Java Networking Project

Network Intrusion Detection System (NIDS)

Anuoluwapo Oguntayo

Bachelor of Software & Electronic Engineering

Galway-Mayo Institute of Technology

2019/2020

Intrusion Detection System

Anuoluwapo Oguntayo
Beng (Hons) Software & Electronic Engineering

Description

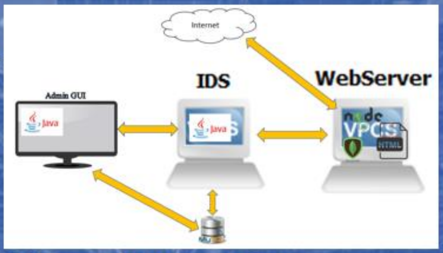
An intrusion detection system (IDS) is software that monitors a system network for malicious activity.
It then notifies the system administrator while collecting data on such events. My IDS Identifies network vulnerabilities in a host server by reading inbound and outbound traffic, compares the packets to sample malicious packet before basing threat level on similarities.

Technologies

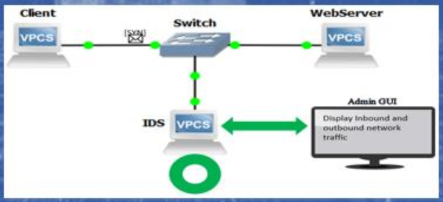
- **Java:** Is used to develop the IDS server, GUI and MySQL communication.
- **MySQL:** For storing and retrieve packet info.
- **JnetPcap:** A java library for analysing network interface.
- **NodeJs:** To host a vulnerable web server that my IDS will be protecting.
- **HTML:** To create a web page.

Can Detect

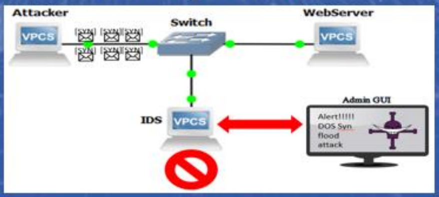
- DOS-ATTACKS
- Cross site scripting (XSS)
- Session hijacking
- Brute Force

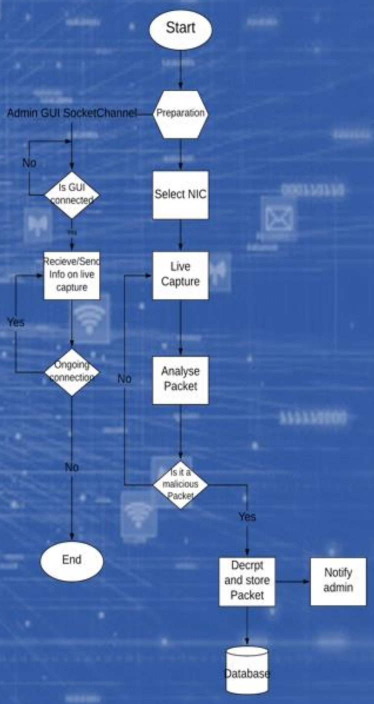


Normal Operation



Intrusion Operation





https://github.com/Cosmos-tec/NIDS_Project

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

____

Acknowledgements

I would like to acknowledge my supervisor for the support throughout this project.

Acronyms

- **NIDS → Network Intrusion Detection System.**
- **IDS → Intrusion Detection System.**
- **GUI → Graphical User Interface.**
- **GNS → Graphical Network Simulator.**
- **VM → Virtual Machines.**
- **OS → Operating System.**
- **NIC → Network Interface Card.**
- **RDBMS → Relational DataBase Management System.**
- **SQL → Structured Query Language.**
- **API → Application Programming Interface.**
- **JNI → Java Native Interface.**
- **DDOS → Distributed Denial Of Service.**
- **MITM → Man In The Middle**
- **TCP → Transmission Control Protocol**

Table of Contents

1 Summary

My project, the NIDS is a java continues integration project. The idea for this project is to be able to detect networking attacks on a hosted server. My NIDS is a software that monitors a system network traffic by analyzing inbound and outbound packets on a given network interface for any suspicious activity, it identifies attack based on abnormal communication in a computer system from port scanning to packet headers, this packets are also displayed on a GUI that shows an alert when the NIDS detects a malicious packet. The goal for this project where to detect a commonly used DDOS-attack, web server vulnerability such as XSS-cross site scripting and session hijacking. When my NIDS detect an attack, it registers the attack info into a database that can be access for through the GUI. The information being stored in the database (MySQL) consist of attack type, src address, port targeted, time of the attack and packet headers. The main function of the GUI is to read packet info and filter out protocols, it can also configure the NIDS alert function, security group and policies.

I approached this project in an agile manner as I listed project requirement with my supervisor, design the operation flow of the NIDS and developed the project with each new feature added being reviewed/tested. Github was the true source for version control of my NIDS.

I choose this project because I wanted to improve my knowledge on java networking and java swing worker while building my cyber security understanding. Overall, I accomplished the main goal of the project which was to be able to

- Detect networking attacks.
- Triggering an alert for attacks.
- Storing information on set attacks.
- Having a GUI to view and filter packet info.
- GUI view database.

2 Introduction

This report contains all the necessary information gather to build and completed this project. This project is a java networking project between the NIDS and the Admin GUI. Throughout this report I will put an efficacy on the tool used to develop the NIDS server and the Admin GUI. The NIDS server consist of JnetPcap library for parsing network information, TCP socket channel between the NIDS and Admin GUI, and prepared statement for communicating with MySQL database. The Admin GUI uses TCP socket channel for access to the NIDS server.

JnetPcap is a java wrapper library for “LibPcap” and “WinPcap” that has a full API translation with packet buffers delivered with no copies. It has the methods and classes that can provide me with deep packet inspection (DPI), with several protocol analysis and life/offline packet capture capabilities. This library is only used to capture packets for my NIDS to analyse. MySQL is a lightweight database that allows for scalability and management for application data. It also comes with a java library for communicating with database cluster.

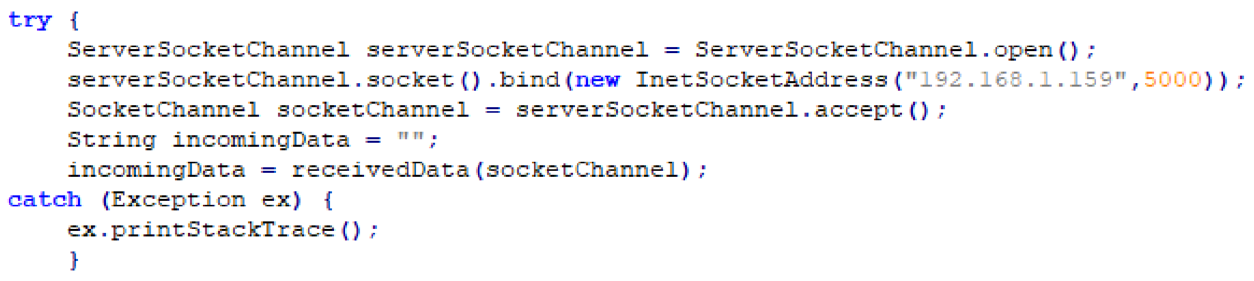
To demonstrate the processing during a networking attack I used GNS 3 to simulate a network and virtual box for managing my virtual machine. This report will also cover how and why I choose this software’s to integrated with my NIDS project.

3 Java Networking Overview

Java networking is the term used to describe the concept of connecting two or more computing devices together in other to share data, this is achieved with java socket programming. I use socket channel to send packet information from the NIDS server to the Admin GUI to display which allows the Admin GUI to connect to the NIDS server from any computer if it knows it IP address and listening port.

The Admin GUI binds itself to a given IP to begin listen at an available port which is in relation to the computers network interface for TCP/IP communication, connection to NIDS s is needed for the Admin GUI to begin it's display operation. Figure 3-1 is the java networking code for the Admin GUI.

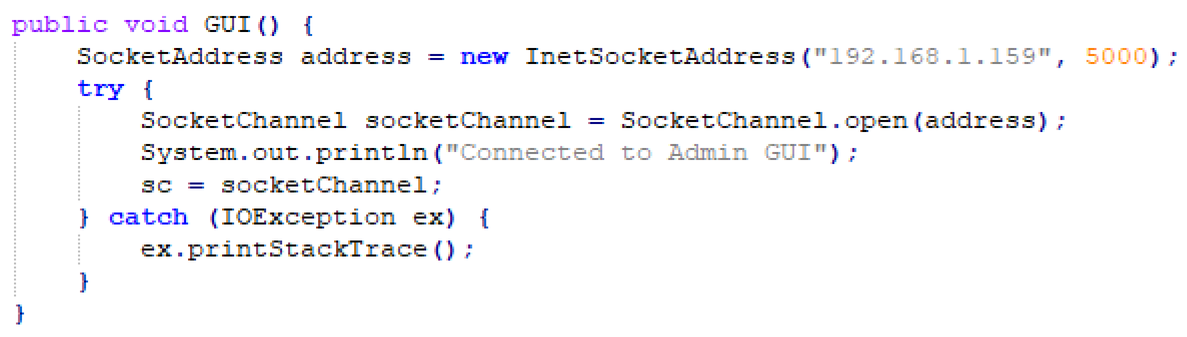
Figure 3-1 Java Networking



```
try {
    ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
    serverSocketChannel.socket().bind(new InetSocketAddress("192.168.1.159",5000));
    SocketChannel socketChannel = serverSocketChannel.accept();
    String incomingData = "";
    incomingData = receivedData(socketChannel);
catch (Exception ex) {
    ex.printStackTrace();
    }
```

The NIDS scans it interfaces in promiscuous mode and has a separate thread to check for connection to the Admin GUI in other to begin sending data to the GUI. Unlike the Admin GUI the NIDS opens a socket channel to the GUI. Figure 3-2 is the java networking code for the NIDS.

