A  Project Report on

DNS 0x20

In the partial fulfilment of the academic requirements for

M. S (Cyber Security) under Wright State University

By

Sudarshan Govindarajan               (U00793458)

**Implementation of the System**

The following is the report that illustrates the implementation of the DNS 0x20 obtained by expanding the existing system.

Most of the java class files are same as provided by the author.

The following class has been modified and all the changes have been commented.

**DenisServer.java**

package com.appspot.trent.denis;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.BindException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.Inet4Address;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.Properties;

import java.util.Random;

import java.util.Set;

import java.util.TreeMap;

public class DnsServer extends Thread {

public DnsServer(int udpPort) throws UnknownHostException, SocketException {

// create master resolver

byte serverAddr[] = Denis.dnsSource.clone();

//Original source for setting DNS address, replaced by above.

//byte serverAddr[] = new byte[4];

// FIXME: default dns server address. You could change it if you want

//serverAddr[0] = (byte) 130;

//serverAddr[1] = (byte) 108;

//serverAddr[2] = (byte) 128;

//serverAddr[3] = (byte) 200;

dnsServerAddr = (Inet4Address) Inet4Address.getByAddress(serverAddr);

try {

badAddress = new IPAddress(new byte[]{(byte)208,67,(byte)219,(byte)130});

} catch (Exception e1) {

e1.printStackTrace();

return;

}

boolean boundPort = false, alternate = false;

try {

masterResolver = new DnsResolver(dnsServerAddr, 53);

} catch (Exception e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

return;

}

hackResolver = new HttpResolver();

//Attempt to bind the port. If this fails, use the alternate port.

//Keep incrementing by one until it works.

while (!boundPort)

{

try {

listenSocket = new DatagramSocket(udpPort);

} catch (Exception e) {

System.out.println("Failed to bind port: " + udpPort);

if (!alternate)

{

System.out.println("Trying alternate: " + Denis.fallbackPort);

udpPort = Denis.fallbackPort;

alternate = true;

}

else

{

System.out.println("Trying: " + ++udpPort);

}

continue;

}

boundPort = true;

}

System.out.println("Running on port: " + udpPort);

initShellCommands();

}

private abstract class CmdHandler {

CmdHandler(String helpMsg) {

this.helpMsg = helpMsg;

}

abstract void handleCmd(String args[]) throws Exception;

String helpMsg;

}

public void run() {

// we don't expect this program to run on heavy load

// everything runs in ONE thread!

byte buffer[] = new byte[1024];

while (true) {

try {

DatagramPacket p = new DatagramPacket(buffer, buffer.length);

listenSocket.receive(p);

if (p.getLength() == 0) {

System.out.println("Exiting service thread");

break;

}

DnsRequest request = new DnsRequest(buffer, p.getLength());

int txnId = request.getTxnId();

ArrayList<DnsQuestion> questions = request.getQuestions();

synchronized(hostCache) {

for (DnsQuestion question: questions) {

String host = question.getDomainName();

char[] storehost=new char[host.length()];

//creation of the char array named storehost which is an array of modified hostname characters through 0x20 bit-encoding

String newhost;

//creation of the string named newhost which is a string of arrayed characters of the modified domainname through 0x20 bit encoding

Random randomNumber= new Random();

//creation of the Random class object named randomNumber

int randomInt;

//creation of the randomInt which stores the random integer obtained through Random class nextInt function

for(int i=0;i<host.length();i++)

{

randomInt = randomNumber.nextInt(100);

//generates a random integer less than 100 and assigned to randomInt

//if the randomInt is less than 50 then character array contains the lower case value

if(randomInt<50) {

storehost[i]=host.charAt(i);

}

//if the randomInt is greater than 50 then character array contains the upper case value

else

{

storehost[i]=Character.toUpperCase(host.charAt(i));

}

}

newhost=new String(storehost);

//converts the character array to string

HostRecord record = resolve(newhost);

