University of windsor

A3: Designing and implementing Jobs to network application protocol using TCP

Course: COMP 3670 – Computer Networks

Assignment 3 – Part 2

Group 11 Team Members

1. Charles Corro
2. Keertana Madhavan
3. Van Minh Ngai
4. Nitin Ramesh
5. Bilal Sohail

Table of Contents

[Introduction 2](#_Toc57669839)

[One to One Jobs 2](#_Toc57669840)

[Online IP address: 2](#_Toc57669841)

[Port Status: 3](#_Toc57669842)

[One to Many Jobs 4](#_Toc57669843)

[TCP Attack 4](#_Toc57669844)

[UDP Attack 5](#_Toc57669845)

[Testing and Screenshots 6](#_Toc57669846)

[Result of scanning IP to check if online or not 7](#_Toc57669847)

[Result of scanning IP and port number to check for open ports (TCP and UDP) 8](#_Toc57669848)

[Result of initiating an UDP Attack 9](#_Toc57669849)

[Result of initiating an TCP attack 9](#_Toc57669850)

[Code 11](#_Toc57669851)

[Packet Capturing PCAP Directory 11](#_Toc57669852)

# Introduction

In addition to our client-server architecture over TCP, we extended our network application to add the following features offered by the job creator. The following one-to-one services were offered to the job-seeker: Detect if a given IP address is online or not, detect the status of a given port at a given IP address. For the One-To-Many jobs, the following services were offered to the job-seeker: execute a TCP flood attack and execute a UDP flood attack against a given port on a given IP.

# One to One Jobs

## Online IP address:

In this feature the job creator would like the job seeker to find out if a given IP addresses is contacted the network or not. On the JobSeeker.java, the seeker would respond to the job creator request for the target’s IP to find out it has contacted the network or not. In our code, when the creator received response “justip” then creator should enter the IP address and send it to the seeker. The seeker then will create the exec process to contact the machine to acquire a connection or respond and report the status of the IP address based on the getRunTime() method. After finishing the task, the seeker should send a “task done” message to the creator.

*The Job Seeker code when it asks initiates the task to check for online machines*

Graphical user interface, text, application, email

Description automatically generated

## Port Status:

In this feature, the job creator wants to detect the status of a given port of a given IP address. The status of the port will check for TCP and IDP port statuses. The creator will send the IP address and the port number to the seeker. The seeker then will initiate a Port scan to scan for open ports using netcat utility tool to read and write connections using TCP or UDP. After the scan, the contents of the scan and send the “task done” message to the creator.

*The Job Seeker code when it asks initiates the task to check for a machine’s Port Status*

Graphical user interface, text, application, email

Description automatically generated

# One to Many Jobs

## TCP Attack

In this feature, the job creator wants to ask more than one job seeker to execute a TCP flood attack against a. given IP and port number. The creator will send the IP address and the port number to the seeker. The seeker then will initiate a connection to the IP Address and send 1 message every second for a total of 60 seconds. After flooding the machine, the seeker will send a “task done” message to the creator.

*The Job Seeker code when it asks initiates the task to initiate TCP Flood Attack*

Graphical user interface, text, application, email

Description automatically generated

## UDP Attack

In this feature, the job creator wants to ask more than one job seeker to execute a UDP flood attack against a. given IP and port number. The creator will send the IP address and the port number to the seeker. The seeker then will initiate a connection to the IP Address and send 1 message every second for a total of 60 seconds. The DatagramSocket will be used to provide a connection-less point for receiving and sending packets. The DatagramPacket will be used to route each message to the machines. After flooding the machine, the seeker will send a “task done” message to the creator.

*The Job Seeker code when it asks initiates the task to initiate a UDP Flood attack*

Graphical user interface, text, application, email

Description automatically generated

# Testing and Screenshots

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Here you need to download and set up the pcap directory from our repository that aids in creating the packets as shown in the screenshots above.

## Result of scanning IP to check if online or not

Graphical user interface, text

Description automatically generated

## Result of scanning IP and port number to check for open ports (TCP and UDP)

Graphical user interface, text

Description automatically generated

## Result of initiating an UDP Attack

Graphical user interface, text

Description automatically generated

## Result of initiating an TCP attack

Graphical user interface, text

Description automatically generated

# Code

## Packet Capturing PCAP Directory

// App.java

package com.github.username;

import java.io.IOException;

import org.pcap4j.core.\*;

import org.pcap4j.util.\*;

import org.pcap4j.packet.\*;

import java.util.\*;

import org.pcap4j.core.PcapNetworkInterface.PromiscuousMode;

import org.pcap4j.core.BpfProgram.BpfCompileMode;

public class App

{

int packet\_count = 0;

private static final String READ\_TIMEOUT\_KEY = PacketListener.class.getName() + ".readTimeout";

private static final int READ\_TIMEOUT = Integer.getInteger(READ\_TIMEOUT\_KEY, 10); // [ms]

private static final String SNAPLEN\_KEY = PacketListener.class.getName() + ".snaplen";

private static final int SNAPLEN = Integer.getInteger(SNAPLEN\_KEY, 65536); // [bytes]

private static final String TIMESTAMP\_PRECISION\_NANO\_KEY = PacketListener.class.getName() + ".timestampPrecision.nano";

private static final boolean TIMESTAMP\_PRECISION\_NANO = Boolean.getBoolean(TIMESTAMP\_PRECISION\_NANO\_KEY);

public int sniff() throws PcapNativeException, NotOpenException

{

PcapNetworkInterface nif = null;

try {

nif = new NifSelector().selectNetworkInterface();

} catch (IOException e) {

e.printStackTrace();

}

if (nif == null) {

return -1;

}

System.out.println(nif.getName() + "(" + nif.getDescription() + ")");

PcapHandle handle = nif.openLive(65536, PromiscuousMode.PROMISCUOUS, 10);

handle.setFilter("tcp port 4445 or udp port 4445", BpfCompileMode.OPTIMIZE);

while (true)

{

Packet packet = handle.getNextPacket();

if (packet != null)

{

if (packet.contains(IpV4Packet.class) && (packet.contains(TcpPacket.class) || packet.contains(UdpPacket.class)))

{

IpV4Packet pkt = packet.get(IpV4Packet.class);

if (packet.contains (TcpPacket.class))

{

TcpPacket tcpPkt = packet.get(TcpPacket.class);

System.out.println(tcpPkt.getHeader().getDstPort().value());

System.out.println("TCP -->"+pkt.getHeader().getSrcAddr() + ":" + tcpPkt.getHeader().getDstPort());

}

else

{

UdpPacket udpPkt = packet.get(UdpPacket.class);

System.out.println(udpPkt.getHeader().getDstPort().value());

System.out.println("UDP -->"+pkt.getHeader().getSrcAddr() + ":" + udpPkt.getHeader().getDstPort());

}

}

else

{

System.out.println("Unknown packet ---> "+ packet.getPayload());

}

packet\_count++;

}

}

}

public static void main(String[] args) throws PcapNativeException, NotOpenException

{

App o1=new App();

o1.sniff();

}

}