Figure 3-2 Java Networking

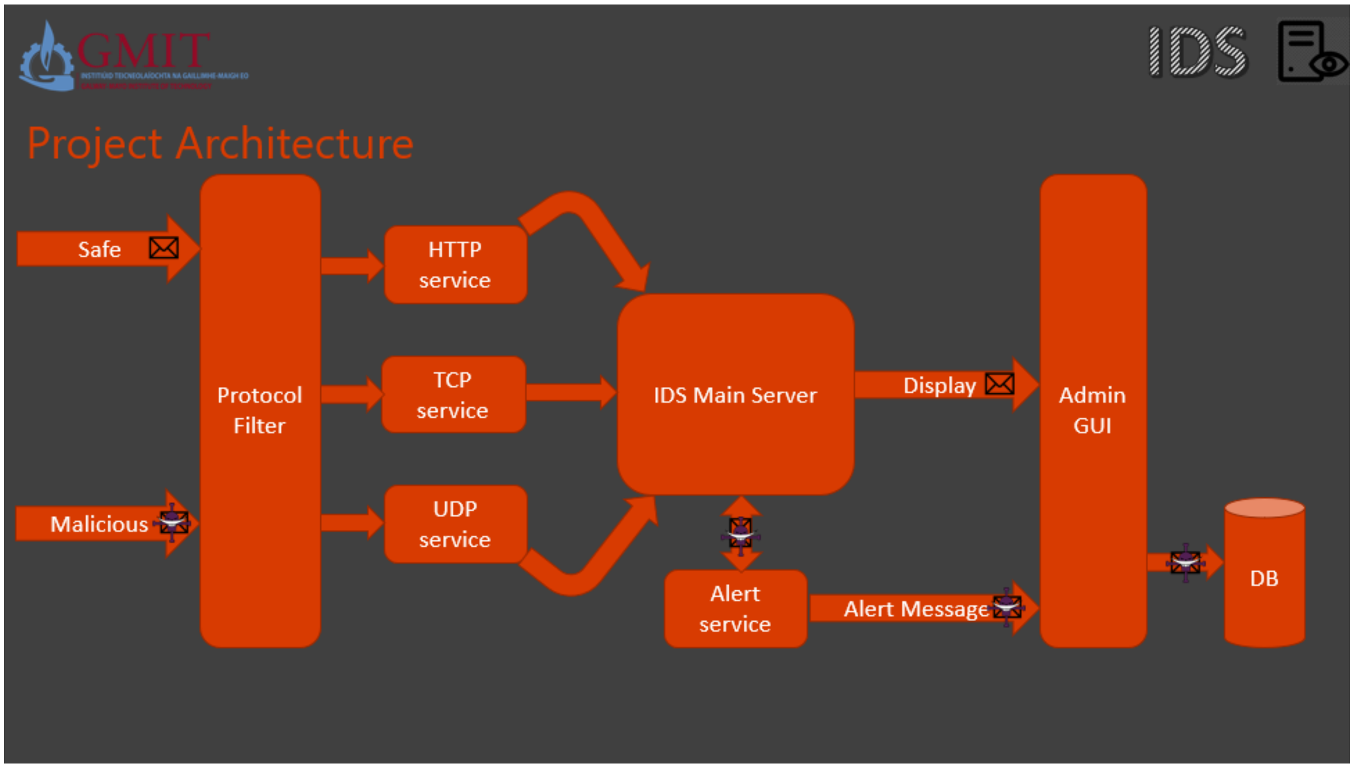


```
public void GUI() {
    SocketAddress address = new InetSocketAddress("192.168.1.159", 5000);
    try {
        SocketChannel socketChannel = SocketChannel.open(address);
        System.out.println("Connected to Admin GUI");
        sc = socketChannel;
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

4 Project Architecture

My development tool for the NIDS is Java Development Kit (JDK) while the IDE's I used are Intellij and NetBeans. Each service in the architecture are separate thread that analyse packet based on the protocol.

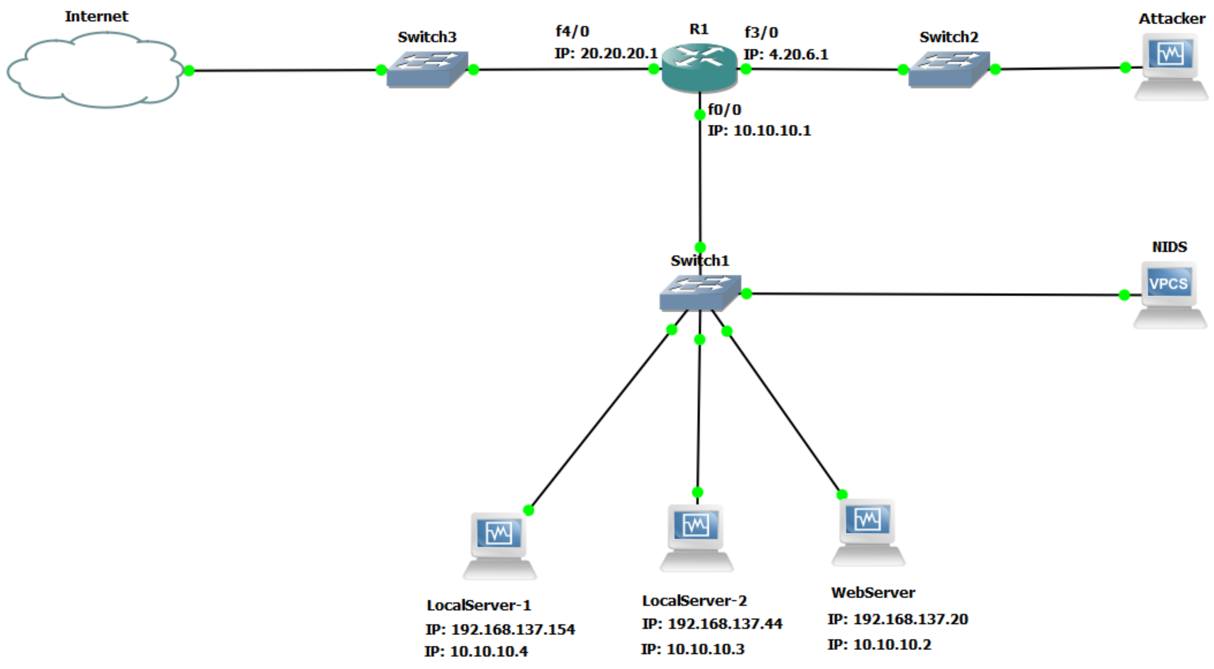
Figure 4-1 Architecture Diagram



5 GNS 3

Graphical Network Simulator (GNS 3) is a free open source software that allows emulation of complex networks. It can emulate cisco internetwork operating systems, cisco IOS devices and virtual computer which allow me to run virtual PC. It helps with visualising a network topology for better understanding on where an attack can take place. GNS 3 provided a virtual environment to test and showcase my NIDS before deployment.

Figure 5-1 Network Topology

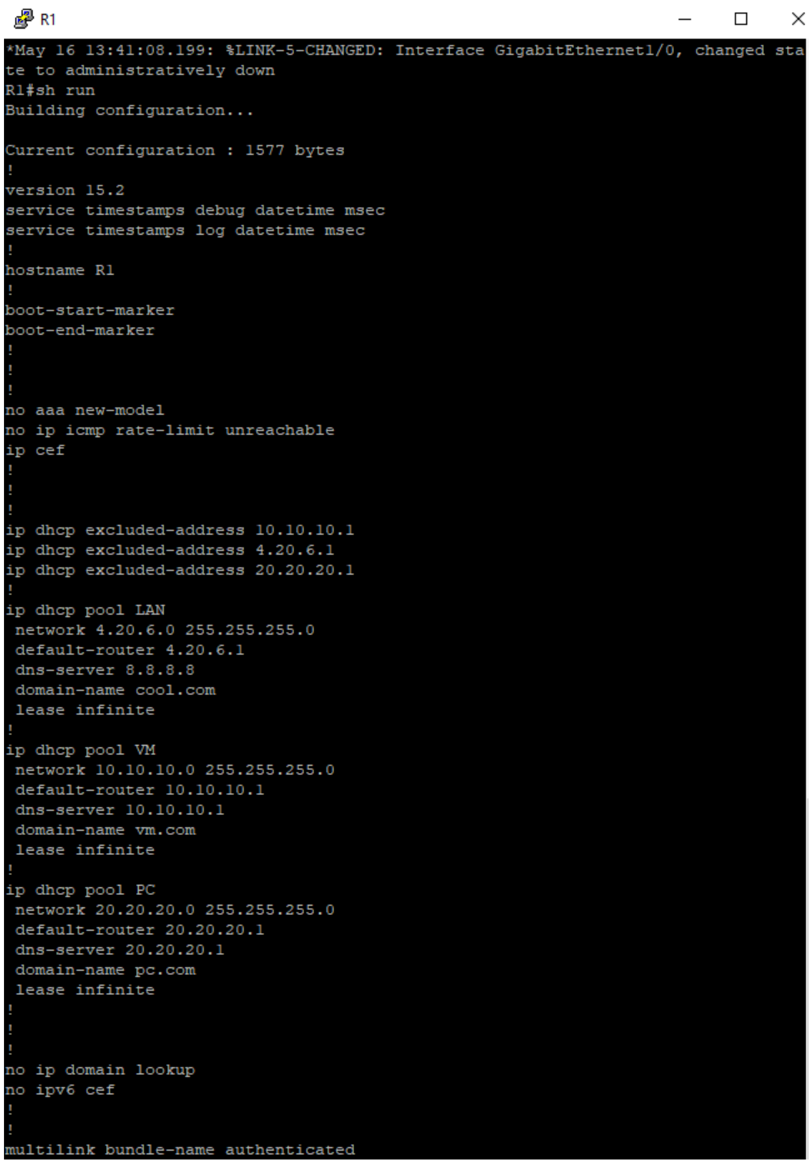


Each PC in the figure 5-1 are a virtual machine (Attacker, Local Server 1, Local Server 2 & Web Server) that are running on virtual box. The router is configuring for DHCP leasing server and interface f4/0 is connected to my computer. The NIDS main Server is in the same subnet as the local servers which are running a subcomponent called IDS (Intrusion Detection System) that mange a single host network traffic. Each server IDS is filter for appropriate attack detection. The NIDS server is not reachable for any other computers except its neighbours running a component of the NIDS, this is to prevent attack from knowing the NIDS location.

5.1 Router Configuration

R1 is configure for DHCP lease server and emulates a real router operation figure 5.1-1 highlights a sample network configuration that is needed for the NIDS.

Figure 5.1-2 R1 Configuration



```
*May 16 13:41:08.199: %LINK-5-CHANGED: Interface GigabitEthernet1/0, changed sta
te to administratively down
R1#sh run
Building configuration...

Current configuration : 1577 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R1
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
!
!
ip dhcp excluded-address 10.10.10.1
ip dhcp excluded-address 4.20.6.1
ip dhcp excluded-address 20.20.20.1
!
ip dhcp pool LAN
 network 4.20.6.0 255.255.255.0
 default-router 4.20.6.1
 dns-server 8.8.8.8
 domain-name cool.com
 lease infinite
!
ip dhcp pool VM
 network 10.10.10.0 255.255.255.0
 default-router 10.10.10.1
 dns-server 10.10.10.1
 domain-name vm.com
 lease infinite
!
ip dhcp pool PC
 network 20.20.20.0 255.255.255.0
 default-router 20.20.20.1
 dns-server 20.20.20.1
 domain-name pc.com
 lease infinite
!
!
!
no ip domain lookup
no ipv6 cef
!
!
multilink bundle-name authenticated
```

Even though R1 is just a simulator for cisco router, it stills performs the necessary operation for me to capture live traffic that would otherwise be the same type of packet being sent to a real computer. This configuration provided an identical networking scenario that can be hosted on a live network, so it makes a perfect testing ground/building for the NIDS.

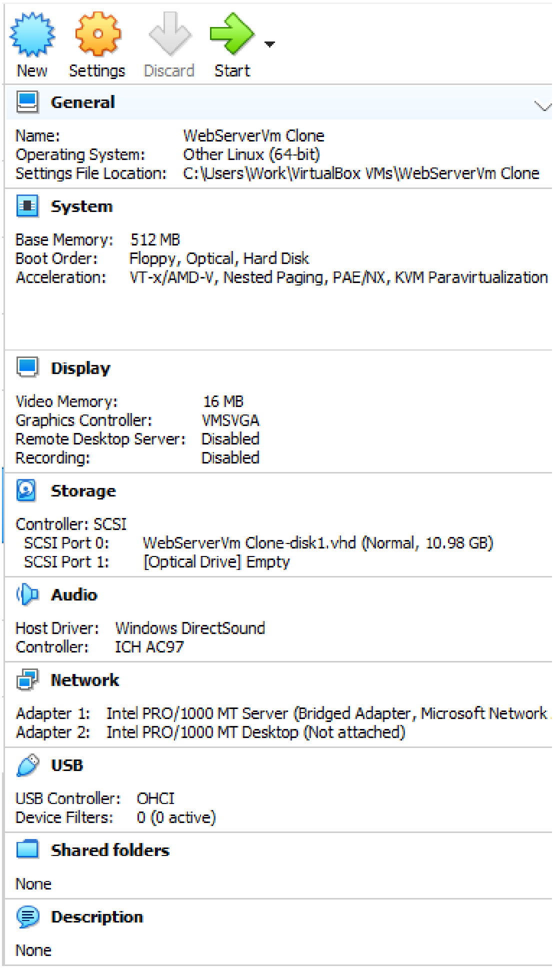
6 Virtual Box

Virtual box is an open source hosted hypervisor for x86 and AMD64/Intel64 virtualization. It is developed by Oracle Corporation. Virtual box can emulate multiple guestOSes under a single host operating system, which gives users the ability to configure and run each VM (virtual Machine) under a choice of software-based virtualization or hardware assisted virtualization if the host is supported. I use virtual box to run a centOS 7 for a model server running my NIDS java application. The vms are configure with 2 possible NIC for live capture.

6.1 VM Configuration

Figure 6.1-1 is the configuration of the vm needed to run my NIDS in a virtual environment. I kept the vm at a low spec as the NIDS is a lightweight software and does not need much resources to run.

Figure 6.1-1 VM Setup



General
Name: WebServerVm Clone
Operating System: Other Linux (64-bit)
Settings File Location: C:\Users\Work\VirtualBox VMs\WebServerVm Clone

System
Base Memory: 512 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: SCSI
SCSI Port 0: WebServerVm Clone-disk1.vhd (Normal, 10.98 GB)
SCSI Port 1: [Optical Drive] Empty

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Server (Bridged Adapter, Microsoft Network
Adapter 2: Intel PRO/1000 MT Desktop (Not attached)

USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Description
None

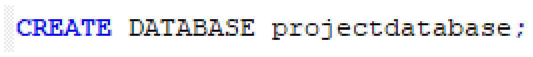
The network configuration is important as the NIDS needs a know the route to the host of it is subcomponent IDS. As specified in the GNS 3 section this are the vm's configuration that is placed in the network topology.

7 MySQL

MySQL is a database management system that stores a collection of data. It has a distinct APIs for creating, accessing, managing, searching and replicating the data it holds. It can also hold large hash tables in memory. I choose this as my data centre because it supports java and many OS's. The NIDS is integrated with MySQL connector and will require it to store and hold packet information during an alert scenario. It's an open source RDBMS.

7.1 MySQL Database Configuration

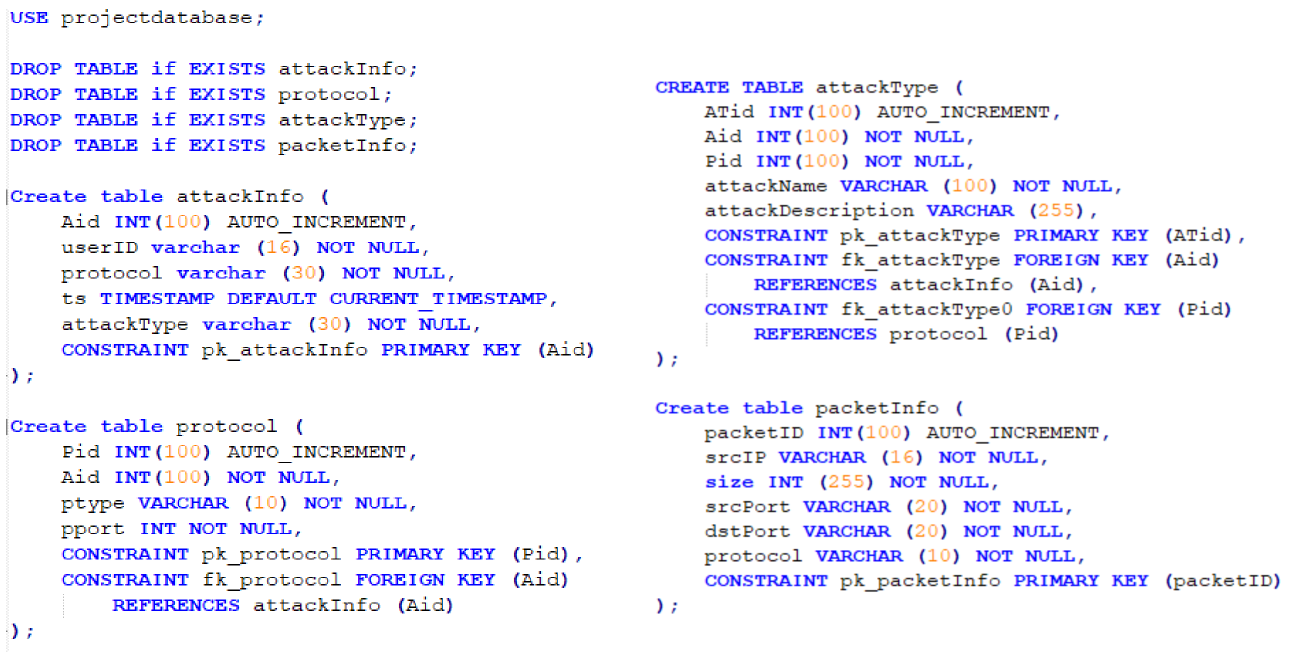
MySQL has its own query language for creating and accessing database, figure 7.1-1 shows the command used to create a database for NIDS.