//tries to resolve the 0x20 bitencoding hostname

// send back response, record may be null.

byte reply[] = DnsResponse.constructPacket(record, txnId);

DatagramPacket replyPkt = new DatagramPacket(reply, reply.length);

replyPkt.setAddress(p.getAddress());

replyPkt.setPort(p.getPort());

listenSocket.send(replyPkt);

//System.out.println("Served " + host + ": " + record);

}

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

public HostRecord resolve(String domainName) throws Exception {

synchronized (hostCache) {

// scan cache

HostRecord record = hostCache.findHost(domainName);

if (record == null) {

// ask master resolver

record = masterResolver.addressForHost(domainName);

}

// XXX: do hacking!

record = hackRecord(record);

// add to cache

hostCache.addHost(record);

return record;

}

}

/\*\*

\* Hack the record if it was originally filtered by opendns

\*

\* @param record

\* @return

\* @throws Exception

\*/

public HostRecord hackRecord(HostRecord record) throws Exception {

for (IPAddress address: record.getIpAddresses()) {

if (address.equals(badAddress)) {

//System.out.println("Hacking "+record.getDomainName());

record = hackResolver.addressForHost(record.getDomainName());

} else {

//System.out.println("Addr " + address + " != " + filteredAddr);

}

}

return record;

}

public void stopThread() throws Exception {

// send an empty packet to listenSocket

Inet4Address listenAddress = (Inet4Address) Inet4Address.getByAddress(new byte[]{127, 0, 0, 1});

int listenPort = listenSocket.getLocalPort();

byte data[] = new byte[0];

DatagramPacket p = new DatagramPacket(data, 0, listenAddress, listenPort);

listenSocket.send(p);

join();

}

public void saveCfg(String fileName) throws Exception {

// save config

synchronized(hostCache) {

Properties p = new Properties();

for (HostRecord r: hostCache.getItems()) {

p.setProperty(r.getDomainName(), r.getIpAddresses().toString());

}

if (fileName.equals("stdout")) {

p.store(System.out, "Denis Cache");

} else {

p.store(new FileOutputStream(fileName), "Denis Cache");

}

}

}

public void loadCfg(String fileName) throws Exception {

// load config

synchronized(hostCache) {

Properties p = new Properties();

p.load(new FileInputStream(fileName));

for (Object k: p.keySet()) {

String key = (String)k;

String value = p.getProperty(key);

// build host record from value

HostRecord record = new HostRecord(key);

String ips[] = value.substring(1, value.length()-1).split(", ");

for (String ip: ips) {

String s[] = ip.split("\\.");

if (s.length == 4) {

byte parts[] = new byte[4];

for (int i = 0; i < s.length; i++)

parts[i] = (byte) Integer.parseInt(s[i]);

record.addIpAddress(new IPAddress(parts));

}

}

hostCache.addHost(record);

}

}

}

/\*\*

\* Print the cache on screen, ignoring empty items

\*/

@SuppressWarnings("unchecked")

public void showCache() {

ArrayList<HostRecord> validRecords = new ArrayList<HostRecord>();

synchronized(hostCache) {

for (HostRecord hostRecord: hostCache.getItems()) {

if (hostRecord.getIpAddresses().size() > 0) {

validRecords.add(hostRecord);

}

}

}

Collections.sort(validRecords);

for (HostRecord hostRecord: validRecords) {

//System.out.println(hostRecord.getDomainName() + " " + hostRecord.getIpAddresses());

for (IPAddress address: hostRecord.getIpAddresses()) {

System.out.println(address.toString() + " " + hostRecord.getDomainName());

}

}

}

public void showShellHelp() {

// show shell help

for (String key: cmdTable.keySet()) {

CmdHandler handler = cmdTable.get(key);

System.out.println(handler.helpMsg);

}

}

public void exitShell() {

// exit shell

exitShell = true;

try {

stopThread();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public void initShellCommands() {

cmdTable.put("save", new CmdHandler("save fileName -- save cache to fileName") {

@Override

void handleCmd(String[] args) throws Exception {

if (args.length == 2) {

String fileName = args[1];

saveCfg(fileName);

