**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Practical’s** | **Date** | **Sign.** |
| 1. | Write programs to implement the following Substitution Cipher Techniques:   1. Caesar Cipher. 2. Monoalphabetic Cipher. |  |  |
| 2. | Write programs to implement the following Substitution Cipher Techniques:   1. Vernam Cipher. 2. Playfair Cipher. |  |  |
| 3. | Write programs to implement the following Transposition Cipher Techniques:   1. Rail Fence Cipher. 2. Simple Columnar Technique. |  |  |
| 4. | Write program to encrypt and decrypt strings using:   1. DES Algorithm. 2. AES Algorithm. |  |  |
| 5. | Write a program to implement RSA algorithm to perform encryption / decryption of a given string. |  |  |
| 6. | Write a program to implement the Diffie-Hellman Key Agreement algorithm to generate symmetric keys. |  |  |
| 7. | Write a program to implement the MD5 algorithm compute the message digest. |  |  |
| 8. | Write a program to calculate HMAC-SHA1 Signature |  |  |
| 9. | Write a program to implement SSL. |  |  |
| 10. | Configure Windows Firewall to block:   1. A port 2. A Program 3. A website |  |  |

**Practical No: 01**

**Aim: Write programs to implement the following Substitution Cipher Techniques:**

1. **Caesar Cipher.**

**Input:**

class CaesarCipher

{

// Encrypts text using a shift od s

public static StringBuffer encrypt(String text, int s)

{

StringBuffer result= new StringBuffer();

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

{

if (Character.isUpperCase(text.charAt(i)))

{

char ch = (char)(((int)text.charAt(i) +

s - 65) % 26 + 65);

result.append(ch);

}

else

{

char ch = (char)(((int)text.charAt(i) +

s - 97) % 26 + 97);

result.append(ch);

}

}

return result;

}

// Driver code

public static void main(String[] args)

{

String text = "pvg";

int s = 4;

System.out.println("Text : " + text);

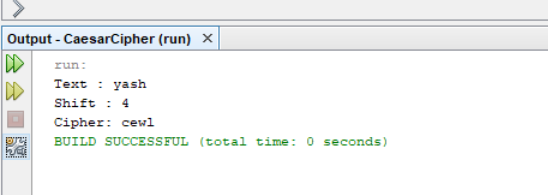
System.out.println("Shift : " + s);

System.out.println("Cipher: " + encrypt(text, s));

}

}

**Output:**



1. **Monoalphabetic Cipher**

**Input :**

import java.util.\*;

public class MonoalphabeticCipherEncryption {

public static void main(String [] args) {

String plaintext = "meet me after the toga party";

String alphabet = "abcdefghijklmnopqrstuvwxyz";

String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

ArrayList<Integer> permutation = new ArrayList<>();

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

permutation.add(i);

Collections.shuffle(permutation);

String key = "", KEY = "";

for (int j = 0; j < ALPHABET.length(); j++) {

key += alphabet.charAt(permutation.get(j));

KEY += ALPHABET.charAt(permutation.get(j));

}

String ciphertext = "";

int i, j;

for (i = 0; i < plaintext.length(); i++) {

for (j = 0; j < alphabet.length(); j++) {

if (plaintext.charAt(i) == alphabet.charAt(j)) {

ciphertext += key.charAt(j);

break;

}

if (plaintext.charAt(i) == ALPHABET.charAt(j)) {

ciphertext += KEY.charAt(j);

break;

}

}

if (j == ALPHABET.length())

ciphertext += plaintext.charAt(i);

}

System.out.println("Monoalphabetic Cipher [Encryption]");

System.out.println("Plain Text : " + plaintext);

System.out.println("key : " + key);

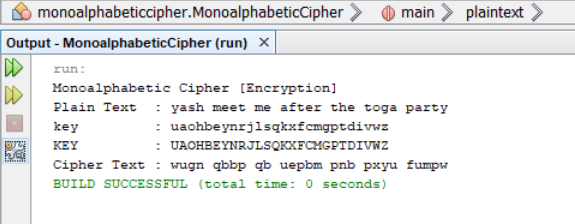
System.out.println("KEY : " + KEY);

System.out.println("Cipher Text : " + ciphertext);

}

}

**Output:**



**Practical No: 02**

**Aim: Write programs to implement the following Substitution Cipher Techniques:**

1. **Playfair Cipher**

**Input:**

import java.util.Scanner;

public class PlayfairCipherEncryption

{

private String KeyWord = new String();

private String Key = new String();

private char matrix\_arr[][] = new char[5][5];

public void setKey(String k)

{

String K\_adjust = new String();

boolean flag = false;

K\_adjust = K\_adjust + k.charAt(0);

for (int i = 1; i < k.length(); i++)

{

for (int j = 0; j < K\_adjust.length(); j++)

{

if (k.charAt(i) == K\_adjust.charAt(j))

{

flag = true;

}

}

if (flag == false)

K\_adjust = K\_adjust + k.charAt(i);

flag = false;

}

KeyWord = K\_adjust;

}

public void KeyGen()

{

boolean flag = true;

char current;

Key = KeyWord;

for (int i = 0; i < 26; i++)

{

current = (char) (i + 97);

if (current == 'j')

continue;

for (int j = 0; j < KeyWord.length(); j++)

{

if (current == KeyWord.charAt(j))

{

flag = false;

break;

}

}

if (flag)

Key = Key + current;

flag = true;

}

System.out.println(Key);

matrix();

}

private void matrix()

{

int counter = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

matrix\_arr[i][j] = Key.charAt(counter);

System.out.print(matrix\_arr[i][j] + " ");

counter++;

}

System.out.println();

}

}

private String format(String old\_text)

{

int i = 0;

int len = 0;

String text = new String();

len = old\_text.length();

for (int tmp = 0; tmp < len; tmp++)

{

if (old\_text.charAt(tmp) == 'j')

{

text = text + 'i';

}

else

text = text + old\_text.charAt(tmp);

}

len = text.length();

for (i = 0; i < len; i = i + 2)

{

if (text.charAt(i + 1) == text.charAt(i))

{

text = text.substring(0, i + 1) + 'x' + text.substring(i + 1);

}

}

return text;

}

private String[] Divid2Pairs(String new\_string)

{

String Original = format(new\_string);

int size = Original.length();

if (size % 2 != 0)

{

size++;

Original = Original + 'x';

}

String x[] = new String[size / 2];

int counter = 0;

for (int i = 0; i < size / 2; i++)

{

x[i] = Original.substring(counter, counter + 2);

counter = counter + 2;

}

return x;

}

public int[] GetDiminsions(char letter)

{

int[] key = new int[2];

if (letter == 'j')

letter = 'i';

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (matrix\_arr[i][j] == letter)

{

key[0] = i;

key[1] = j;

break;

}

}

}

return key;

}

public String encryptMessage(String Source)

{

String src\_arr[] = Divid2Pairs(Source);

String Code = new String();

char one;

char two;

int part1[] = new int[2];

int part2[] = new int[2];

for (int i = 0; i < src\_arr.length; i++)

{

one = src\_arr[i].charAt(0);

two = src\_arr[i].charAt(1);

part1 = GetDiminsions(one);

part2 = GetDiminsions(two);

if (part1[0] == part2[0])

{

if (part1[1] < 4)

part1[1]++;

else

part1[1] = 0;

if (part2[1] < 4)

part2[1]++;

else

part2[1] = 0;

}

else if (part1[1] == part2[1])

{

if (part1[0] < 4)

part1[0]++;

else

part1[0] = 0;

if (part2[0] < 4)

part2[0]++;

else

part2[0] = 0;

}

else

{

int temp = part1[1];

part1[1] = part2[1];

part2[1] = temp;

}

Code = Code + matrix\_arr[part1[0]][part1[1]]

+ matrix\_arr[part2[0]][part2[1]];

}

return Code;

}

public static void main(String[] args)

{

PlayfairCipherEncryption x = new PlayfairCipherEncryption();

Scanner sc = new Scanner(System.in);

System.out.println("Enter a keyword:");

String keyword = sc.next();

x.setKey(keyword);

x.KeyGen();

System.out

.println("Enter word to encrypt: (Make sure length of message is even)");

String key\_input = sc.next();

if (key\_input.length() % 2 == 0)

{

System.out.println("Encryption: " + x.encryptMessage(key\_input));

}

else

{

System.out.println("Message length should be even");

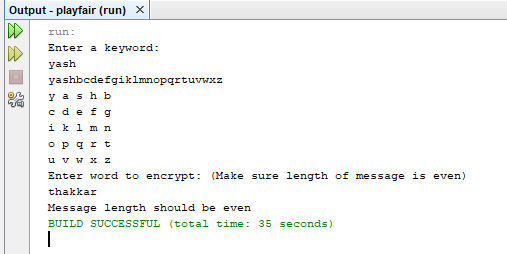
}

sc.close();

}

}

**Output:**



1. **Vernam Cipher**

**Input:**

import java.lang.Math;

public class xor1 {

public static void main(String args[]) {

// This would be the text we encrypt (in this case "hello")

// We convert it to a character array

String text = new String("hello");

char[] arText = text.toCharArray();

// This would be our vernam cipher (should be same length as our text)

// Here we use the same letters, but theoretically should be random

// characters generated on the fly. USE RANDOM LETTERS!

String cipher = new String("XYZHG");

char[] arCipher = cipher.toCharArray();

// Array to hold our encryption (again same length)

char[] encoded = new char[5];

// Encrypt the text by using XOR (exclusive OR) each character

// of our text against cipher.

System.out.println("Encoded " + text + " to be... ");

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

encoded[i] = (char) (arText[i] ^ arCipher[i]);

System.out.print(encoded[i]);

}

System.out.println("\nDecoded to be... ");

// Run through the encrypted text and against the cipher again

// This decrypts the text.

