\d+\.\d+\.\d
    matches = pattern.findall(logs)

    # Dictionary to store counts and details
    attack_counts = defaultdict(int)
    attack_details = defaultdict(list)
    logs_data = []

    for match in matches:
        timestamp = match[0]
        sid = match[1]
        gid = match[2]
        rev = match[3]
        attack_type = normalize_attack_type(match[4])
        classification = match[5]
        priority = match[6]
        protocol = match[7]
        source = match[8]
        destination = match[9]
        details = f"SID: {sid}, GID: {gid}, REV: {rev}, Classification: {classification}, Priority: {priority}, Protocol: {protocol}, Source: {source}, Destination: {destination}"

        attack_counts[attack_type] += 1
        attack_details[attack_type].append(details)

        logs_data.append({
            'timestamp': datetime.strptime(timestamp, '%m/%d/%Y-%H:%M:%S.%f'),
            'attack_type': attack_type,
            'classification': classification,
            'priority': int(priority),
            'protocol': protocol,
            'source': source,
            'destination': destination
        })

    return attack_counts, attack_details, logs_data
```

Figure (6): Analyse and visualize the attack logs - part1

```python
def normalize_attack_type(attack_type):
    if 'FTP Connection attempt' in attack_type:
        return 'FTP Connection attempt'
    elif 'ICMP Echo Request' in attack_type:
        return 'ICMP Echo Request'
    elif 'SSH Authentication Attempt' in attack_type:
        return 'SSH Authentication Attempt'
    elif 'TCP Port Scan' in attack_type:
        return 'TCP Port Scan'
    else:
        return attack_type

# Function to plot the attack counts
def plot_attack_counts(attack_counts):
    attack_types = list(attack_counts.keys())
    counts = list(attack_counts.values())

    plt.figure(figsize=(10, 6))
    plt.bar(attack_types, counts, color='skyblue')
    plt.xlabel('Attack Type')
    plt.ylabel('Count')
    plt.title('Count of Each Attack Type')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig('attack_counts_bar.png')
    plt.show()

# Function to plot attack counts as a pie chart
def plot_attack_counts_pie(attack_counts):
    attack_types = list(attack_counts.keys())
    counts = list(attack_counts.values())

    plt.figure(figsize=(10, 6))
    plt.pie(counts, labels=attack_types, autopct='%1.1f%%', startangle=140)
    plt.title('Distribution of Each Attack Type')
    plt.tight_layout()
    plt.savefig('attack_counts_pie.png')
    plt.show()
```

Figure (7): Analyse and visualize the attack logs - part2

```python
# Function to plot attack counts per protocol
def plot_attack_counts_by_protocol(logs_data):
    protocol_counts = defaultdict(int)
    for log in logs_data:
        protocol_counts[log['protocol']] += 1

    protocols = list(protocol_counts.keys())
    counts = list(protocol_counts.values())

    plt.figure(figsize=(10, 6))
    plt.bar(protocols, counts, color='lightcoral')
    plt.xlabel('Protocol')
    plt.ylabel('Count')
    plt.title('Count of Attacks by Protocol')
    plt.tight_layout()
    plt.savefig('attack_counts_by_protocol.png')
    plt.show()

# Function to plot attack counts per source IP
def plot_attack_counts_by_source(logs_data):
    source_counts = defaultdict(int)
    for log in logs_data:
        source_counts[log['source']] += 1

    source_keys = sorted(source_counts, key=source_counts.get, reverse=True)[:10]  # Top 10 sources
    source_values = [source_counts[key] for key in source_keys]

    plt.figure(figsize=(12, 6))
    plt.bar(source_keys, source_values, color='lightgreen')
    plt.xlabel('Source IP')
    plt.ylabel('Count')
    plt.title('Count of Attacks by Source IP')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig('attack_counts_by_source.png')
    plt.show()
```

Figure (8): Analyse and visualize the attack logs - part3

```python
# Function to plot attack counts per destination IP
def plot_attack_counts_by_destination(logs_data):
    destination_counts = defaultdict(int)
    for log in logs_data:
        destination_counts[log['destination']] += 1

    destination_keys = sorted(destination_counts, key=destination_counts.get, reverse=True)[:10]  # Top 10 destinations
    destination_values = [destination_counts[key] for key in destination_keys]

    plt.figure(figsize=(12, 6))
    plt.bar(destination_keys, destination_values, color='orange')
    plt.xlabel('Destination IP')
    plt.ylabel('Count')
    plt.title('Count of Attacks by Destination IP')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig('attack_counts_by_destination.png')
    plt.show()

# Function to plot attack counts per source-destination pair
def plot_attack_counts_by_source_destination(logs_data):
    source_dest_counts = defaultdict(int)
    for log in logs_data:
        source_dest_counts[(log['source'], log['destination'])] += 1

    source_dest_keys = sorted(source_dest_counts, key=source_dest_counts.get, reverse=True)[:10]  # Top 10 pairs
    source_dest_values = [source_dest_counts[key] for key in source_dest_keys]
    source_dest_labels = [f"{src} -> {dst}" for src, dst in source_dest_keys]

    plt.figure(figsize=(12, 6))
    plt.bar(source_dest_labels, source_dest_values, color='teal')
    plt.xlabel('Source -> Destination')
    plt.ylabel('Count')
    plt.title('Count of Attacks by Source-Destination Pair')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig('attack_counts_by_source_destination.png')
    plt.show()
```

Figure (9): Analyse and visualize the attack logs - part4

```python
# Function to save attack details to a text file
def save_attack_details(attack_details, file_path):
    with open(file_path, 'w') as file:
        for attack_type, details in attack_details.items():
            file.write(f"\n{attack_type}:\n")
            for detail in details:
                file.write(f"{detail}\n")
            file.write("\n" + "="*80 + "\n")

# Main function to run the analysis
def main():
    file_path = 'file.txt'  # Path to the log file
    attack_counts, attack_details, logs_data = parse_log_file(file_path)

    # Print the counts of each attack type
    print("Counts of each attack type:")
    for attack_type, count in attack_counts.items():
        print(f"{attack_type}: {count}")

    # Print the details of each attack type
    print("\nDetails of each attack type:")
    for attack_type, details in attack_details.items():
        print(f"\n{attack_type}:")
        for detail in details:
            print(f"{detail}")

    # Save attack details to a text file
    save_attack_details(attack_details, 'attack_details.txt')

    # Generate the various plots
    plot_attack_counts(attack_counts)
    plot_attack_counts_pie(attack_counts)
    plot_attack_counts_by_protocol(logs_data)
    plot_attack_counts_by_source(logs_data)
    plot_attack_counts_by_destination(logs_data)
    plot_attack_counts_by_source_destination(logs_data)


if __name__ == "__main__":
    main()
```

Figure (10): Analyse and visualize the attack logs - part5

Reference

Wikipedia Contributors (2019). *VirtualBox*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/VirtualBox [Accessed 13 May 2024].