```
CREATE DATABASE projectdatabase;
```

The "CREATE DATABASE" is the command use to create the NIDS database with the name "projectdatabase".

Figure 7.1-1 MySQL DB



```
USE projectdatabase;

DROP TABLE if EXISTS attackInfo;
DROP TABLE if EXISTS protocol;
DROP TABLE if EXISTS attackType;
DROP TABLE if EXISTS packetInfo;

Create table attackInfo (
    Aid INT(100) AUTO_INCREMENT,
    userID varchar (16) NOT NULL,
    protocol varchar (30) NOT NULL,
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    attackType varchar (30) NOT NULL,
    CONSTRAINT pk_attackInfo PRIMARY KEY (Aid)
);

Create table protocol (
    Pid INT(100) AUTO_INCREMENT,
    Aid INT(100) NOT NULL,
    ptype VARCHAR (10) NOT NULL,
    pport INT NOT NULL,
    CONSTRAINT pk_protocol PRIMARY KEY (Pid),
    CONSTRAINT fk_protocol FOREIGN KEY (Aid)
        REFERENCES attackInfo (Aid)
);

CREATE TABLE attackType (
    ATid INT(100) AUTO_INCREMENT,
    Aid INT(100) NOT NULL,
    Pid INT(100) NOT NULL,
    attackName VARCHAR (100) NOT NULL,
    attackDescription VARCHAR (255),
    CONSTRAINT pk_attackType PRIMARY KEY (ATid),
    CONSTRAINT fk_attackType FOREIGN KEY (Aid)
        REFERENCES attackInfo (Aid),
    CONSTRAINT fk_attackType0 FOREIGN KEY (Pid)
        REFERENCES protocol (Pid)
);

Create table packetInfo (
    packetID INT(100) AUTO_INCREMENT,
    srcIP VARCHAR (16) NOT NULL,
    size INT (255) NOT NULL,
    srcPort VARCHAR (20) NOT NULL,
    dstPort VARCHAR (20) NOT NULL,
    protocol VARCHAR (10) NOT NULL,
    CONSTRAINT pk_packetInfo PRIMARY KEY (packetID)
);
```

8 JnetPcap

JnetPcap is a java wrapper around LibPcap and WinPcap native libraries found on various unix and windows platforms. It exposes the functionality as a java programming interface (API) hence I choose to use it in my project as a live capture networking traffic.

Features include but not limited to

- LibPcap programming Model.
- Easy to learn API.
- Native in-memory buffer is wrapped inside a java.nio.ByteBuffer object.
- Low level and efficient JNI implementation.

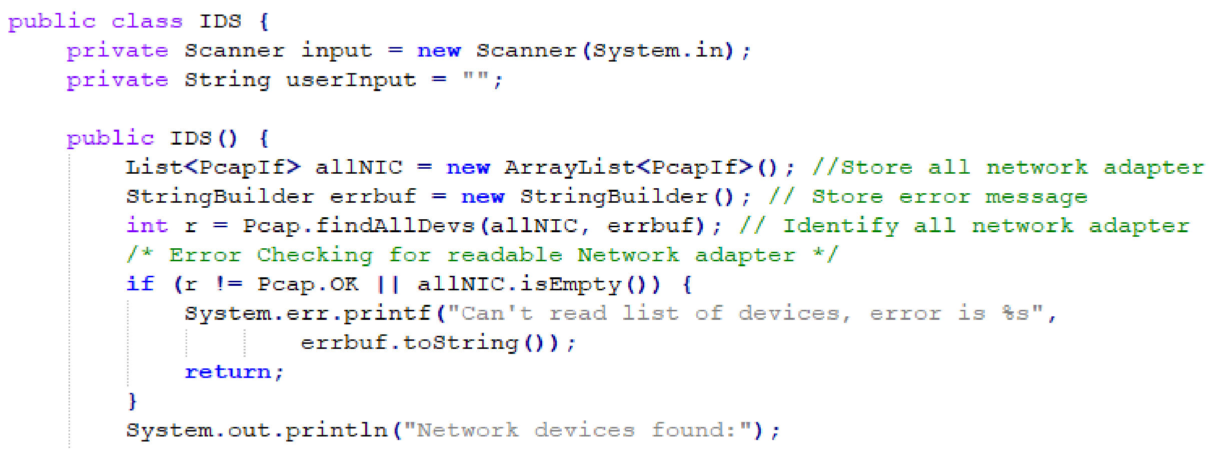
JnetPcap is an open source library the NIDS uses to begin live capture on network interfaces for it to then analyse and filter out malicious packet from legitimate packet.

If your project has motors, put this section in to explain. Describe both hardware and software in subsections. Include a subsection on your motor's power requirements.

8.1 JnetPcap List Interface Code

Figure 8.1-1 is how I list the available network interface with jnetpcap buy storing all network adapter in a list and checking that the list is not empty in other to begin live capture. "Pcap.findAllDevs" returns an int for every interface identified on the host system. "Pcap.OK" returns an int 0 or 1 to confirm that an interface is capturable.

Figure 8.1-1 Available Interfaces.



```
public class IDS {
    private Scanner input = new Scanner(System.in);
    private String userInput = "";

    public IDS() {
        List<PcapIf> allNIC = new ArrayList<PcapIf>(); //Store all network adapter
        StringBuilder errbuf = new StringBuilder(); // Store error message
        int r = Pcap.findAllDevs(allNIC, errbuf); // Identify all network adapter
        /* Error Checking for readable Network adapter */
        if (r != Pcap.OK || allNIC.isEmpty()) {
            System.err.printf("Can't read list of devices, error is %s",
                    errbuf.toString());
            return;
        }
        System.out.println("Network devices found:");
```

Figure 8.1-2 list information on every device that are available on the system.

Figure 8.1-2 Interfaces description.

```
int i = 0;
for (PcapIf device : allNIC) {
    String description = (device.getDescription() != null) ? device
            .getDescription() : "No description available";
    System.out.printf("#%d: %s [%s]\n", i++, device.getName(),
            description);
}
```

8.2 JnetPcap Capture Code

If all the available network interfaces are ok, then I use the code in fig 8.2-1 to begin capture on a chosen interface.

Figure 8.2-2 Capture Interfaces.

```
int snaplen = 64 * 1024; // Capture all packets, no trucation
int flags = Pcap.MODE_PROMISCUOUS; // capture all packets
int timeout = 10 * 1000; // 10 seconds in millis
Pcap pcap = Pcap.openLive(device.getName(), snaplen, flags, timeout, errbuf);
if (pcap == null) {
    System.err.printf("Error while opening device for capture: "
            + errbuf.toString());
    return;
}
```

To begin capture I specified that the snap length (snaplen) to be the maximum so all packets can be capture for each frame. I set the mode to promiscuous so all network traffic to be sent to the NIDS as a central processing unit. The line “Pcap pcap = Pcap.openlive…” is what then begins the capture once all requirement has been met.

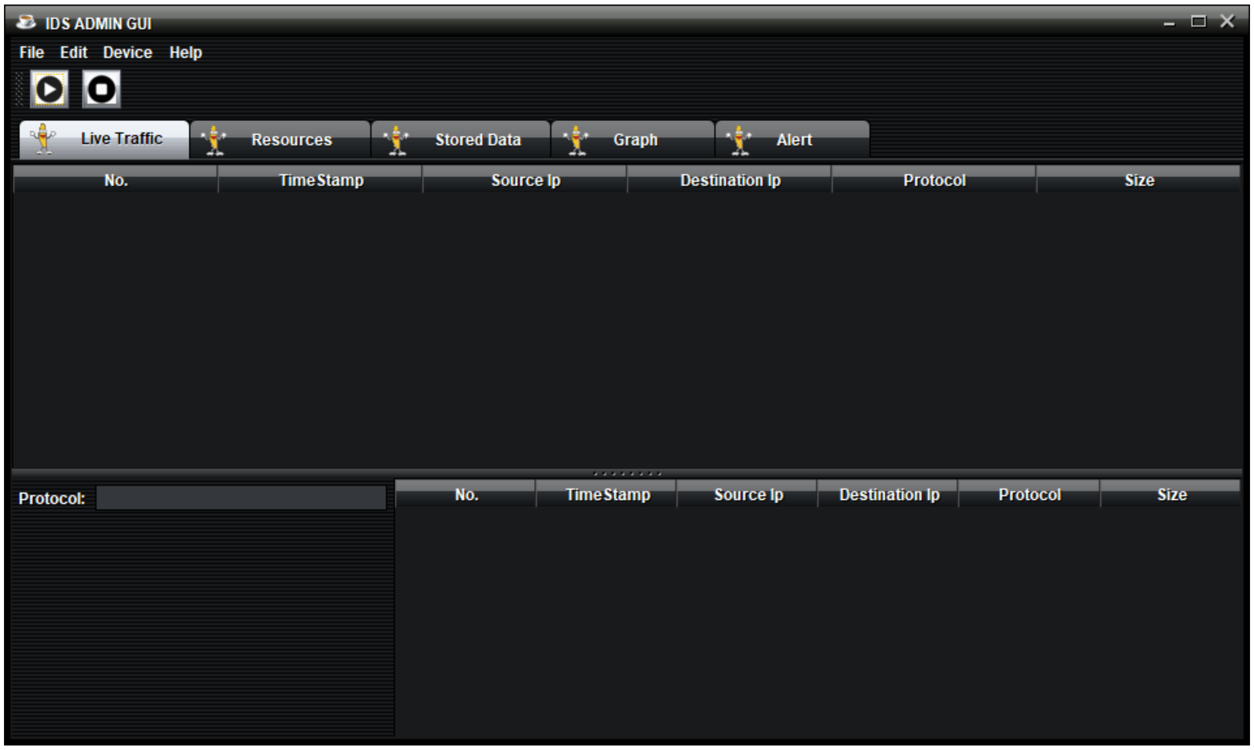
9 Admin GUI

The Admin GUI is developed with java swing worker. It us a jtatoo noir look and feel. The purpose of the GUI is to have something that can display traffic information and show stored packet info in the database. This can also display an alert when the NIDS has detected a malicious packet. Communication or data transfer between the NIDS and the Admin GUI is socket channel.

Features:

- Live capture info of inbound and outbound packets.
- Start and Stop.
- Filter Protocol.
- Previous Alert information.
- Delete info from the database.
- IDS modify configuration.

Figure 9-1 Admin GUI.



9.1 GUI Code

Figure 9.1-1 show the Mainframe class which inherits JFrame to build the GUI.

Figure 9.1-2 Mainframe