} else {

showShellHelp();

}

}

});

cmdTable.put("load", new CmdHandler("load filename -- load cache from fileName") {

@Override

void handleCmd(String[] args) throws Exception {

if (args.length == 2) {

String fileName = args[1];

loadCfg(fileName);

} else {

showShellHelp();

}

}

});

cmdTable.put("help", new CmdHandler("help -- show help message") {

@Override

void handleCmd(String[] args) throws Exception {

showShellHelp();

}

});

cmdTable.put("exit", new CmdHandler("exit -- exit shell") {

@Override

void handleCmd(String[] args) throws Exception {

exitShell();

}

});

cmdTable.put("cache", new CmdHandler("cache -- show cache items") {

@Override

void handleCmd(String[] args) throws Exception {

showCache();

}

});

cmdTable.put("resolve", new CmdHandler("resolve domainName -- resolve host") {

@Override

void handleCmd(String[] args) throws Exception {

if (args.length == 2) {

HostRecord hostRecord = resolve(args[1]);

System.out.println(hostRecord);

} else {

showShellHelp();

}

}

});

}

/\*\*

\* Start a shell

\*/

public void runShell() {

exitShell = false;

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

try {

while (!exitShell) {

System.out.print(">> ");

String line = br.readLine();

String args[] = line.split(" ");

if (args.length < 1) {

showShellHelp();

} else {

CmdHandler handler = cmdTable.get(args[0]);

if (handler == null) {

showShellHelp();

} else {

// run command

try {

handler.handleCmd(args);

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

}

} catch (Exception ex) {

exitShell = true;

ex.printStackTrace();

}

}

HostCache hostCache = new HostCache();

Resolver masterResolver;

Resolver hackResolver;

Inet4Address dnsServerAddr;

IPAddress badAddress; // FIXME: if the returned address is badAddress, we hack it

DatagramSocket listenSocket;

HashMap<String, CmdHandler> cmdTable = new HashMap<String, CmdHandler>();

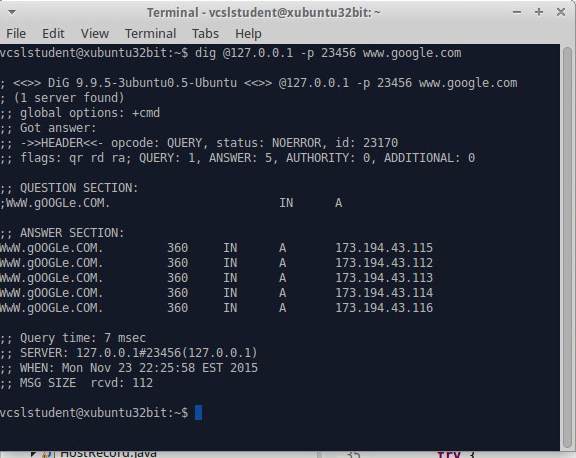
boolean exitShell;