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

char temp = (char) (encoded[i] ^ arCipher[i]);

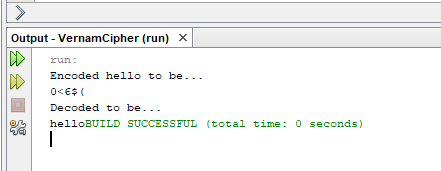
System.out.print(temp);

}

System.out.print(“\n”);

}}

**Output:**



**Practical No: 03**

**Aim: Write programs to implement the following Transposition Cipher Techniques:**

1. **Rail Fence Cipher**

**Input:**

import java.util.\*;

class RailFenceBasic{

int depth;

String Encryption(String plainText,int depth)throws Exception

{

int r=depth,len=plainText.length();

int c=len/depth;

char mat[][]=new char[r][c];

int k=0;

String cipherText="";

for(int i=0;i< c;i++)

{

for(int j=0;j< r;j++)

{

if(k!=len)

mat[j][i]=plainText.charAt(k++);

else

mat[j][i]='X';

}

}

for(int i=0;i< r;i++)

{

for(int j=0;j< c;j++)

{

cipherText+=mat[i][j];

}

}

return cipherText;

}

String Decryption(String cipherText,int depth)throws Exception

{

int r=depth,len=cipherText.length();

int c=len/depth;

char mat[][]=new char[r][c];

int k=0;

String plainText="";

for(int i=0;i< r;i++)

{

for(int j=0;j< c;j++)

{

mat[i][j]=cipherText.charAt(k++);

}

}

for(int i=0;i< c;i++)

{

for(int j=0;j< r;j++)

{

plainText+=mat[j][i];

}

}

return plainText;

}

}

class RailFence{

public static void main(String args[])throws Exception

{

RailFenceBasic rf=new RailFenceBasic();

Scanner scn=new Scanner(System.in);

int depth;

String plainText,cipherText,decryptedText;

System.out.println("Enter plain text:");

plainText=scn.nextLine();

System.out.println("Enter depth for Encryption:");

depth=scn.nextInt();

cipherText=rf.Encryption(plainText,depth);

System.out.println("Encrypted text is:\n"+cipherText);

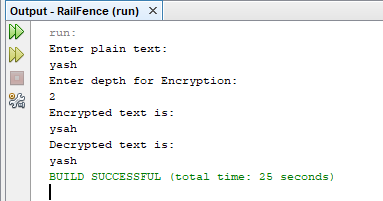
decryptedText=rf.Decryption(cipherText, depth);

System.out.println("Decrypted text is:\n"+decryptedText);

}

}

**Output:**



1. **Simple Columnar Technique**

**Input :**

import java.util.\*;

class simpleColumnar

{

public static void main(String sap[]){

Scanner sc = new Scanner(System.in);

System.out.print("\nEnter plaintext(enter in lower case): ");

String message = sc.next();

System.out.print("\nEnter key in numbers: ");

String key = sc.next();

int columnCount = key.length();

int rowCount = (message.length()+columnCount)/columnCount;

int plainText[][] = new int[rowCount][columnCount];

int cipherText[][] = new int[rowCount][columnCount];

System.out.print("\n-----Encryption-----\n");

cipherText = encrypt(plainText, cipherText, message, rowCount, columnCount, key);

String ct = "";

for(int i=0; i<columnCount; i++)

{

for(int j=0; j<rowCount; j++)

{

if(cipherText[j][i] == 0)

ct = ct + 'x';

else{

ct = ct + (char)cipherText[j][i];

}

}

}

System.out.print("\nCipher Text: " + ct);

System.out.print("\n\n\n-----Decryption-----\n");

plainText = decrypt(plainText, cipherText, ct, rowCount, columnCount, key);

// prepare final string

String pt = "";

for(int i=0; i<rowCount; i++)

{

for(int j=0; j<columnCount; j++)

{

if(plainText[i][j] == 0)

pt = pt + "";

else{

pt = pt + (char)plainText[i][j];

}

}

}

System.out.print("\nPlain Text: " + pt);

System.out.println();

}

static int[][] encrypt(int plainText[][], int cipherText[][], String message, int rowCount, int columnCount, String key){

int i,j;

int k=0;

for(i=0; i<rowCount; i++)

{

for(j=0; j<columnCount; j++)

{

if(k < message.length())

{

plainText[i][j] = (int)message.charAt(k);

k++;

}

else

{

break;

}

}

}

for(i=0; i<columnCount; i++)

{

int currentCol= ( (int)key.charAt(i) - 48 ) -1;

for(j=0; j<rowCount; j++)

{

cipherText[j][i] = plainText[j][currentCol];

}

}

System.out.print("Cipher Array(read column by column): \n");

for(i=0;i<rowCount;i++){

for(j=0;j<columnCount;j++){

System.out.print((char)cipherText[i][j]+"\t");

}

System.out.println();

}

return cipherText;

}

static int[][] decrypt(int plainText[][], int cipherText[][], String message, int rowCount, int columnCount, String key){

int i,j;

int k=0;

for(i=0; i<columnCount; i++)

{

int currentCol= ( (int)key.charAt(i) - 48 ) -1;

for(j=0; j<rowCount; j++)

{

plainText[j][currentCol] = cipherText[j][i];

}

}

System.out.print("Plain Array(read row by row): \n");

for(i=0;i<rowCount;i++){

for(j=0;j<columnCount;j++){

System.out.print((char)plainText[i][j]+"\t");

}

System.out.println();

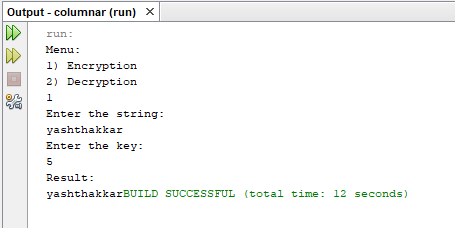
}++++++++++++++++++

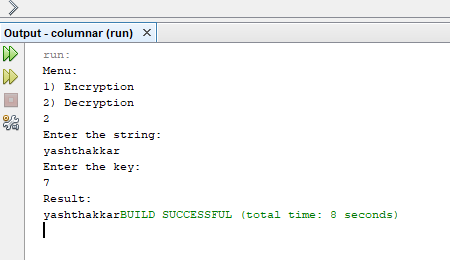
return plainText;

}

}

**Output:**





**Practical No : 04**

**Aim: Write program to encrypt and decrypt strings using**

1. **DES Algorithm**

**Input:**

import javax.swing.\*;

import java.security.SecureRandom;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Random ;

class DES {

byte[] skey = new byte[1000];

String skeyString;

static byte[] raw;

String inputMessage,encryptedData,decryptedMessage;

public DES() {

try {

generateSymmetricKey();

inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");

byte[] ibyte = inputMessage.getBytes();

byte[] ebyte=encrypt(raw, ibyte);

String encryptedData = new String(ebyte);

System.out.println("Encrypted message "+encryptedData);

JOptionPane.showMessageDialog(null,"Encrypted Data "+"\n"+encryptedData);

byte[] dbyte= decrypt(raw,ebyte);

String decryptedMessage = new String(dbyte);

System.out.println("Decrypted message "+decryptedMessage);

JOptionPane.showMessageDialog(null,"Decrypted Data "+"\n"+decryptedMessage);

}

catch(Exception e) {

System.out.println(e);

}

}

void generateSymmetricKey() {

try {

Random r = new Random();

int num = r.nextInt(10000);

String knum = String.valueOf(num);

byte[] knumb = knum.getBytes();

skey=getRawKey(knumb);

skeyString = new String(skey);

System.out.println("DES Symmetric key = "+skeyString);

}

catch(Exception e) {

System.out.println(e);

}

}

private static byte[] getRawKey(byte[] seed) throws Exception {

KeyGenerator kgen = KeyGenerator.getInstance("DES");

SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");

sr.setSeed(seed);

kgen.init(56, sr);

SecretKey skey = kgen.generateKey();

raw = skey.getEncoded();

return raw;

}

private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {

SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");

Cipher cipher = Cipher.getInstance("DES");

cipher.init(Cipher.ENCRYPT\_MODE, skeySpec);

byte[] encrypted = cipher.doFinal(clear);

return encrypted;

}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {

SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");

Cipher cipher = Cipher.getInstance("DES");

cipher.init(Cipher.DECRYPT\_MODE, skeySpec);

byte[] decrypted = cipher.doFinal(encrypted);

return decrypted;

}

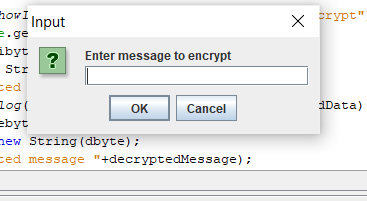
public static void main(String args[]) {

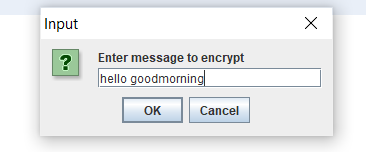
DES des = new DES();

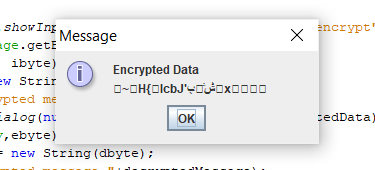
}

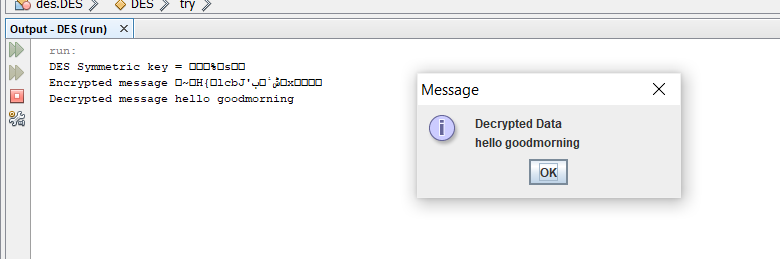
}

**Output :**









1. **AES Algorithm**

**Input:**

import java.security.MessageDigest;

import java.util.Arrays;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.Cipher;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

public class AES {

static String IV = "AAAAAAAAAAAAAAAA";

static String plaintext = "test text 123\0\0\0"; /\*Note null padding\*/

static String encryptionKey = "0123456789abcdef";

public static void main(String [] args) {

try {

System.out.println("==Java==");

System.out.println("plain: " + plaintext);

byte[] cipher = encrypt(plaintext, encryptionKey);

System.out.print("cipher: ");

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

System.out.print(new Integer(cipher[i])+" ");

System.out.println("");

String decrypted = decrypt(cipher, encryptionKey);

System.out.println("decrypt: " + decrypted);

} catch (Exception e) {

e.printStackTrace();

}

}

public static byte[] encrypt(String plainText, String encryptionKey) throws Exception {

Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding", "SunJCE");

SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"), "AES");

cipher.init(Cipher.ENCRYPT\_MODE, key,new IvParameterSpec(IV.getBytes("UTF-8")));

return cipher.doFinal(plainText.getBytes("UTF-8"));

}

public static String decrypt(byte[] cipherText, String encryptionKey) throws Exception{

Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding", "SunJCE");

SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"), "AES");

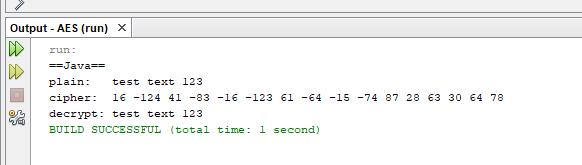
cipher.init(Cipher.DECRYPT\_MODE, key,new IvParameterSpec(IV.getBytes("UTF-8")));

return new String(cipher.doFinal(cipherText),"UTF-8");

}

}

**Output:**



**Practical No: 05**

**Aim: Write a program to implement RSA algorithm to perform encryption / decryption of a given string.**

**Input:**

import java.util.\*;

class RSA

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

int p,q,n,z,d=0,e,i;

System.out.println("Enter the number to be encrypted and decrypted");

int msg=sc.nextInt();

double c,msgback;

System.out.println("Enter 1st prime number p");

p=sc.nextInt();

System.out.println("Enter 2nd prime number q");

q=sc.nextInt();

n=p\*q;

z=(p-1)\*(q-1);

System.out.println("the value of z = "+z);

for(e=2;e<z;e++)