```
public class MainFrame extends JFrame {
```

The main frame is split into top and bottom panel with figure 9.1-2 for menu bar, figure 9.1-3 for play action and figure 9.1-4 for each tabbed pane.

Figure 9.1-2 Menu bar

```
public JMenuBar createMenu() {
    JMenuBar menuBar = new JMenuBar();
    JMenu file,edit,device,help;

    file = new JMenu("File");
    edit = new JMenu("Edit");
    device = new JMenu("Device");
    help = new JMenu("Help");

    menuBar.add(file);
    menuBar.add(edit);
    menuBar.add(device);
    menuBar.add(help);

    return menuBar;
}
```

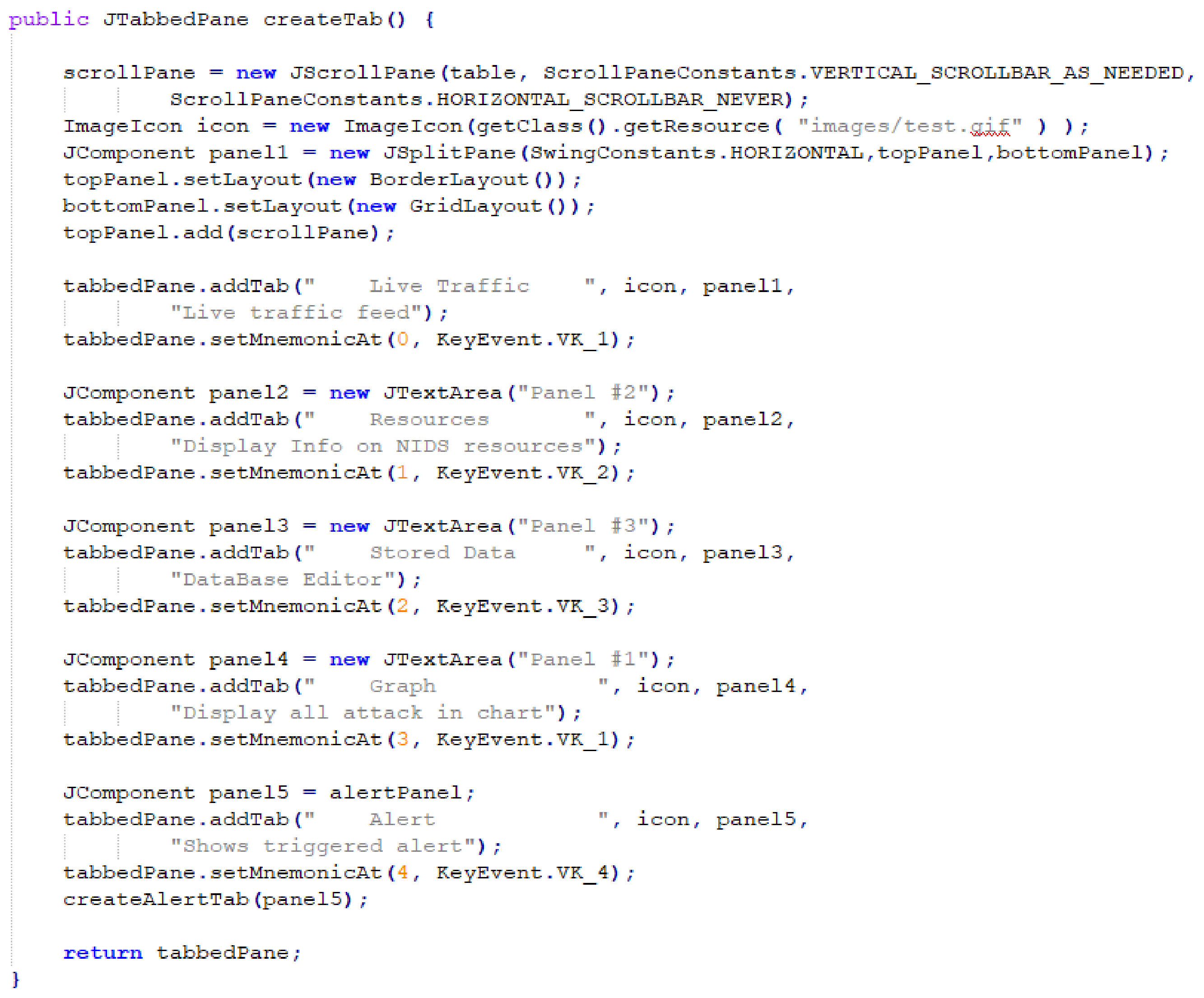
Creating and adding each menu option to a menu bar so it can be added to the GUI main window.

Figure 9.1-2 Action button

```
private class playAction extends AbstractAction {

    // set up action's name, icon, descriptions and mnemonic
    public playAction()
    {
        putValue( NAME, "Play" );
        putValue( SMALL_ICON, new ImageIcon(
                getClass().getResource( "images/play.png" ) ) );
        putValue( SHORT_DESCRIPTION, "Play" );
        putValue( LONG_DESCRIPTION,
                "Start/Resume live network traffic Capture " );
        putValue( MNEMONIC_KEY, new Integer( 'P' ) );
    }

    // display window in which user can input entry
    public void actionPerformed( ActionEvent e )
    {
        //Start live Capture
        sendMessage(socketChannel,"Play");
    }
}
```

Play action is an abstract button with an abstraction that begins the display of the incoming packet data to the GUI by sending a play message to the NIDS, when pressed.

Figure 9.1-3 TabPane



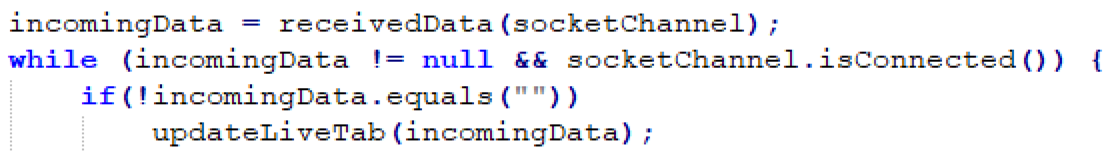
```
public JTabbedPane createTab() {

    scrollPane = new JScrollPane(table, ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
    ImageIcon icon = new ImageIcon(getClass().getResource( "images/test.gif" ) );
    JComponent panel1 = new JSplitPane(SwingConstants.HORIZONTAL,topPanel,bottomPanel);
    topPanel.setLayout(new BorderLayout());
    bottomPanel.setLayout(new GridLayout());
    topPanel.add(scrollPane);

    tabbedPane.addTab("    Live Traffic    ", icon, panel1,
            "Live traffic feed");
    tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

    JComponent panel2 = new JTextArea("Panel #2");
    tabbedPane.addTab("    Resources       ", icon, panel2,
            "Display Info on NIDS resources");
    tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);

    JComponent panel3 = new JTextArea("Panel #3");
    tabbedPane.addTab("    Stored Data     ", icon, panel3,
            "DataBase Editor");
    tabbedPane.setMnemonicAt(2, KeyEvent.VK_3);

    JComponent panel4 = new JTextArea("Panel #1");
    tabbedPane.addTab("    Graph            ", icon, panel4,
            "Display all attack in chart");
    tabbedPane.setMnemonicAt(3, KeyEvent.VK_1);

    JComponent panel5 = alertPanel;
    tabbedPane.addTab("    Alert            ", icon, panel5,
            "Shows triggered alert");
    tabbedPane.setMnemonicAt(4, KeyEvent.VK_4);
    createAlertTab(panel5);

    return tabbedPane;
}
```

Live Traffic and alert Tab are set to a scrollable pane as this frame can contain a lot of information due to capture all incoming network traffic configuration.

9.2 Live Traffic Code

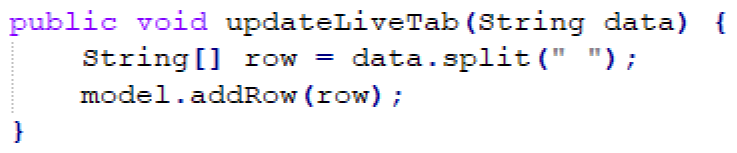
Live traffic tab is the incoming packet info from the NIDS which has converted the byte data to a string object for the GUI to use. When connection has been made with the Admin GUI and NIDS using socket channel as seen in figure 3-1 & 3-2 java networking, the admin GUI start's it is live display of data being captured by the NIDS. Figure 9.2-1 & 9.2-1 is how I update the live traffic tab with incoming data.

Figure 9.2-1 Incoming Data



```java
incomingData = receivedData(socketChannel);
while (incomingData != null && socketChannel.isConnected()) {
    if(!incomingData.equals(""))
        updateLiveTab(incomingData);
```

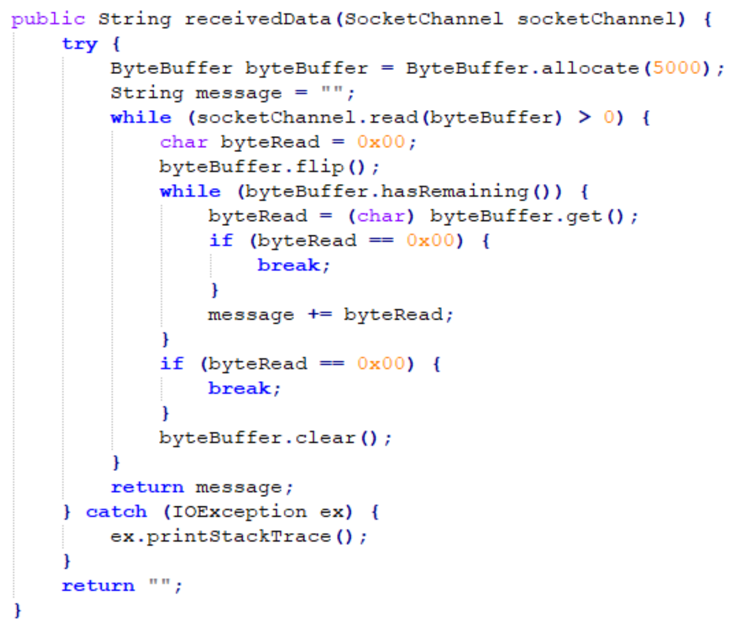
Figure 9.2-2 Update Live Tab