}

**Output**

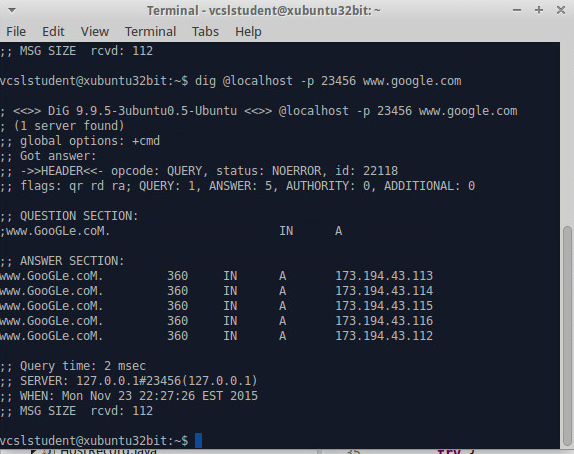
**First attempt**

**dig @127.0.0.1 –p 23456** [**www.google.com**](http://www.google.com)



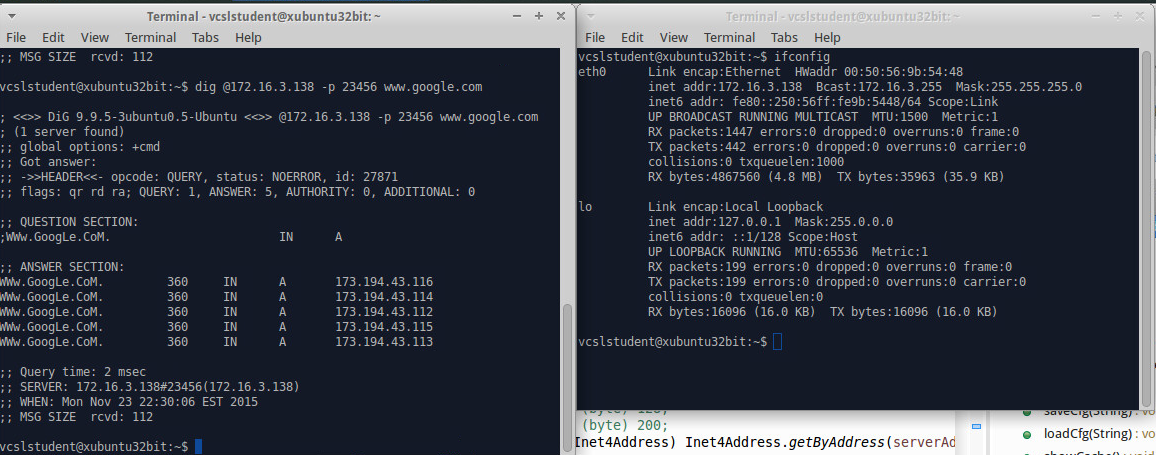
**Second attempt**

**dig @localhost –p 23456** [**www.google.com**](http://www.google.com)



**Third attempt**

**dig @ipaddress –p 23456** [**www.google.com**](http://www.google.com)



[www.google.com](http://www.google.com) is represented different during each time