{

if(gcd(e,z)==1) // e is for public key exponent

{

break;

}

}

System.out.println("the value of e = "+e);

for(i=0;i<=9;i++)

{

int x=1+(i\*z);

if(x%e==0) //d is for private key exponent

{

d=x/e;

break;

}

}

System.out.println("the value of d = "+d);

c=(Math.pow(msg,e))%n;

System.out.println("Encrypted message is : -");

System.out.println(c);

msgback=(Math.pow(c,d))%n;

System.out.println("Derypted message is : -");

System.out.println(msgback);

}

static int gcd(int e, int z)

{

if(e==0)

return z;

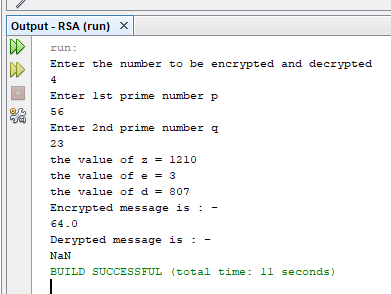
else

return gcd(z%e,e);

}

}

**Output :**



**Practical No: 06**

**Aim: Write a program to implement the Diffie-Hellman Key Agreement algorithm to generate symmetric keys.**

**Input:**

import java.util.\*;

class Diffie\_Hellman

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter modulo(p)");

int p=sc.nextInt();

System.out.println("Enter primitive root of "+p);

int g=sc.nextInt();

System.out.println("Choose 1st secret no(Alice)");

int a=sc.nextInt();

System.out.println("Choose 2nd secret no(BOB)");

int b=sc.nextInt();

int A = (int)Math.pow(g,a)%p;

int B = (int)Math.pow(g,b)%p;

int S\_A = (int)Math.pow(B,a)%p;

int S\_B =(int)Math.pow(A,b)%p;

if(S\_A==S\_B)

{

System.out.println("ALice and Bob can communicate with each other!!!");

System.out.println("They share a secret no = "+S\_A);

}

else

{

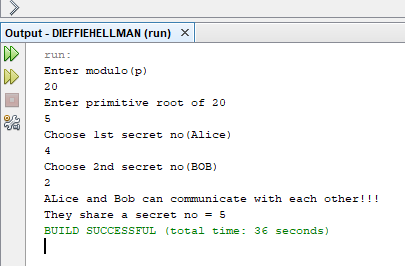
System.out.println("ALice and Bob cannot communicate with each other!!!");

}

}

}

**Output:**



**Practical No : 07**

**Aim: Write a program to implement the MD5 algorithm compute the message digest.**

**Input:**

package javamd5hash;

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class JavaMD5Hash {

public static void main(String[] args) {

System.out.println("For null " + md5(""));

System.out.println("For simple text "+ md5("This is my text"));

System.out.println("For simple numbers " + md5("12345"));

}

public static String md5(String input) {

String md5 = null;

if(null == input) return null;

try {

MessageDigest digest = MessageDigest.getInstance("MD5");

digest.update(input.getBytes(), 0, input.length());

md5 = new BigInteger(1, digest.digest()).toString(16);

}

catch (NoSuchAlgorithmException e) {

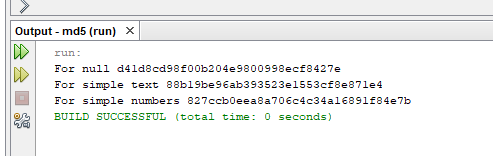
e.printStackTrace(); }

return md5;

}

}

**Output:**



**Practical No: 08**

**Aim: Write a program to calculate HMAC-SHA1 Signature**

**Input:**

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.SignatureException;

import java.util.Formatter;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

public class HmacSha1Signature {

private static final String HMAC\_SHA1\_ALGORITHM = "HmacShA1"; //Message digest algorithm(MD5 & SHA1)

private static String toHexString(byte[] bytes) {

Formatter formatter = new Formatter();

for (byte b : bytes) {

formatter.format("%02x", b);

}

return formatter.toString();

}

public static String calculateRFC2104HMAC(String data, String key)

throws SignatureException, NoSuchAlgorithmException, InvalidKeyException

{

SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC\_SHA1\_ALGORITHM);

Mac mac = Mac.getInstance(HMAC\_SHA1\_ALGORITHM);

mac.init(signingKey);

return toHexString(mac.doFinal(data.getBytes()));

}

public static void main(String[] args) throws Exception {

String hmac = calculateRFC2104HMAC("data", "key");

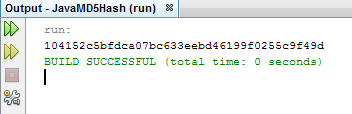
System.out.println(hmac);

assert hmac.equals("104152c5bfdca07bc633eebd46199f0255c9f49d");

}

}

**Output:**



**Practical No: 09**

**Aim: Write a program to implement SSL**

**Input :**

**Output:**

**Practical No : 10**

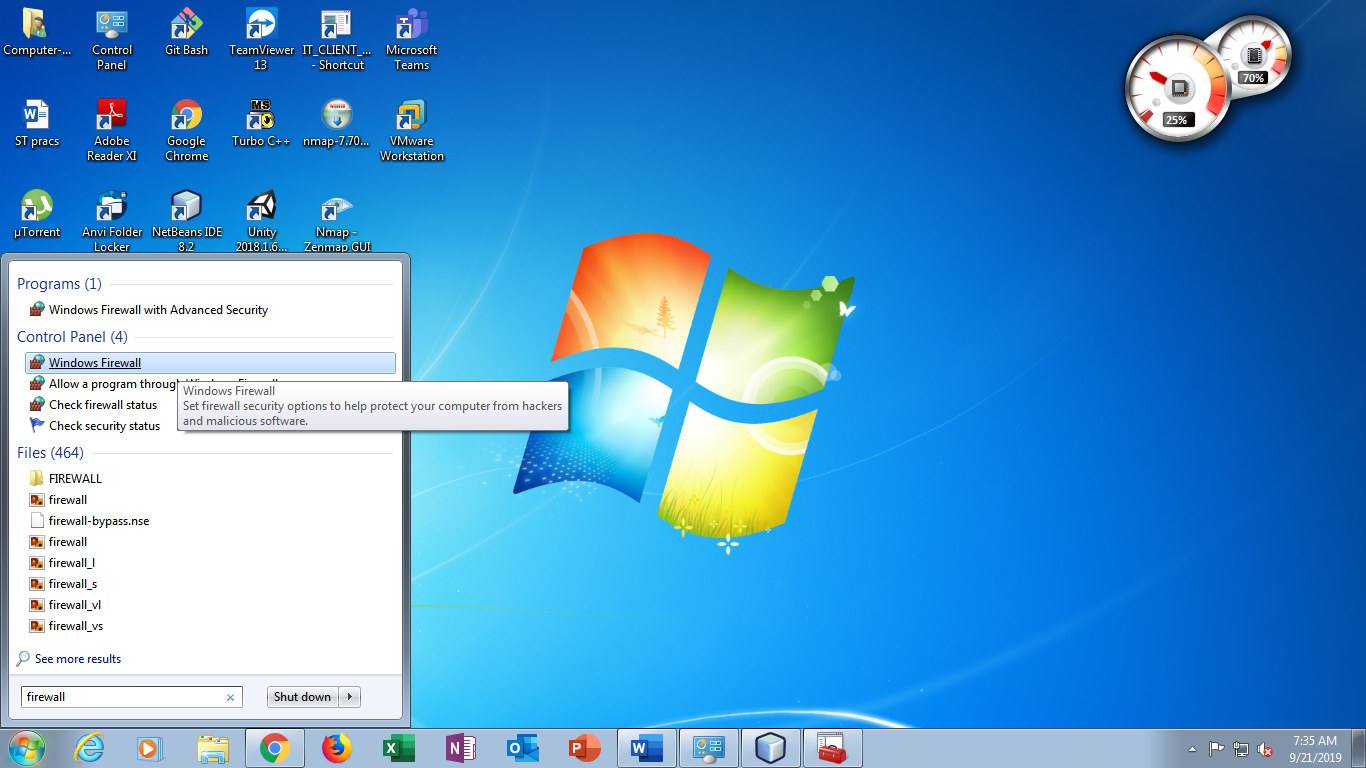
**Aim :Write a program to configure windows firewall to block.**