```java
public void updateLiveTab(String data) {
    String[] row = data.split(" ");
    model.addRow(row);
}
```

Data is displayed in a table format so every time a new data is receive it creates a new table of the received data. Figure 9.2-3 is how each data being received is processed byte by byte.

Figure 9.2-3 NIDS to GUI

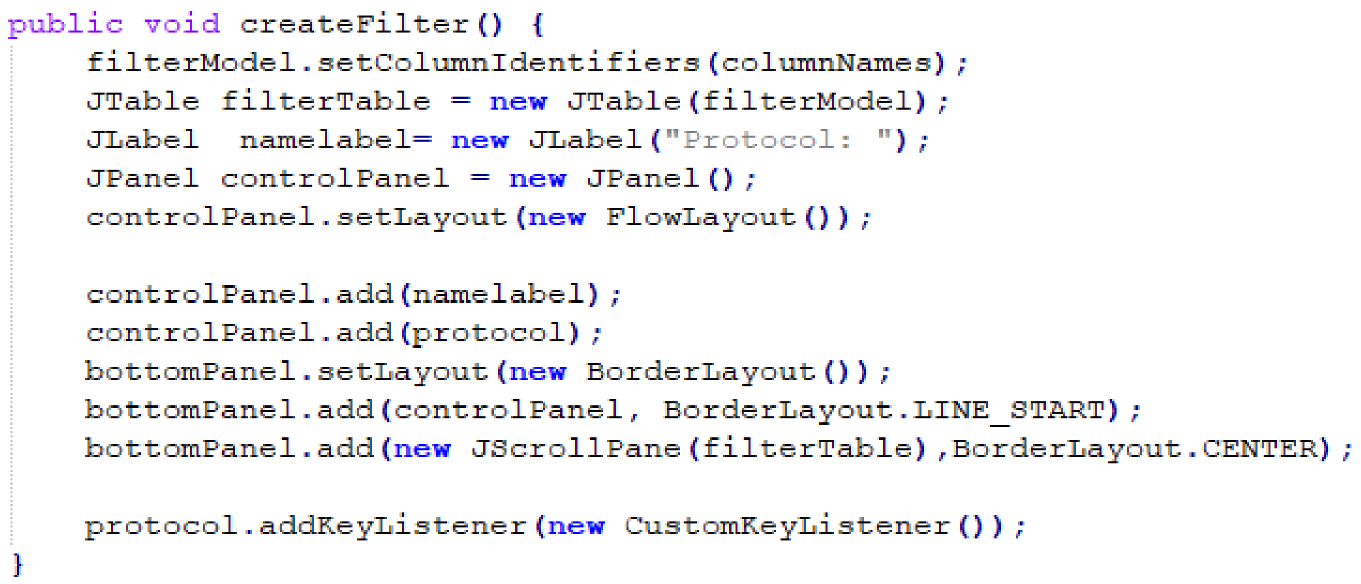


```java
public String receivedData(SocketChannel socketChannel) {
    try {
        ByteBuffer byteBuffer = ByteBuffer.allocate(5000);
        String message = "";
        while (socketChannel.read(byteBuffer) > 0) {
            char byteRead = 0x00;
            byteBuffer.flip();
            while (byteBuffer.hasRemaining()) {
                byteRead = (char) byteBuffer.get();
                if (byteRead == 0x00) {
                    break;
                }
                message += byteRead;
            }
            if (byteRead == 0x00) {
                break;
            }
            byteBuffer.clear();
        }
        return message;
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return "";
}
```

9.3 Filter Protocol Code

Filter section on the GUI is placed in the bottom panel of the window. The table and modal are set to the same as the live traffic. Layout is updated to a boarder and can use a listener for the protocol input.

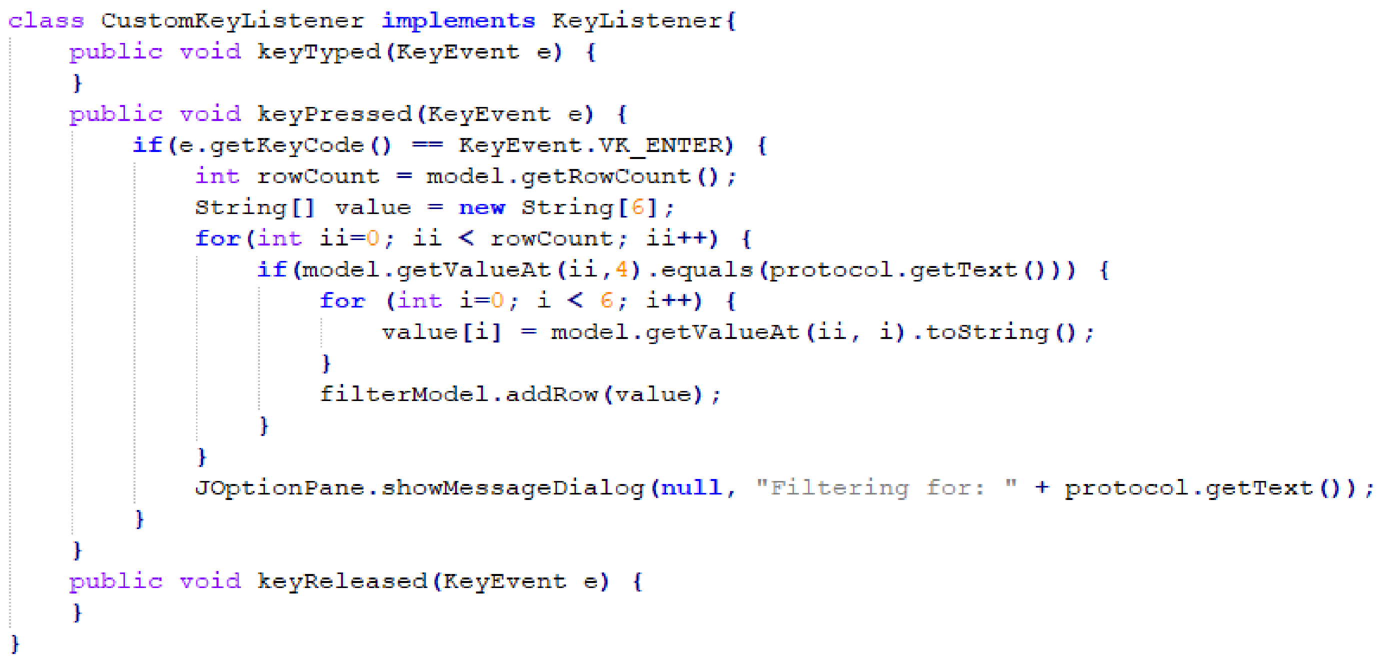
Figure 9.3-1 Filter Section



```
public void createFilter() {
    filterModel.setColumnIdentifiers(columnNames);
    JTable filterTable = new JTable(filterModel);
    JLabel  namelabel= new JLabel("Protocol: ");
    JPanel controlPanel = new JPanel();
    controlPanel.setLayout(new FlowLayout());

    controlPanel.add(namelabel);
    controlPanel.add(protocol);
    bottomPanel.setLayout(new BorderLayout());
    bottomPanel.add(controlPanel, BorderLayout.LINE_START);
    bottomPanel.add(new JScrollPane(filterTable),BorderLayout.CENTER);

    protocol.addKeyListener(new CustomKeyListener());
}
```

During a filter operation the key input text field is used when the enter key is pressed. The number of rows in the Live tab is counted and stored in an int (rowCount) then a loop for the number of row count to find each row in the table that contains the specified protocol, if the value at the protocol column (4) matches the value of the filter input then it saves that row in a string array called value which is then displayed in the filter tab.

Figure 9.3-2 Filter Operation



```
class CustomKeyListener implements KeyListener{
    public void keyTyped(KeyEvent e) {
    }
    public void keyPressed(KeyEvent e) {
        if(e.getKeyCode() == KeyEvent.VK_ENTER) {
            int rowCount = model.getRowCount();
            String[] value = new String[6];
            for(int ii=0; ii < rowCount; ii++) {
                if(model.getValueAt(ii,4).equals(protocol.getText())) {
                    for (int i=0; i < 6; i++) {
                        value[i] = model.getValueAt(ii, i).toString();
                    }
                    filterModel.addRow(value);
                }
            }
            JOptionPane.showMessageDialog(null, "Filtering for: " + protocol.getText());
        }
    }
    public void keyReleased(KeyEvent e) {
    }
}
```

9.4 AlertTab Code

Alert tab displays data in the database that the NIDS has registered when an alert is triggered. This will also update its self-live when an alert is triggered. It updates using the sql query class for fetching data. Alert tab is created in figure 9.4-1 and then the runs the “alertTab” class on a separate thread to handle the sql query result.

Figure 9.4-1 Create Alert Tab

```java
void createAlertTab(JComponent aTab) {
    String[] columnNames = {"Attack Type", "TimeStamp", "Src Ip", "Dst Ip","Info"};
    model1.setColumnIdentifiers(columnNames);
    JTable table = new JTable(model1);
    alertPanel.setLayout(new GridLayout());
    alertPanel.add(new JScrollPane(table, ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER));
    textPane.setText("HexDump of rawPacket\n");
    alertPanel.add(new JScrollPane(textPane,ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER));
    try {
        SqlQuery sql = new SqlQuery();
        ResultSet res = sql.findAlertInfo();
        if(res != null)
            new Thread(new alertTab(model1,textPane));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 9.4-2 Alert Tab

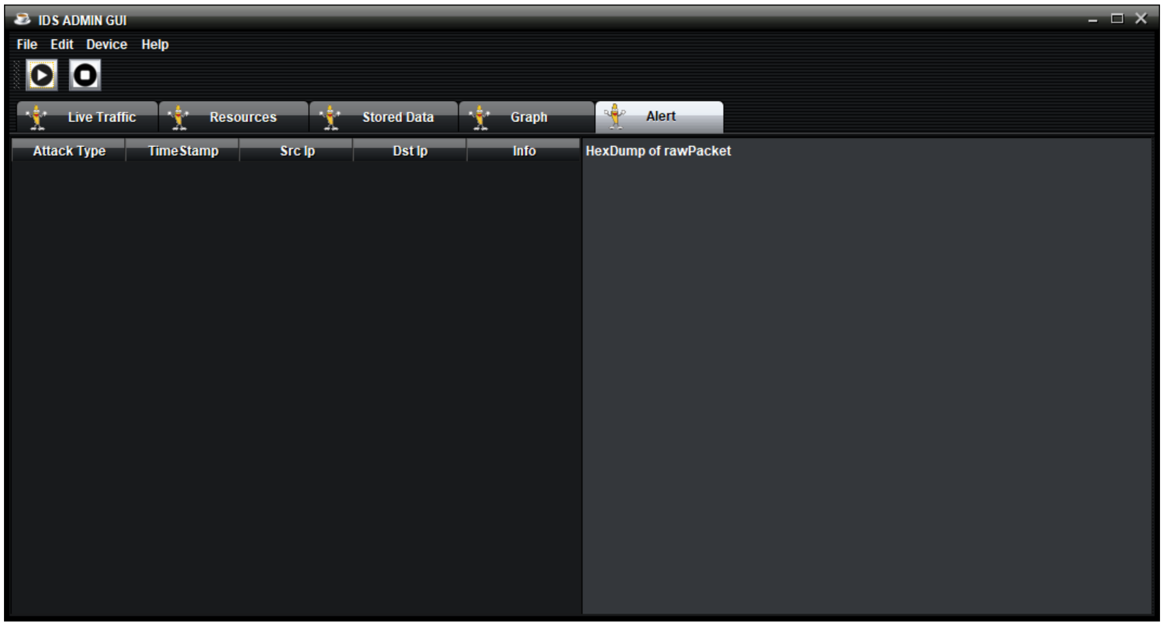
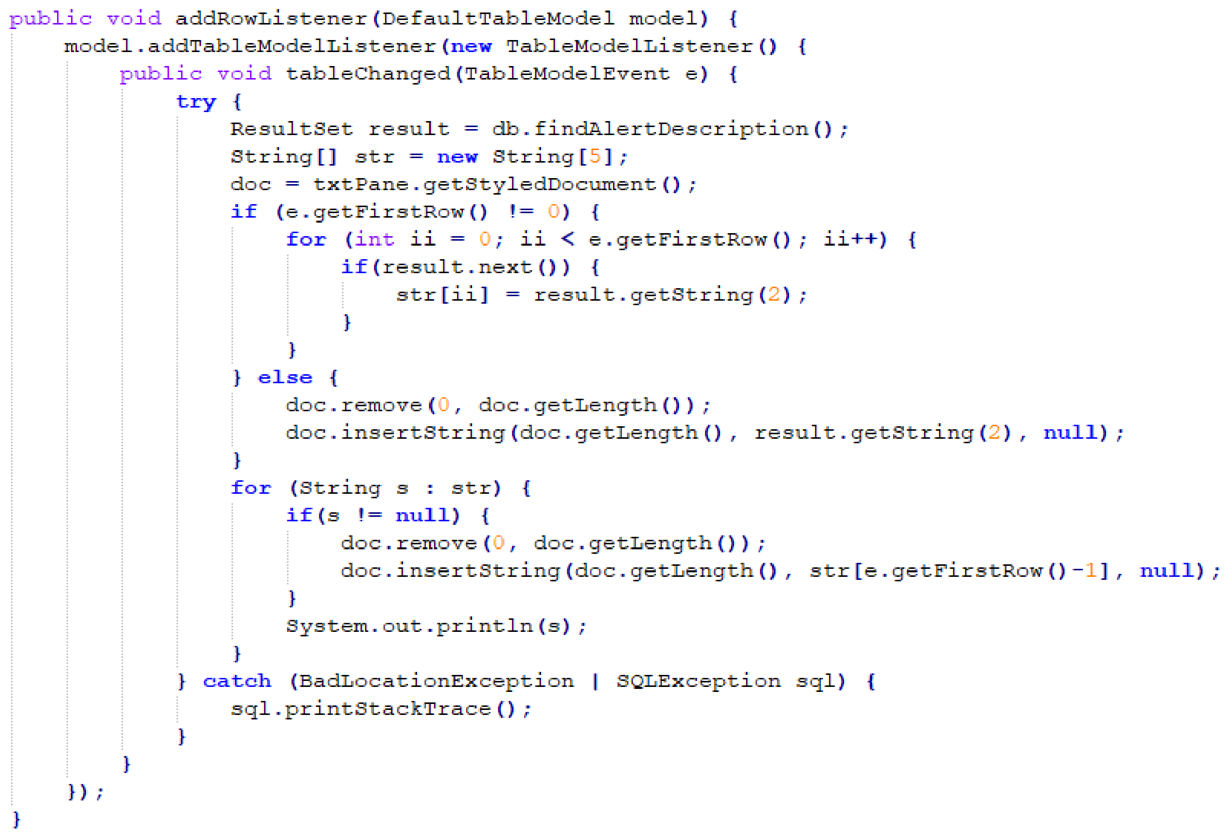