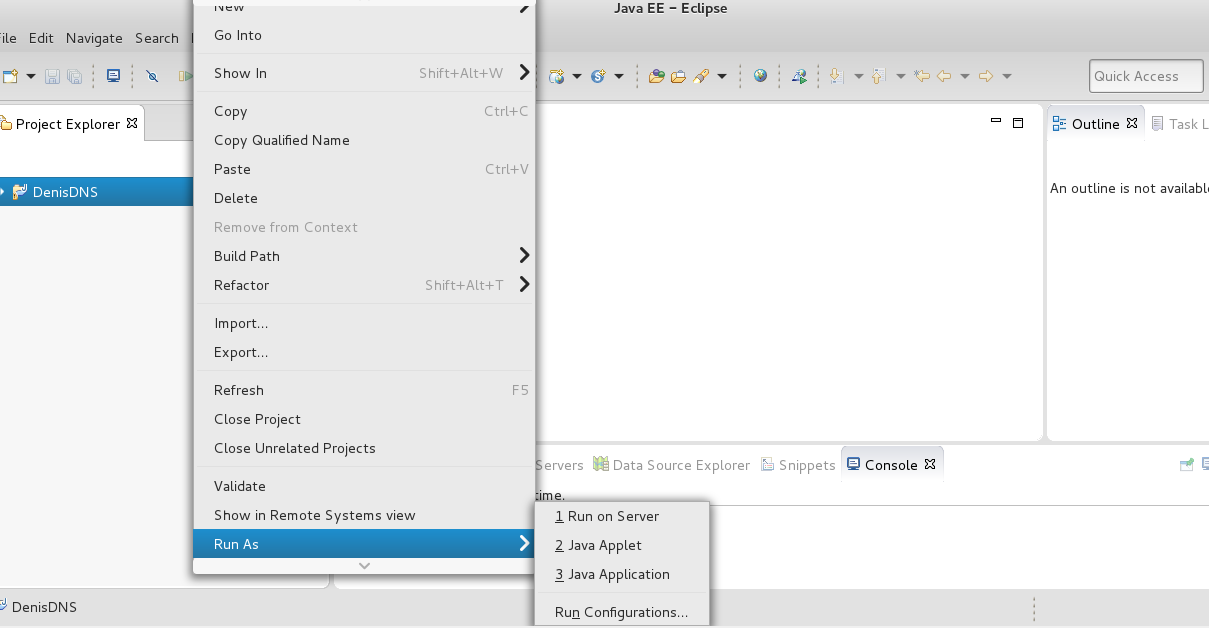
**Manual for Execution**

Executing DNS 0x20

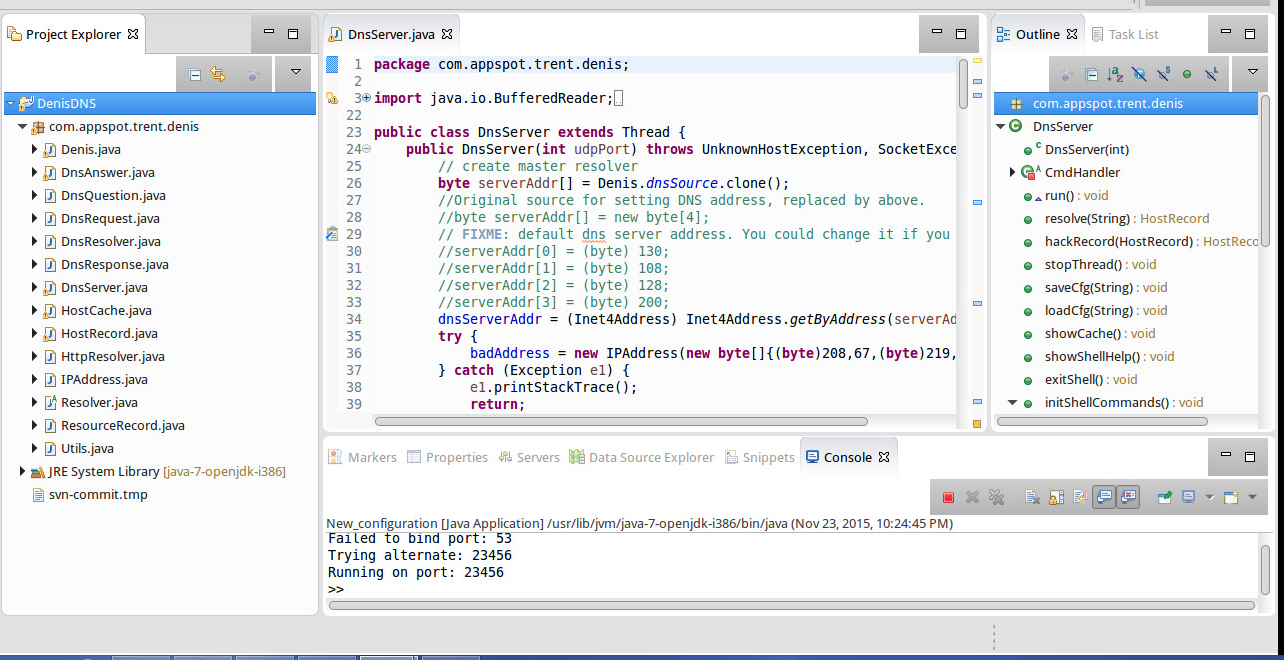
Step 1: Install the eclipse

Step 2: Import the file Project 3

Step 3:Click RunAs and Run Configurations



Step 4: The following output will be obtained



Execute the Dig command as client

**dig @127.0.0.1 –p 23456 www.google.com**