Figure 9.4-3 Alert Query



```
public void addRowListener(DefaultTableModel model) {
    model.addTableModelListener(new TableModelListener() {
        public void tableChanged(TableModelEvent e) {
            try {
                ResultSet result = db.findAlertDescription();
                String[] str = new String[5];
                doc = txtPane.getStyledDocument();
                if (e.getFirstRow() != 0) {
                    for (int ii = 0; ii < e.getFirstRow(); ii++) {
                        if(result.next()) {
                            str[ii] = result.getString(2);
                        }
                    }
                } else {
                    doc.remove(0, doc.getLength());
                    doc.insertString(doc.getLength(), result.getString(2), null);
                }
                for (String s : str) {
                    if(s != null) {
                        doc.remove(0, doc.getLength());
                        doc.insertString(doc.getLength(), str[e.getFirstRow()-1], null);
                    }
                    System.out.println(s);
                }
            } catch (BadLocationException | SQLException sql) {
                sql.printStackTrace();
            }
        }
    });
}
```

Each table created is fitted with a row listener to show corresponding hex dump packet data to the alert row. When an alert row is clicked it uses the result of the database to display the packet on a textpane by matching the selected row with the result row. If a new row is selected the text pane is remove from the document so the new row can be displayed in the textpane. Each string data in the result is placed in a string array so trying to match the row will be an issue because array start at element 0 and MySQL row start with 1 hence the line “doc.insertString(doc.getLength(), str[e.getFirstRow()-1], null);” to match rows. If row selected is row 1 then it shouldn’t be an issue.

10 NIDS Server

The NIDS has a subcomponent (IDS) that can be ran on a single host to manage that host network traffic rather than the NIDS that manages multiple hosts. NIDS attempts to detect hacking activities, denial of service attacks or port scans on computer network. The NIDS detect these malicious activities by identifying suspicious patterns in the packets before triggering an alert operation.

Features:

- DDOS Attack detection.
- Web Vulnerability.
- Man-in-the-Middle Attack detection.
- Brute Force Attack detection.

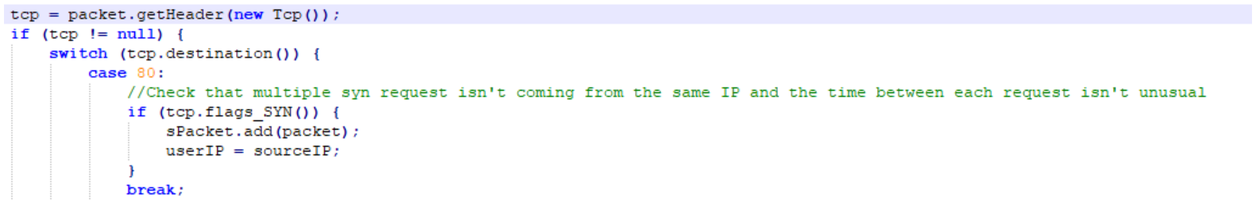
When the IDS is running it request the user to select an interface from the available network interfaces, you can select what type of attack pattern to look out for (TCP, UDP, ICMP). We will follow TCP based attack detection in this report.

10.1 TCP-SYN flood Detection

A syn flood attack is still performed today as a method to shutdown or disable servers so I choose it as an attack feature my NIDS can detect. A tcp-syn flood is when an attack uses the flawed operation of a tcp communication by disrupting its normal function. In a normal tcp communication as in figure 14-2 involves the user computer sending a syn packet to the server and it response with a syn-ack packet waiting to receive an ack from the user computer, when the user computer sends an ack a tcp connection is establish. An attacker breaks this normal communication as it floods the server with a large number of syn packet but doesn't not send an ack, this is when the server begins to send a syn-ack in response to the attacker syn packet but doesn't recive an ack so it put each connection in a waiting stack which uses up the server resources blocking other users from accessing the server. Figure 14-2 also shows this with the attacker sending numerous syn packet.

Now that we know how the tcp-syn flood attack is perform let's look at how the NIDS detects this attack. In figure 10.1-1 the NIDS is running in TCP attack mode and if the targeted port is 80.

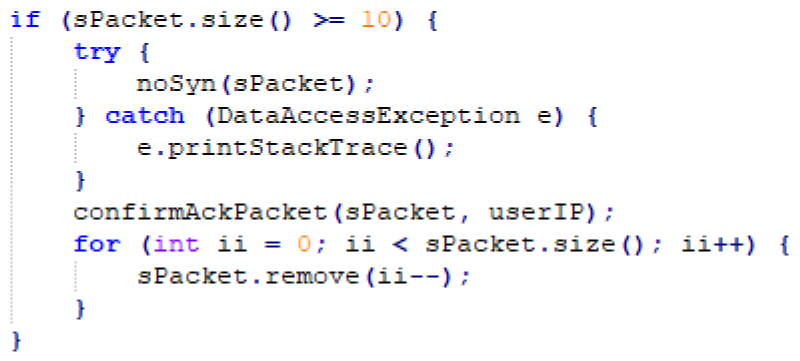
Figure 10.1-1 TCP SYN



```
tcp = packet.getHeader(new Tcp());
if (tcp != null) {
    switch (tcp.destination()) {
        case 80:
            //Check that multiple syn request isn't coming from the same IP and the time between each request isn't unusual
            if (tcp.flags_SYN()) {
                sPacket.add(packet);
                userIP = sourceIP;
            }
            break;
```

The IDS check that the current packet received has a tcp header and if it does is the tcp syn flag set, if both condition are true we can now check if an ack is received by storing the next incoming packet to sPacket an array list.

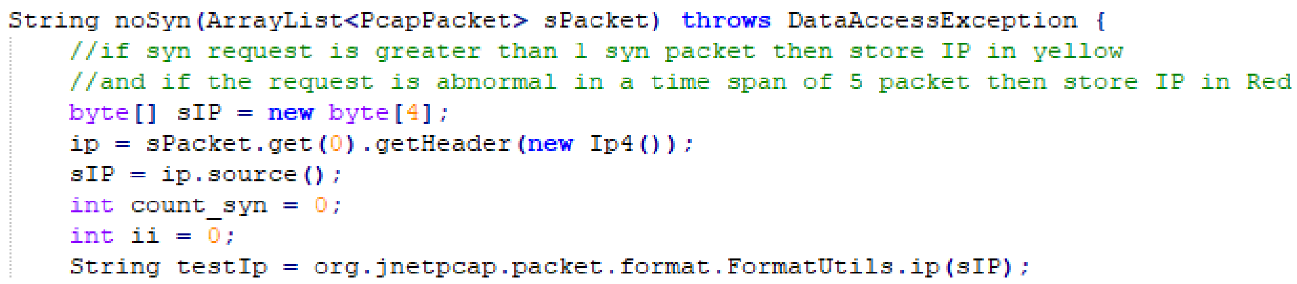
Figure 10.1-2 TCP SYN Count



```
if (sPacket.size() >= 10) {
    try {
        noSyn(sPacket);
    } catch (DataAccessException e) {
        e.printStackTrace();
    }
    confirmAckPacket(sPacket, userIP);
    for (int ii = 0; ii < sPacket.size(); ii++) {
        sPacket.remove(ii--);
    }
}
```

When a number of following packet after the syn has being stored (sPacket.size() >= 10) we count the number of syn in the list before checking that an ack has being sent.

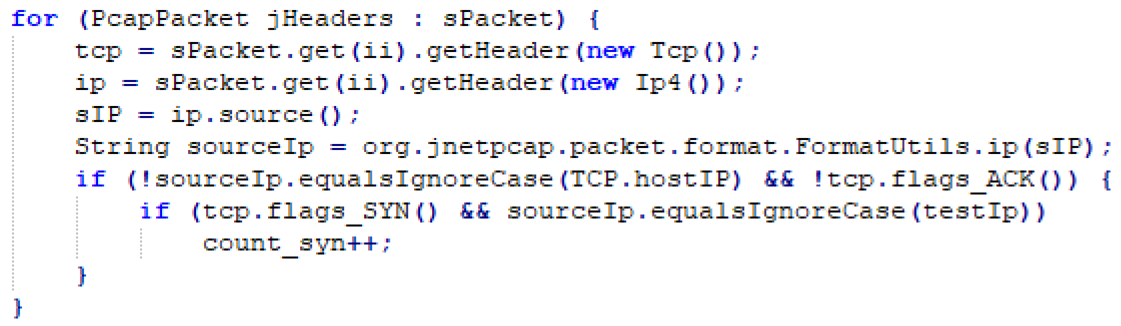
Figure 10.1-3 No Syn



```
String noSyn(ArrayList<PcapPacket> sPacket) throws DataAccessException {
    //if syn request is greater than 1 syn packet then store IP in yellow
    //and if the request is abnormal in a time span of 5 packet then store IP in Red
    byte[] sIP = new byte[4];
    ip = sPacket.get(0).getHeader(new Ip4());
    sIP = ip.source();
    int count_syn = 0;
    int ii = 0;
    String testIp = org.jnetpcap.packet.format.FormatUtils.ip(sIP);
```

Figure 10.1-3 is the method call fo noSyn packet by using the sPacket array list. The source ip is extracted from each packet in other to determine if user is an attacker.

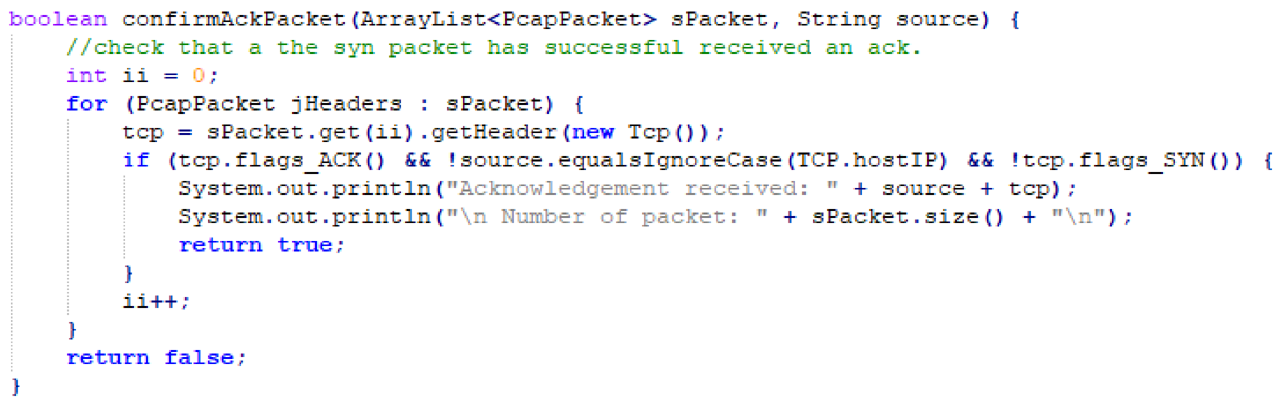
Figure 10.1-4 No Syn



```java
for (PcapPacket jHeaders : sPacket) {
    tcp = sPacket.get(ii).getHeader(new Tcp());
    ip = sPacket.get(ii).getHeader(new Ip4());
    sIP = ip.source();
    String sourceIp = org.jnetpcap.packet.format.FormatUtils.ip(sIP);
    if (!sourceIp.equalsIgnoreCase(TCP.hostIP) && !tcp.flags_ACK()) {
        if (tcp.flags_SYN() && sourceIp.equalsIgnoreCase(testIp))
            count_syn++;
    }
}
```

I then loop through every packet in the list and confirm if the source ip of the computer sent a syn in the without the ack flag being sent.

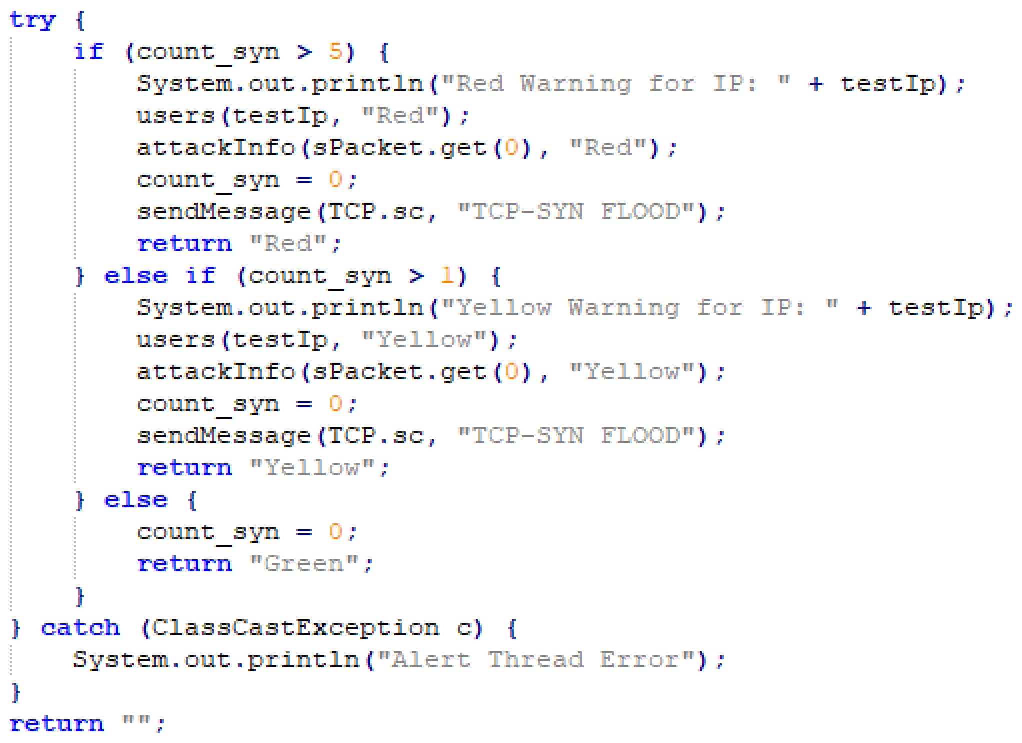
Figure 10.1-5 Confirm ACK



```java
boolean confirmAckPacket(ArrayList<PcapPacket> sPacket, String source) {
    //check that a the syn packet has successful received an ack.
    int ii = 0;
    for (PcapPacket jHeaders : sPacket) {
        tcp = sPacket.get(ii).getHeader(new Tcp());
        if (tcp.flags_ACK() && !source.equalsIgnoreCase(TCP.hostIP) && !tcp.flags_SYN()) {
            System.out.println("Acknowledgement received: " + source + tcp);
            System.out.println("\n Number of packet: " + sPacket.size() + "\n");
            return true;
        }
        ii++;
    }
    return false;
}
```

To confirm that a syn packet has received an ack I loop for each packet in sPacket and checked if the source ip of the syn packet matches the ack source ip. It returns a true if an ack has being received. An ack must be present in a tcp connections hence the reason to confirm if its an attacker or legitimate attack.

Figure 10.1-6 No Syn



```
try {
    if (count_syn > 5) {
        System.out.println("Red Warning for IP: " + testIp);
        users(testIp, "Red");
        attackInfo(sPacket.get(0), "Red");
        count_syn = 0;
        sendMessage(TCP.sc, "TCP-SYN FLOOD");
        return "Red";
    } else if (count_syn > 1) {
        System.out.println("Yellow Warning for IP: " + testIp);
        users(testIp, "Yellow");
        attackInfo(sPacket.get(0), "Yellow");
        count_syn = 0;
        sendMessage(TCP.sc, "TCP-SYN FLOOD");
        return "Yellow";
    } else {
        count_syn = 0;
        return "Green";
    }
} catch (ClassCastException c) {
    System.out.println("Alert Thread Error");
}
return "";
```

Finally, in figure 10.1-6 checks the number of syn packet being sent and if the packet is greater than an unusual amount which is anything over 2 then it should start an alert service, store the attacker info and triggers an alert to the Adnin GUI. The reason for a small window is due to prevent the syn packet from increasing and causing severe damage.

10.2 Alert Service

The alert service only executes when an attack has been detected, it inserts into MySQL and sends an alert message to the admin GUI. During an alert the attacker info is stored using the class User entry with all method to access the information on set attacks. Figure 10.2-1 & 10.2-2 is the UserEntry class build.

Figure 10.2-1 UserEntry

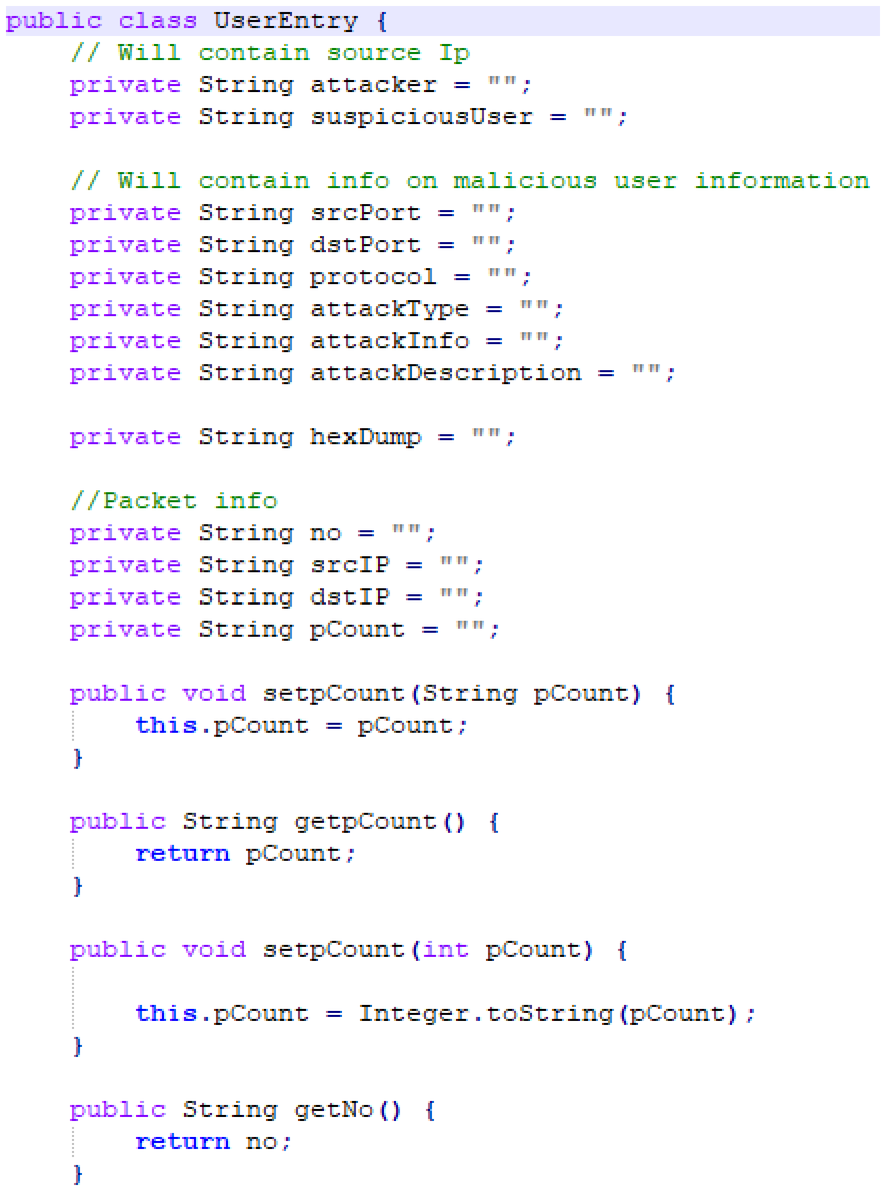

```
public class UserEntry {
    // Will contain source Ip
    private String attacker = "";
    private String suspiciousUser = "";

    // Will contain info on malicious user information
    private String srcPort = "";
    private String dstPort = "";
    private String protocol = "";
    private String attackType = "";
    private String attackInfo = "";
    private String attackDescription = "";

    private String hexDump = "";

    //Packet info
    private String no = "";
    private String srcIP = "";
    private String dstIP = "";
    private String pCount = "";

    public void setpCount(String pCount) {
        this.pCount = pCount;
    }

    public String getpCount() {
        return pCount;
    }

    public void setpCount(int pCount) {

        this.pCount = Integer.toString(pCount);
    }

    public String getNo() {
        return no;
    }
```

Figure 10.2-2 UserEntry

```
public void setSrcPort(String srcPort) {
    this.srcPort = srcPort;
}

public String getDstPort() {
    return dstPort;
}

public void setDstPort(String dstPort) {
    this.dstPort = dstPort;
}

public String getProtocol() {
    return protocol;
}

public void setProtocol(String protocol) {
    this.protocol = protocol;
}

public String getAttackType() {
    return attackType;
}

public void setAttackType(String attackType) {
    this.attackType = attackType;
}

public String getAttackDescription() {
    return attackDescription;
}

public void setAttackDescription(String attackDescription) {
    this.attackDescription = attackDescription;
}

public String getAttackInfo() {
    return attackInfo;
}

public void setAttackInfo(String attackInfo) {
    this.attackInfo = attackInfo;
}

public String getHexDump() {
    return hexDump;
}

public void setHexDump(String hexDump) {
    this.hexDump = hexDump;
}

public String all() {
    return getpCount() + " " + getNo() + " " + getSrcIP() +
            " " + getDstIP() + " " + getProtocol() + " " +
            getSize();
}
```

Once the UserEntry has been filled with the packet info the alert insert into MySQL process can be completed. I used pre statement from MySQL library to insert to the database because pre statement allows me to construct SQL commands for the database to execute, see fig 10.2-3.

Figure 10.2-3 Alert

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Alert implements UserDataAccess {

    private  Connection connection;
    private  PreparedStatement sqlAttackInfo;
    private  PreparedStatement sqlDescription;
    private  PreparedStatement sqlProtocol;
    private  PreparedStatement sqlPacketInfo;
    private  PreparedStatement sqlattactType;

    // set up PreparedStatements to access database
    public Alert() throws Exception
    {
        // connect to project database

        connect();

        sqlAttackInfo = connection.prepareStatement(
                "INSERT INTO attackinfo (srcIP, dstIP, srcPort, dstPort, protocol, attackType, attackInfo)" +
                        "VALUES (?, ?, ?, ?, ?, ?, ?)");
        sqlDescription = connection.prepareStatement(
                "INSERT INTO description ( discription, hexPacket)" +
                        "VALUES (?, ?)");

    }
```

To be able to insert to the database we first call the connect method with figure 10.2-4.

Figure 10.2-4 Alert Connect

```
private void connect() throws Exception
{
    // Cloudscape database driver class name
    String driver = "com.mysql.cj.jdbc.Driver";

    // URL to connect to projectdatabase database
    String url = "jdbc:mysql://192.168.1.159:3306/nids?useUnicode=true&useJDBCCompliantTimezoneShift=true

    // load database driver class
    Class.forName( driver );

    // connect to database
    connection = DriverManager.getConnection( url,"root","root" );

    connection.setAutoCommit( false );
}
```

The method insertDB performs the insertion to the database.

Figure 10.2-4 Alert Insert

```
public  void insertDB(UserEntry uE) throws DataAccessException {
    try {
        int result;

        sqlAttackInfo.setString(1, uE.getAttacker());
        sqlAttackInfo.setString(2, TCP.hostIP);
        sqlAttackInfo.setString(3, uE.getSrcPort());
        sqlAttackInfo.setString(4, uE.getDstPort());
        sqlAttackInfo.setString(5, uE.getProtocol());
        sqlAttackInfo.setString(6, uE.getAttackType());
        sqlAttackInfo.setString(7, uE.getAttackInfo());
        result = sqlAttackInfo.executeUpdate();
        //System.out.println("Result: " + result);

        if(result == 0) {
            System.out.println("Something went wrong");
            connection.rollback();
        }

        sqlDescription.setString(1, uE.getAttackDescription());
        sqlDescription.setString(2, uE.getHexDump());
        result = sqlDescription.executeUpdate();

        if(result == 0) {
            System.out.println("Something went wrong");
            connection.rollback();
        }

        connection.commit();

    } catch (SQLException sqlException) {
        try {
            connection.rollback(); // rollback update
        }

        catch ( SQLException exception ) {
            throw new DataAccessException( exception );
        }
    }
}
```

11 Problem Solving

I faced numerous issues when developing the NIDS, because java was not a programming language I was proficient with hence the reason I choose a java networking project for my fyp.

A lot of the issued I faced during the project where in development, with an already completed research trying to figure out how an attack could take down a server was something I had to also understand in-other to build a robust system. One of the problems I faced in development was how to build the gui and send/receive data. I manage to solve this issue by using java doc to understand swing worker and java NIO. Whenever there was a bug in my code, I used break point to sniff out the bug location then I log the output to see if the code produces the expected output. Error checking is vital lesson I learned when developing the NIDS.

Detecting MITM attack was difficult because it not necessary a server network issue and it can be difficult to tell a legitimate user from a hacker impersonating as a legitimate user for example session hijacking. The way I solve this issue is by checking if 2 or more session with the same session is present.

12 Conclusion

The project isn't something I feel can ever bee closed/finished because there are still numerous attack pattern the NIDS does not know and plenty that may be invented in the future but for what I set out to complete was to have a better understanding with java and java networking which I feel I am more proficient now than when I started this project. The NIDS is in its prototype stage. Overall, this project was interesting and fun to understand how an attacker approach a situation and system admin response. Network security is a never-ending game of cops and robbers.

Checklist:

- Admin GUI.
- Attack Detection.
- Simulation of attack.
- Virtualization.
- Networking.
- GUI.
- Emulation tool.
- Agile Project.

13 References

https://www.virtualbox.org/

http://tutorials.jenkov.com/

https://owasp.org/

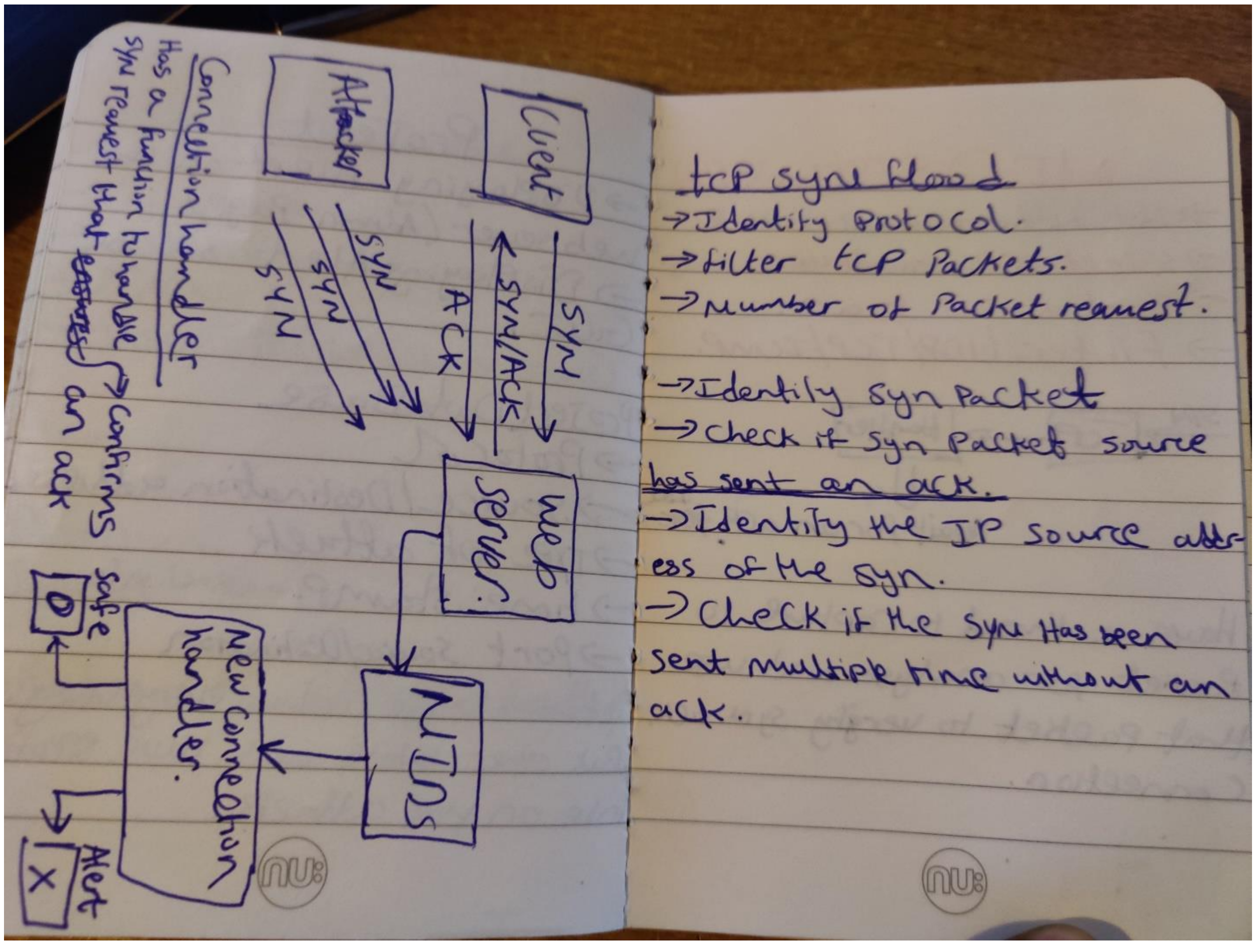
http://jnetpcap.sourceforge.net/docs/jnetpcap-1.0-javadoc/

14 Code and Logbook

Figure 14-1 java Networking

```java
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SocketChannel;

public class MessageHandler {
    public static String data = "";

    public static String receivedData(SocketChannel socketChannel) {
        try {
            ByteBuffer byteBuffer = ByteBuffer.allocate(500);
            String message = "";
            socketChannel.configureBlocking(false);
            while (socketChannel.read(byteBuffer) > 0) {
                char byteRead = 0x00;
                byteBuffer.flip();
                while (byteBuffer.hasRemaining()) {
                    byteRead = (char) byteBuffer.get();
                    if (byteRead == 0x00) {
                        break;
                    }
                    message += byteRead;
                }
                if (byteRead == 0x00) {
                    break;
                }
                byteBuffer.clear();
            }
            return message;
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return "";
    }

    public static void sendMessage (SocketChannel socketChannel, UserEntry ue) {
        try {
            ByteBuffer buffer = ByteBuffer.allocate(ue.all().length() + 1);
            buffer.put(ue.all().getBytes());
            buffer.put((byte) 0x00);
            buffer.flip();
            while(buffer.hasRemaining()) {
                socketChannel.write(buffer);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
```

Figure 14-2 TCP-SYN Flood



tcp syn flood

→Identify protocol.

→filter tcp packets.

→Number of packet request.

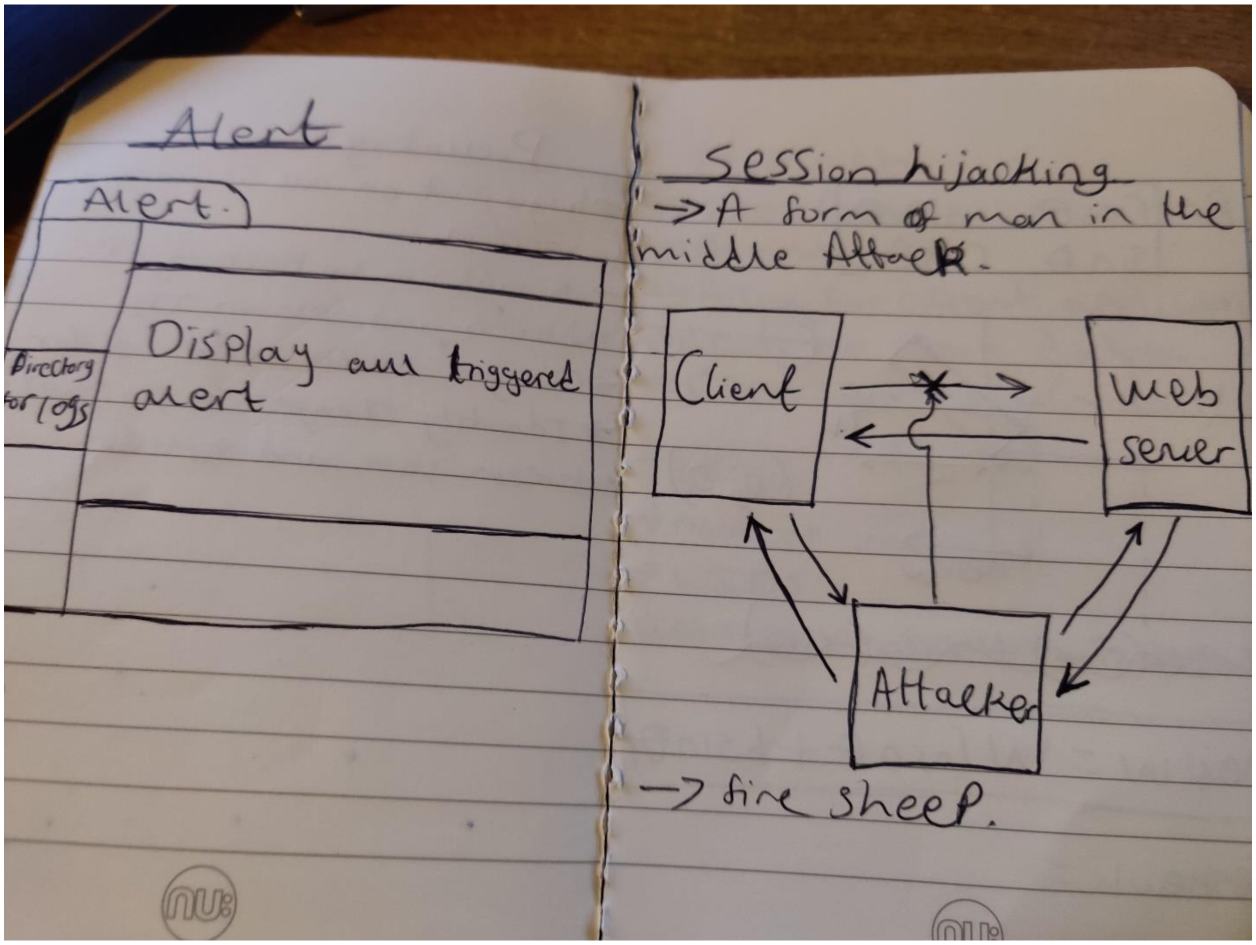
→Identify syn packet

→check if syn packet source has sent an ack.

→Identify the IP source address of the syn.

→Check if the syn has been sent multiple time without an ack.

Figure 14-3 Session Hijacking


Alert
Alert.
Directory for logs
Display all triggered alert
Session hijacking
→ A form of man in the middle Attack.
Client
Web server
Attacker
→ fire sheep.