1. **A Port**
2. **An Program**
3. **A Website**

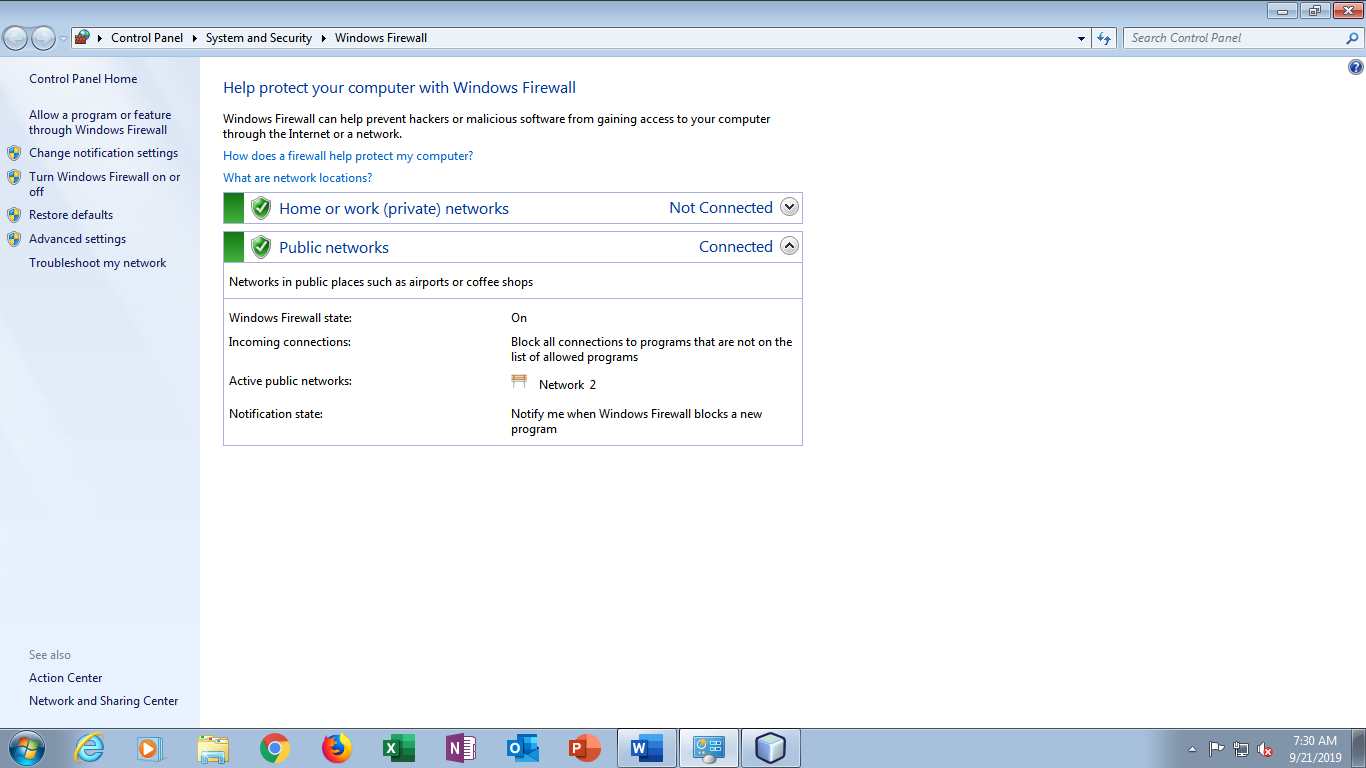
**Output:**

1. **A Port**

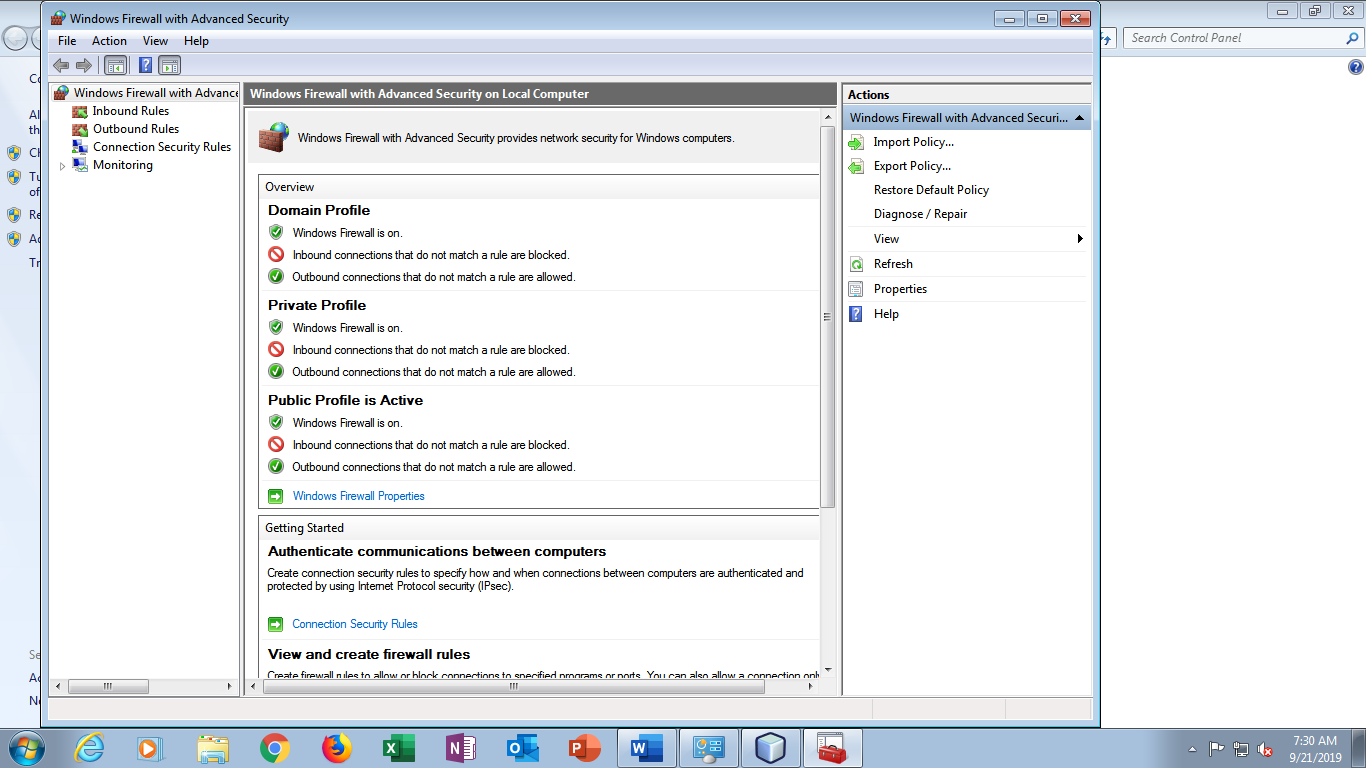
Step 1:



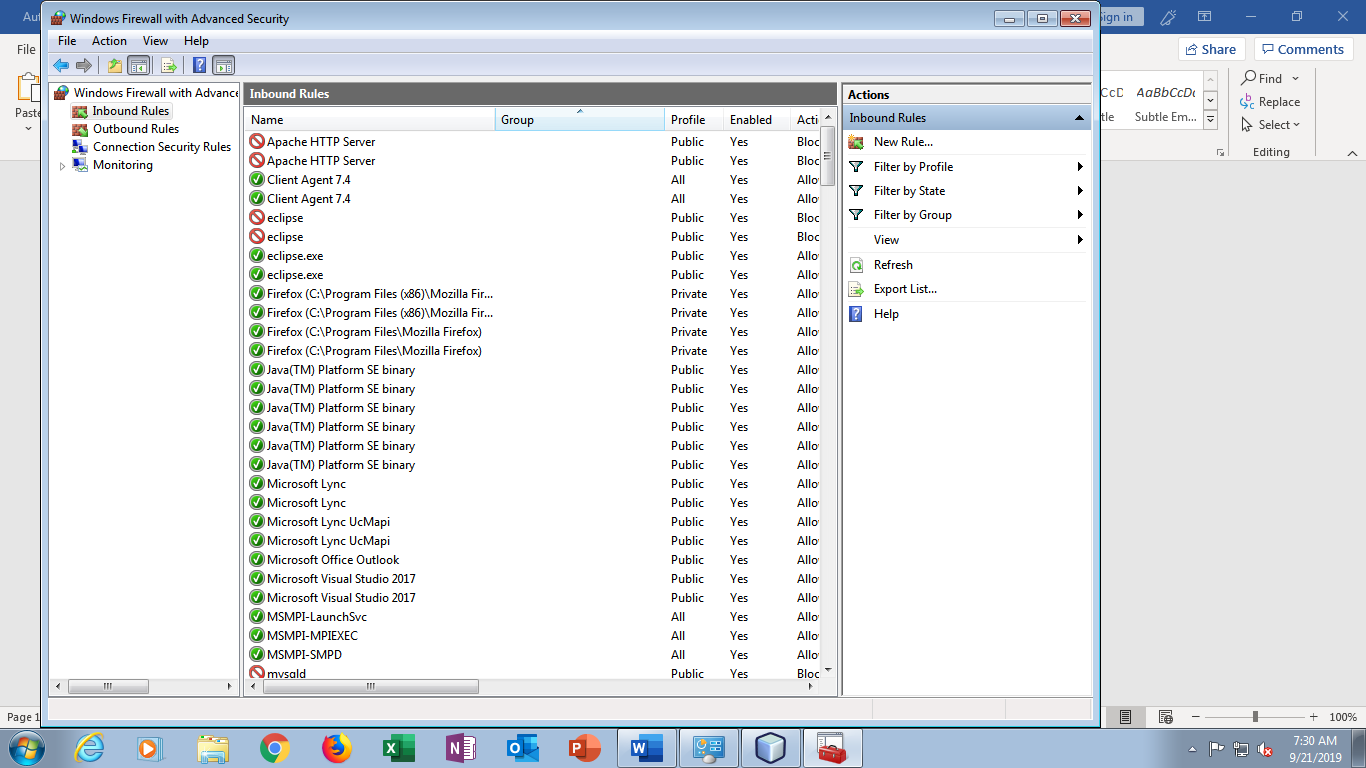
Step 2:



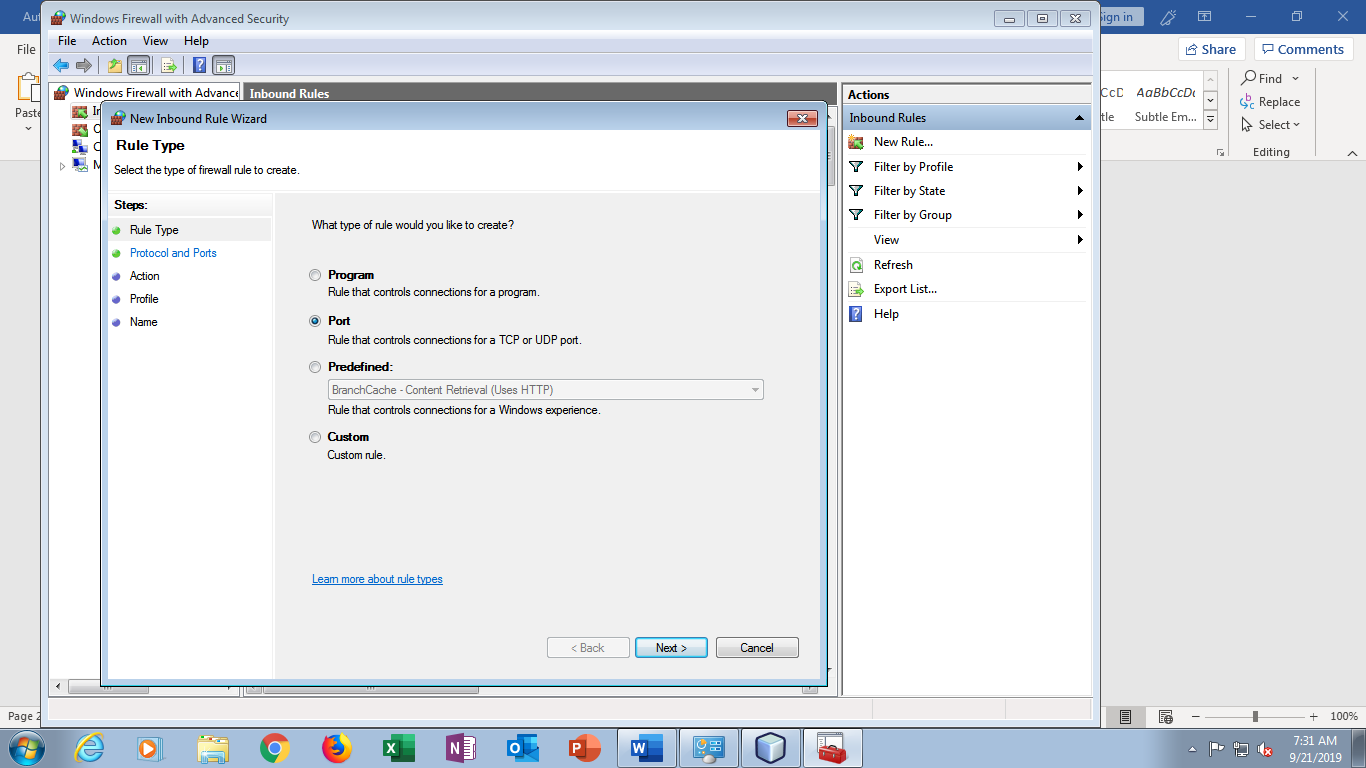
Step 3:



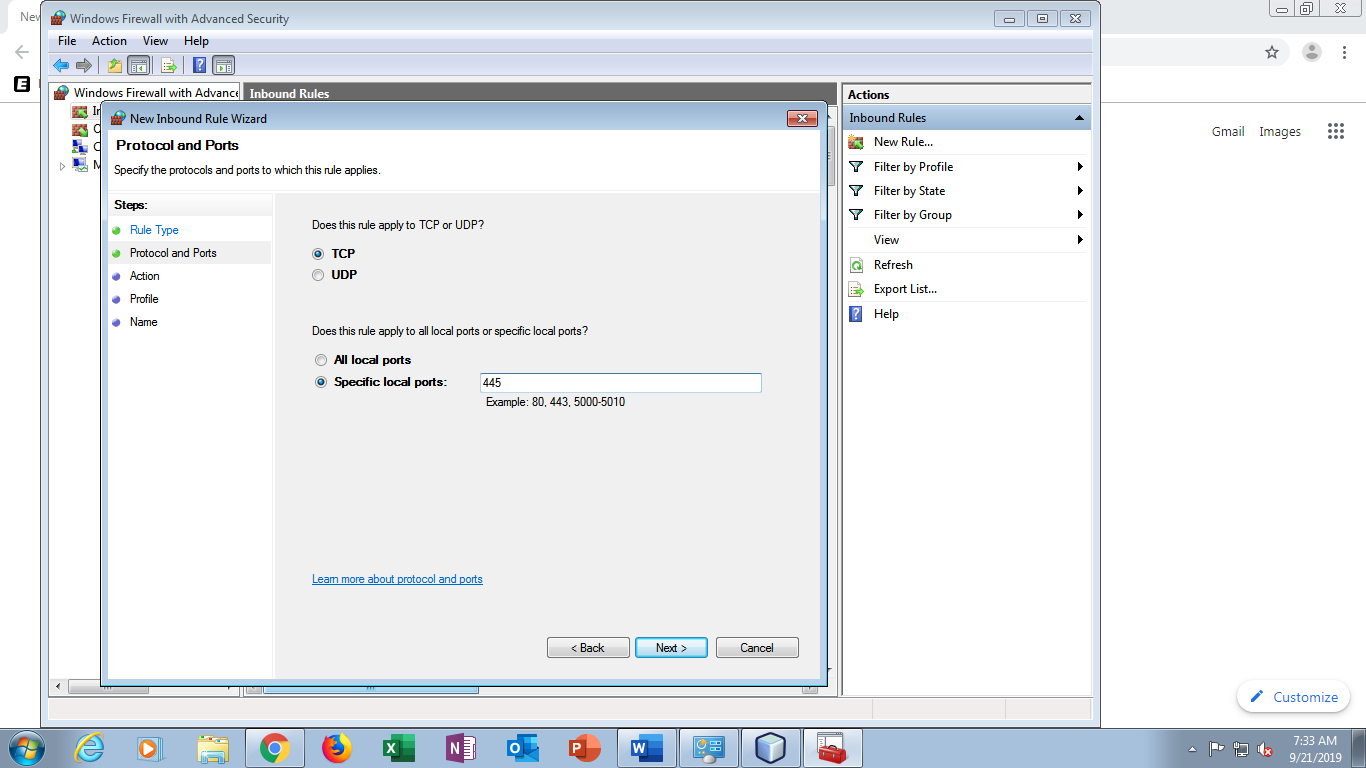
Step 4:



Step 5:



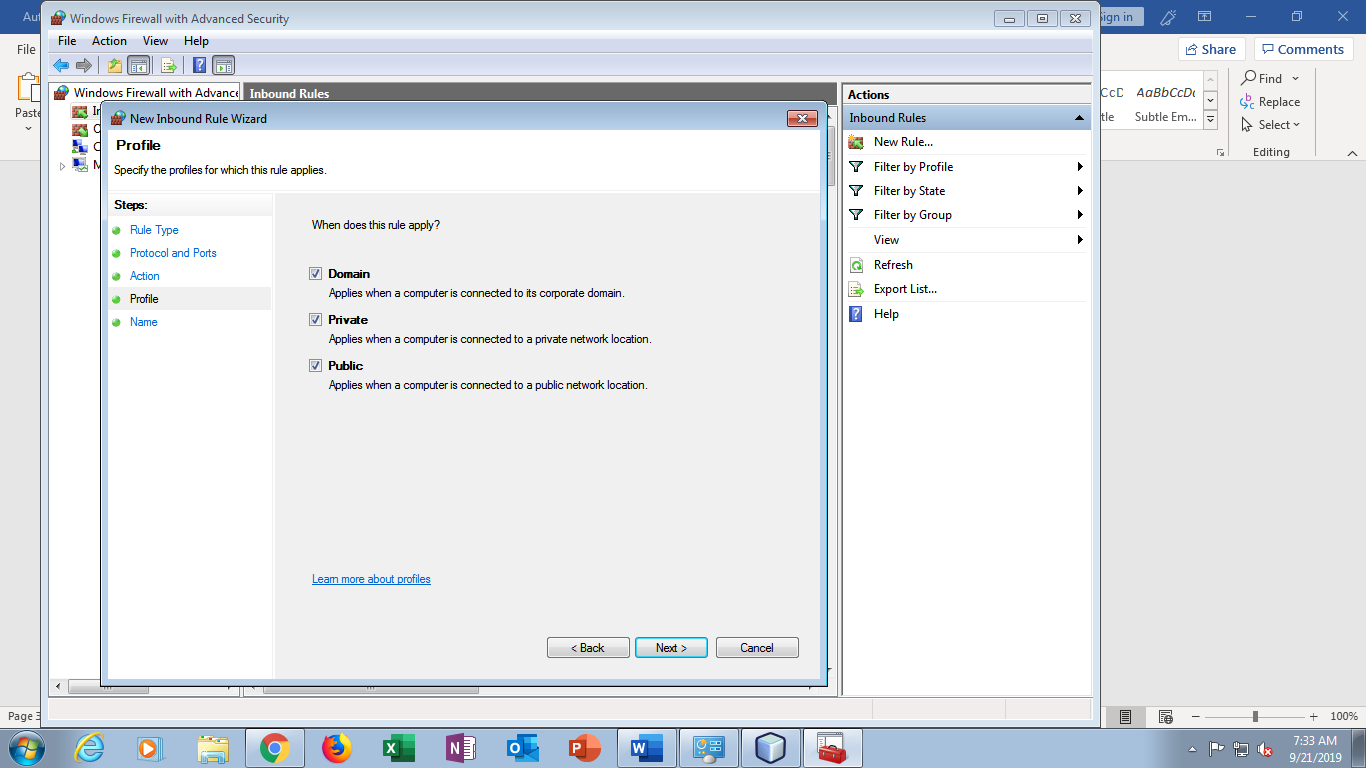
Step 6:



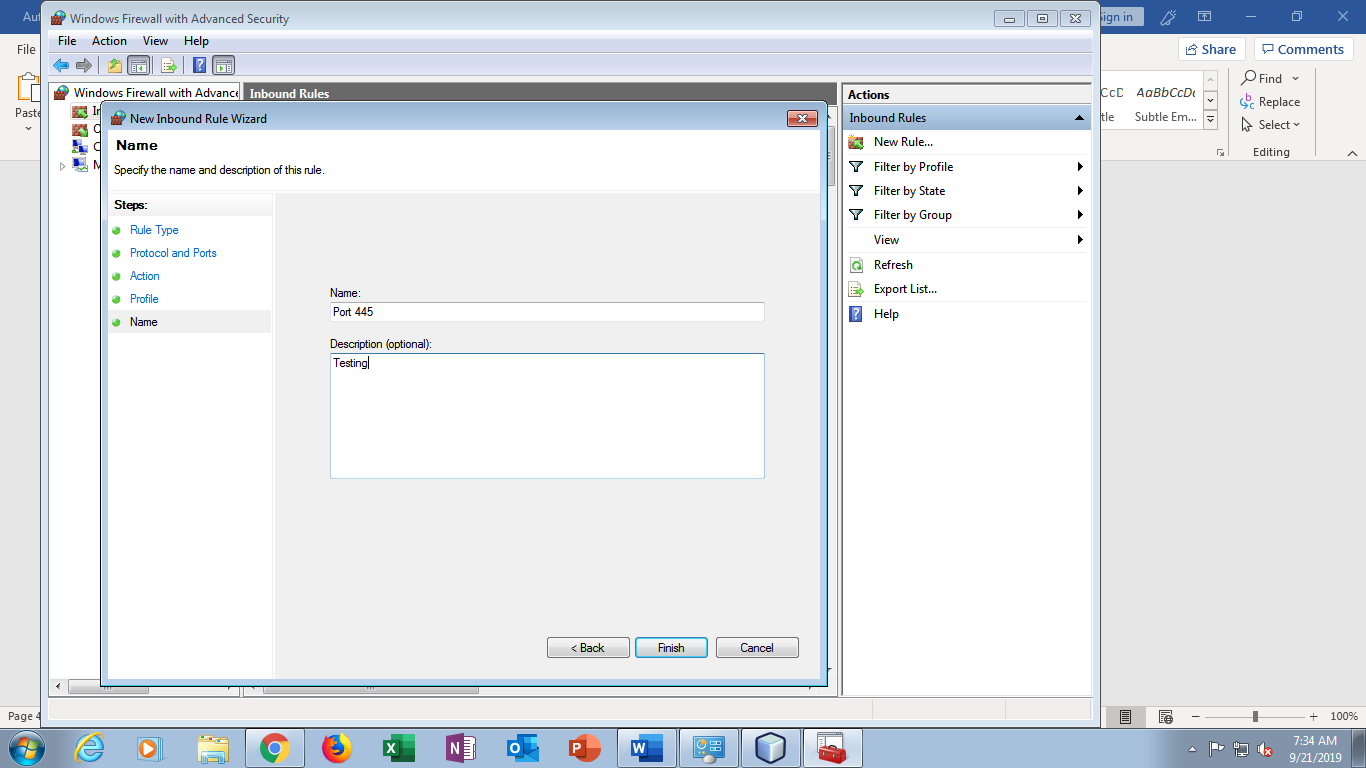
Step 7:



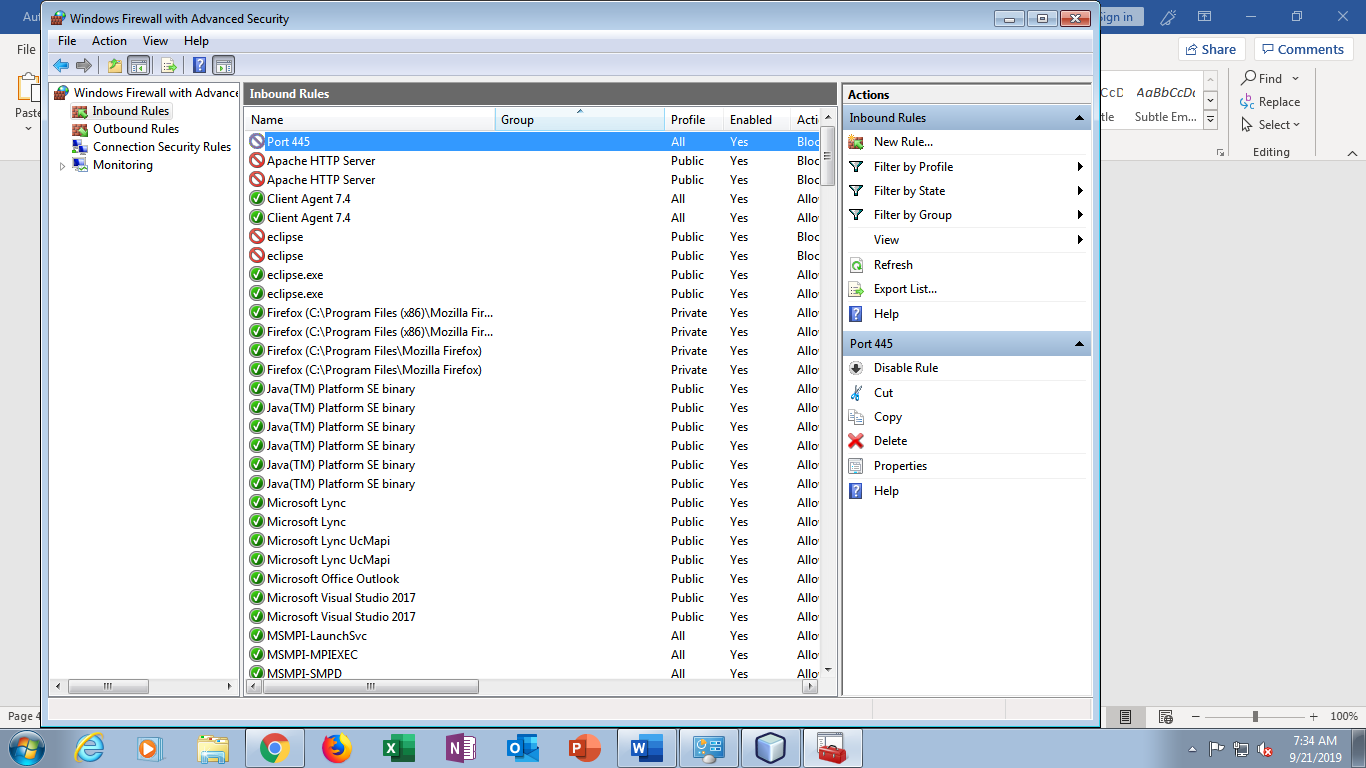
Step 8:



Step 9:

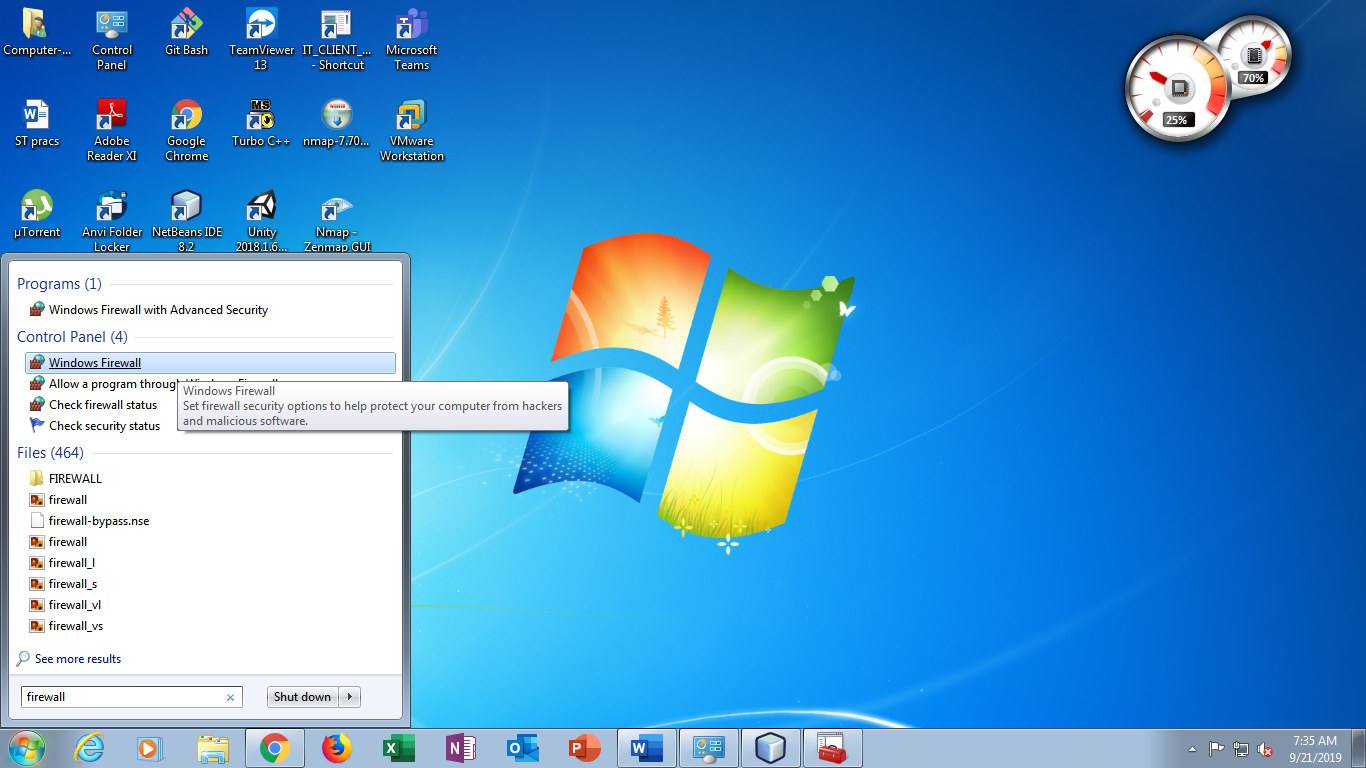


Step 10:

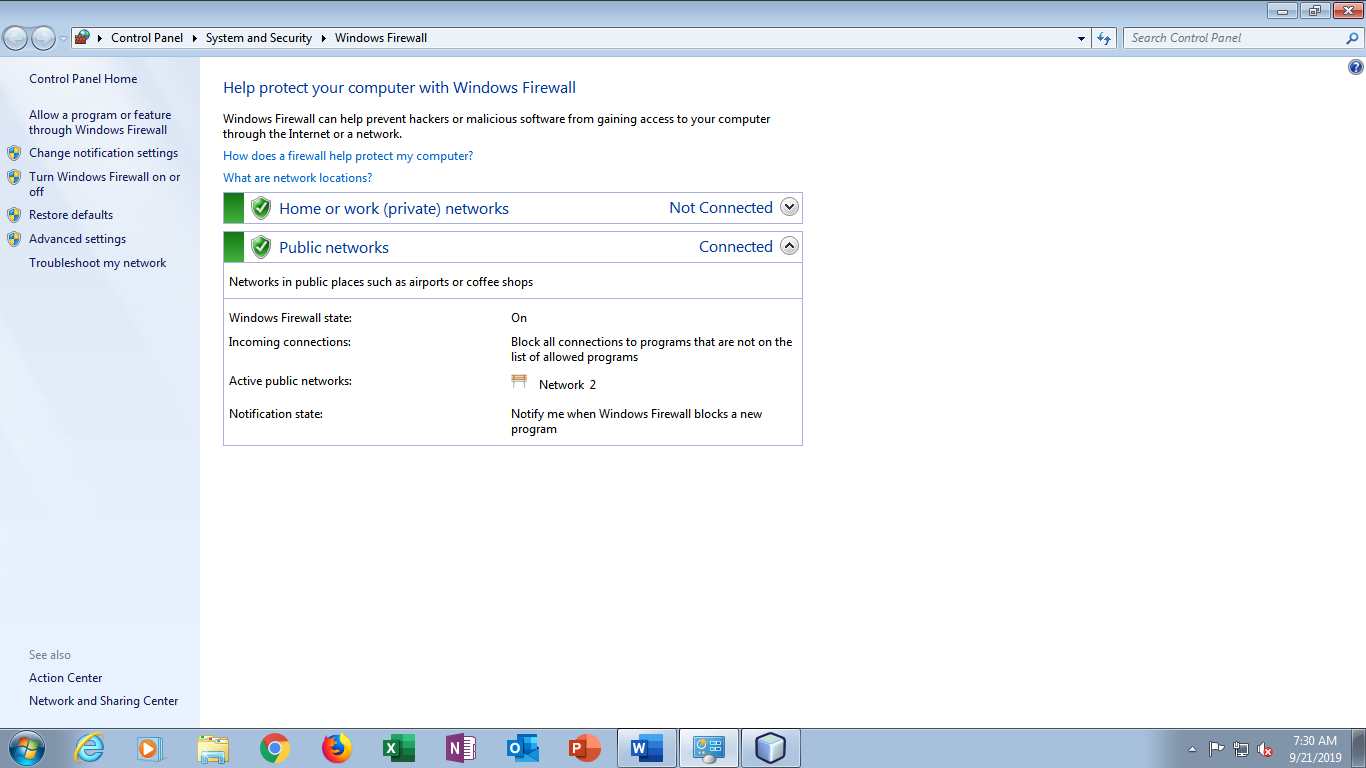


1. **A Program**

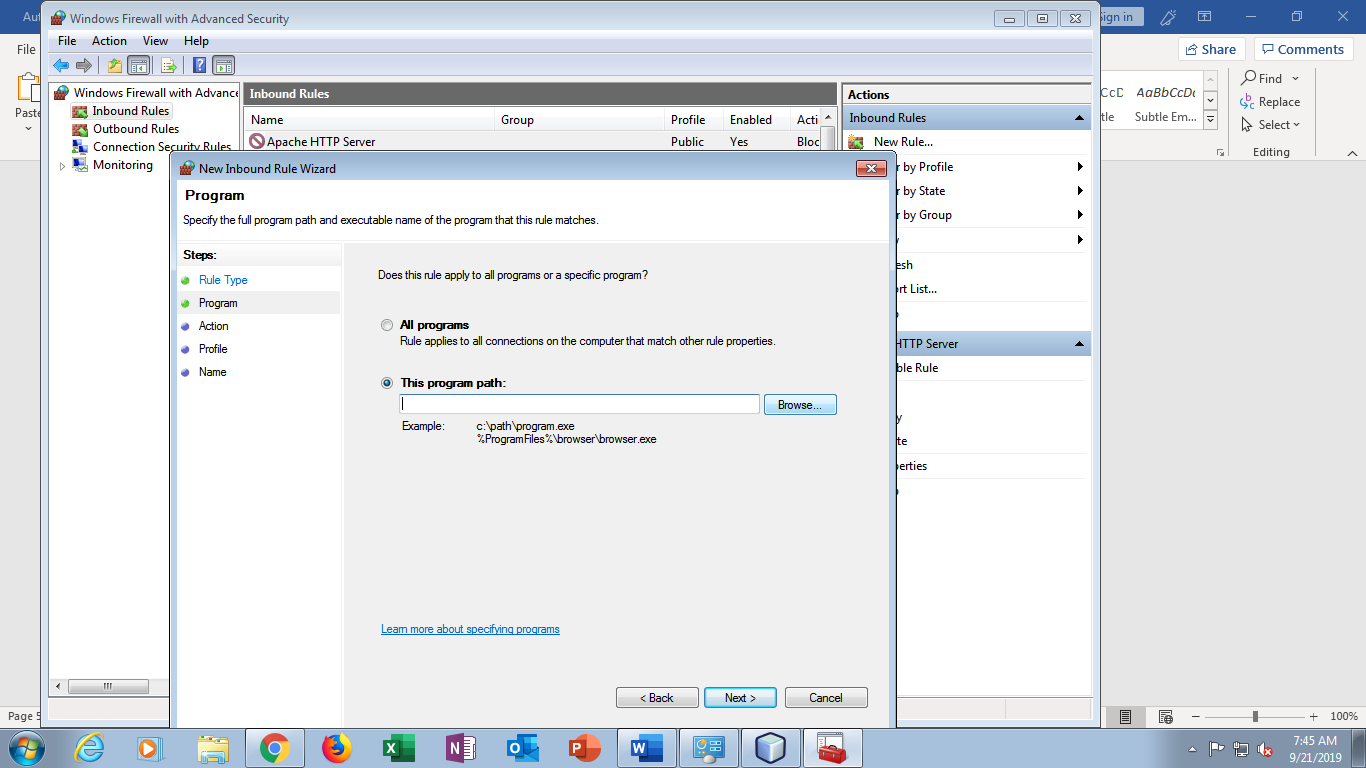
Step 1:



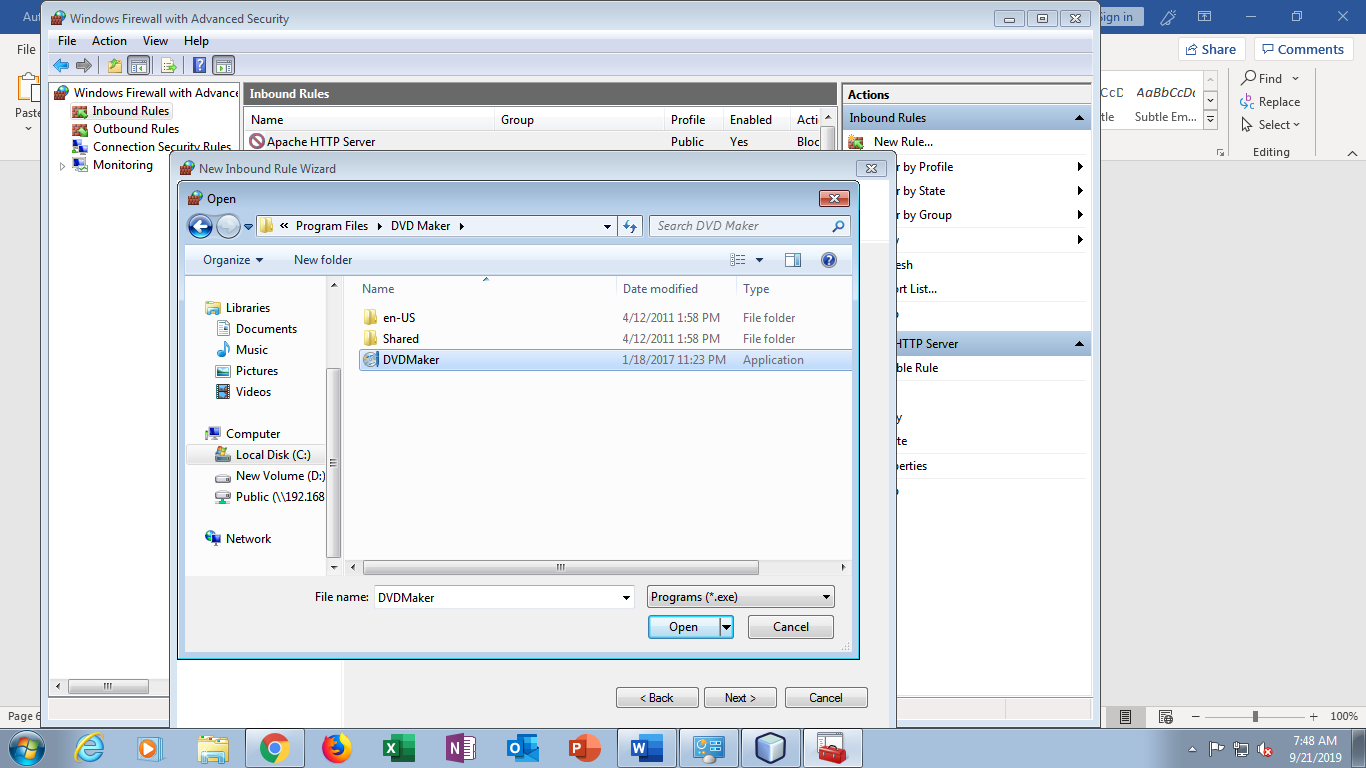
Step 2:



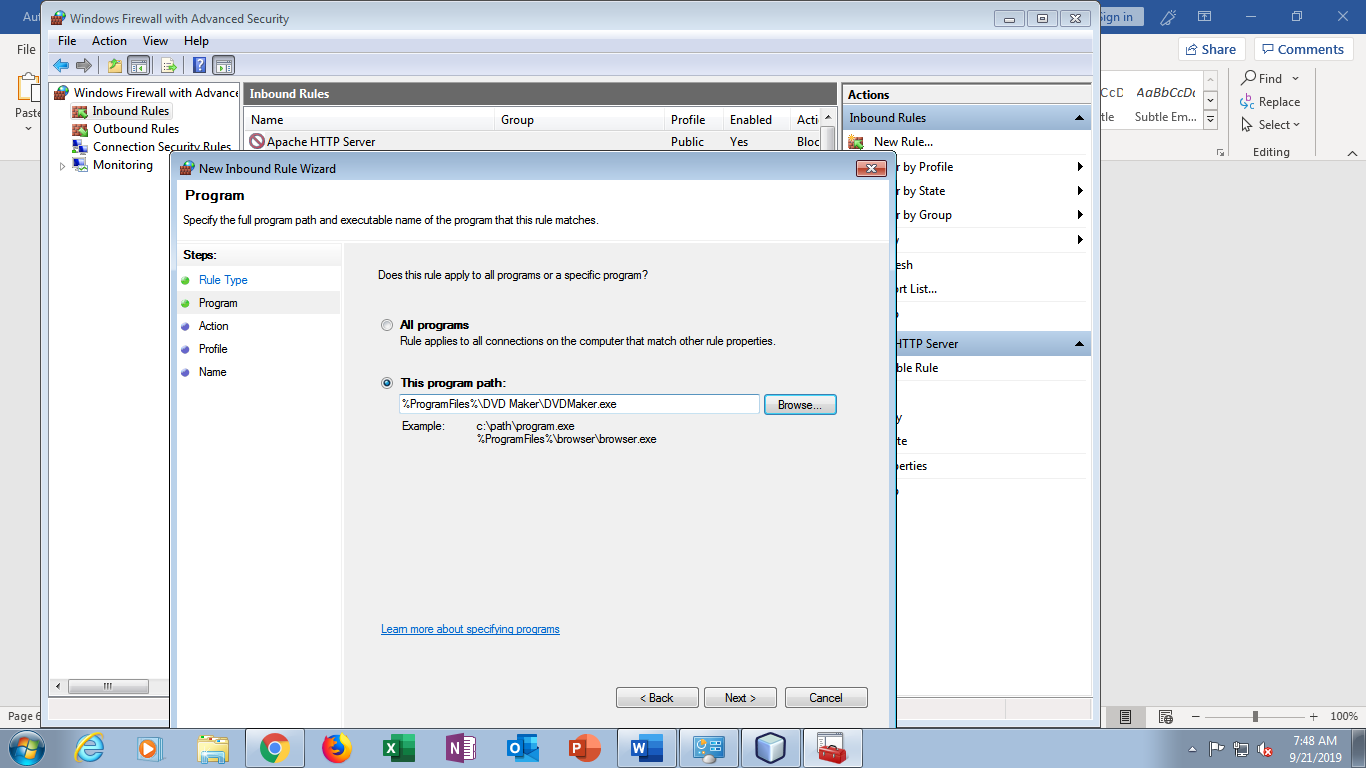
Step 3:



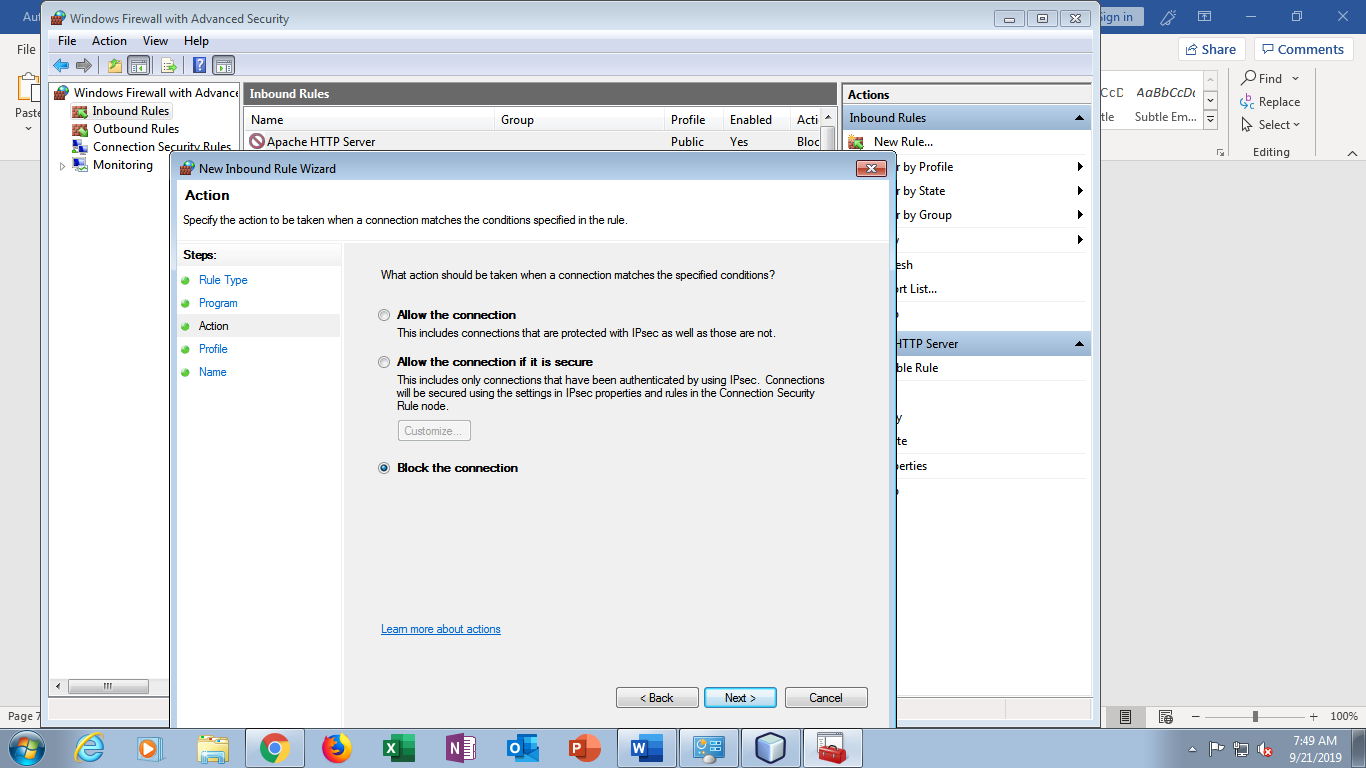
Step 4:



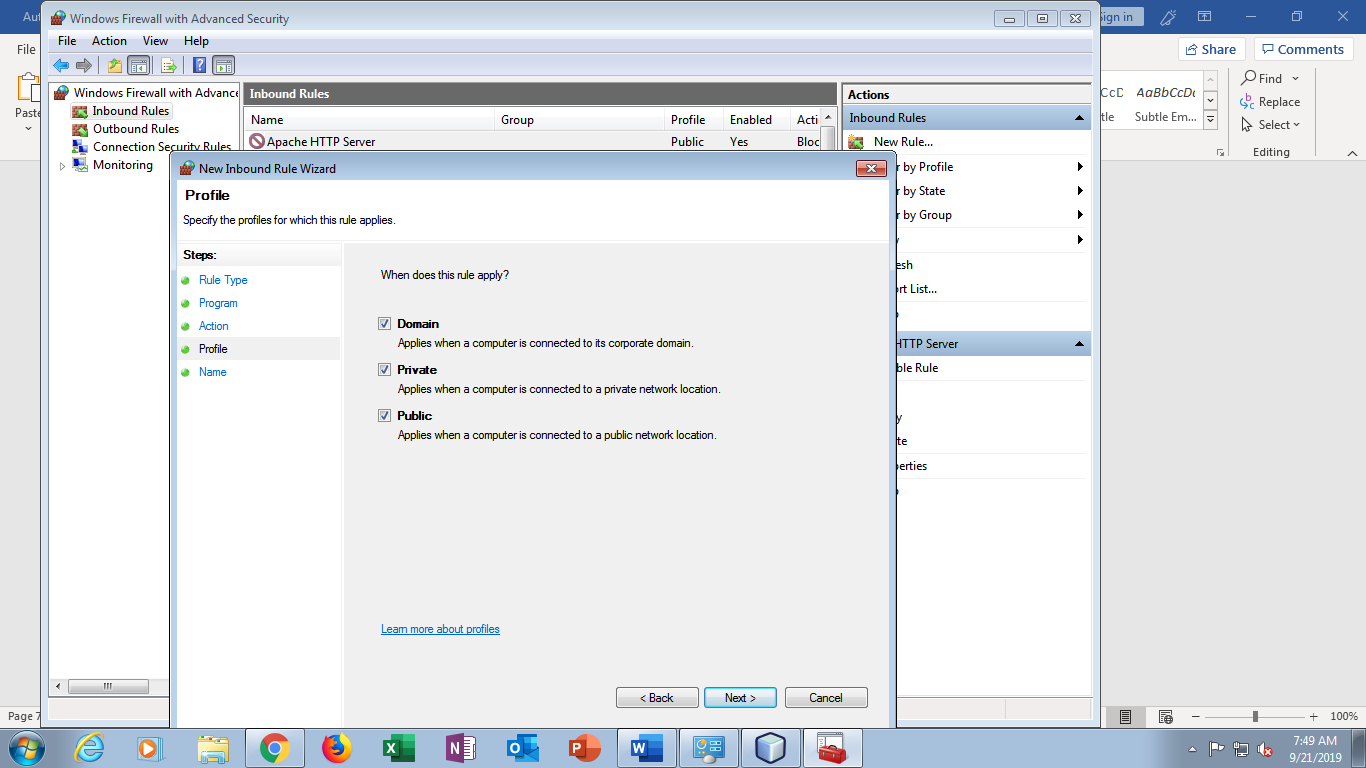
Step 5:



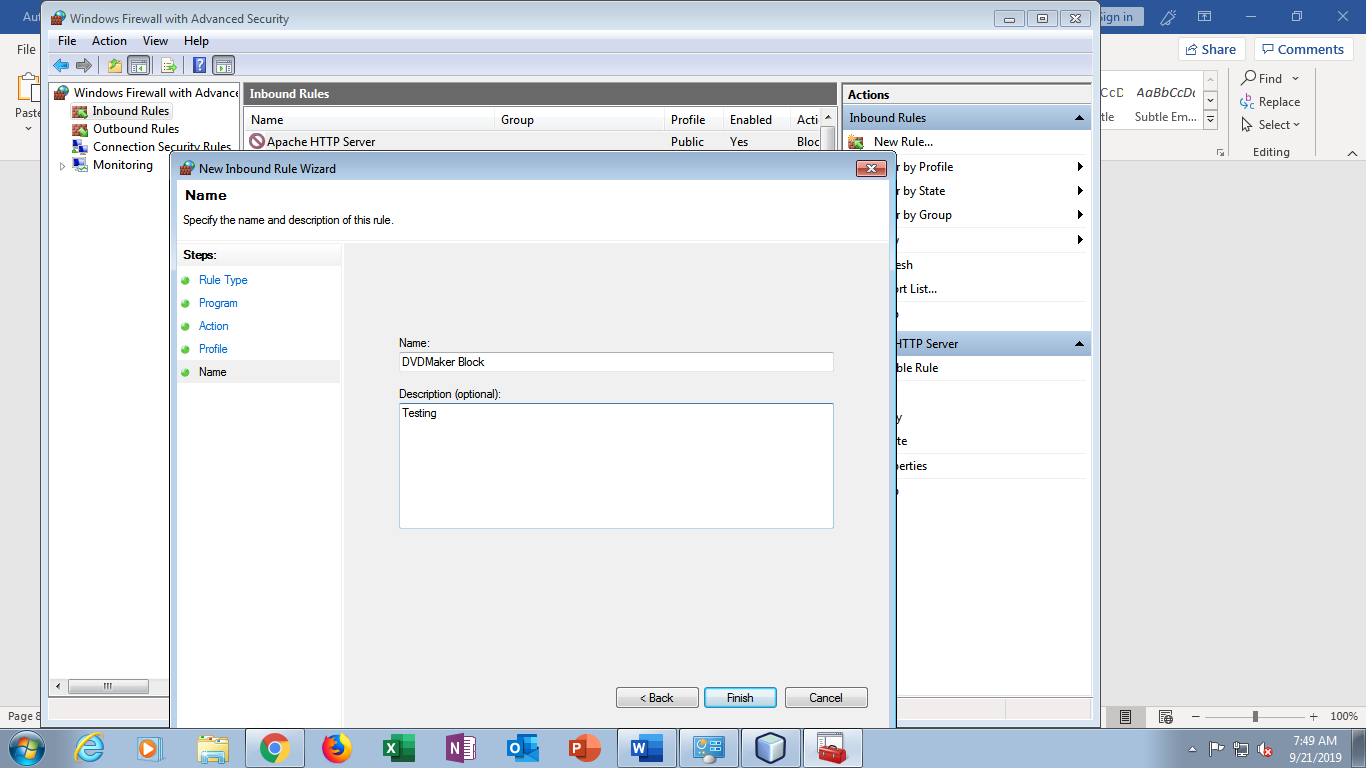
Step 6:



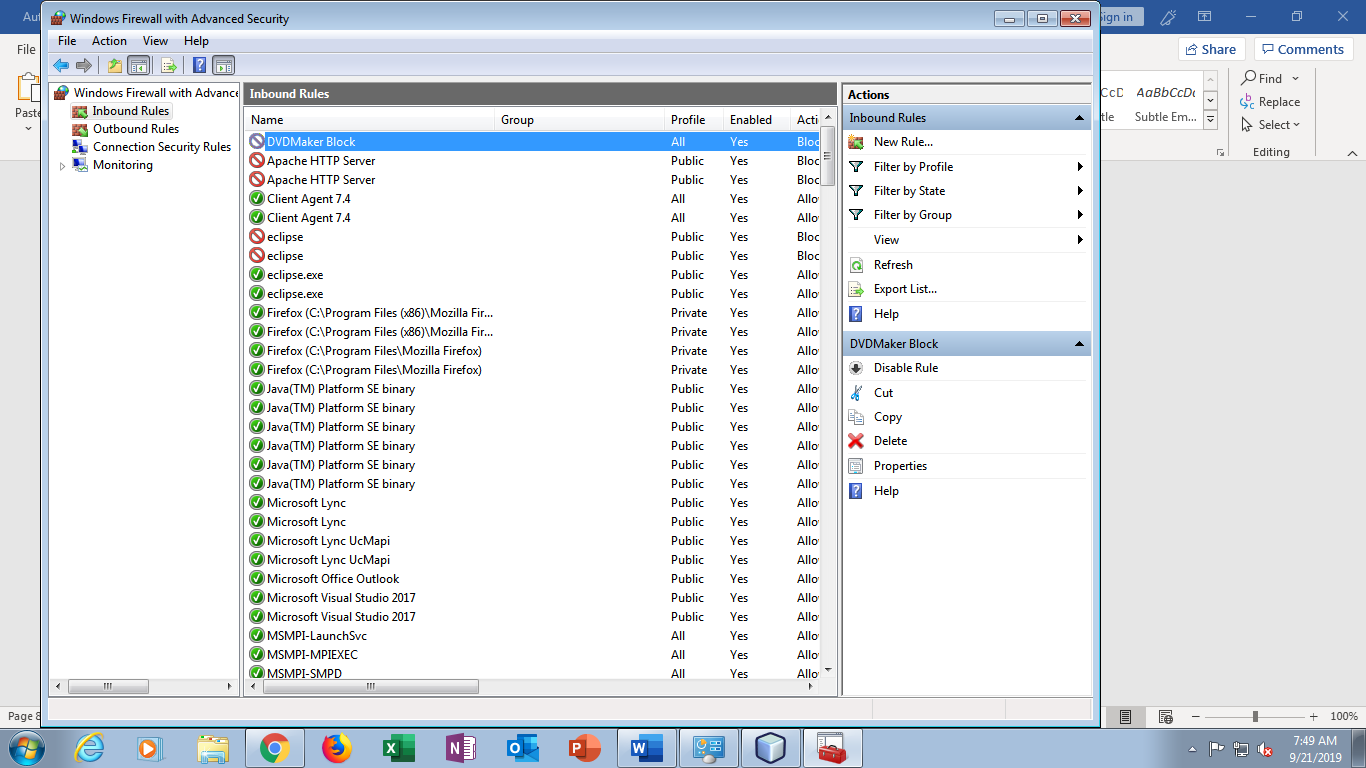
Step 7:



Step 8:

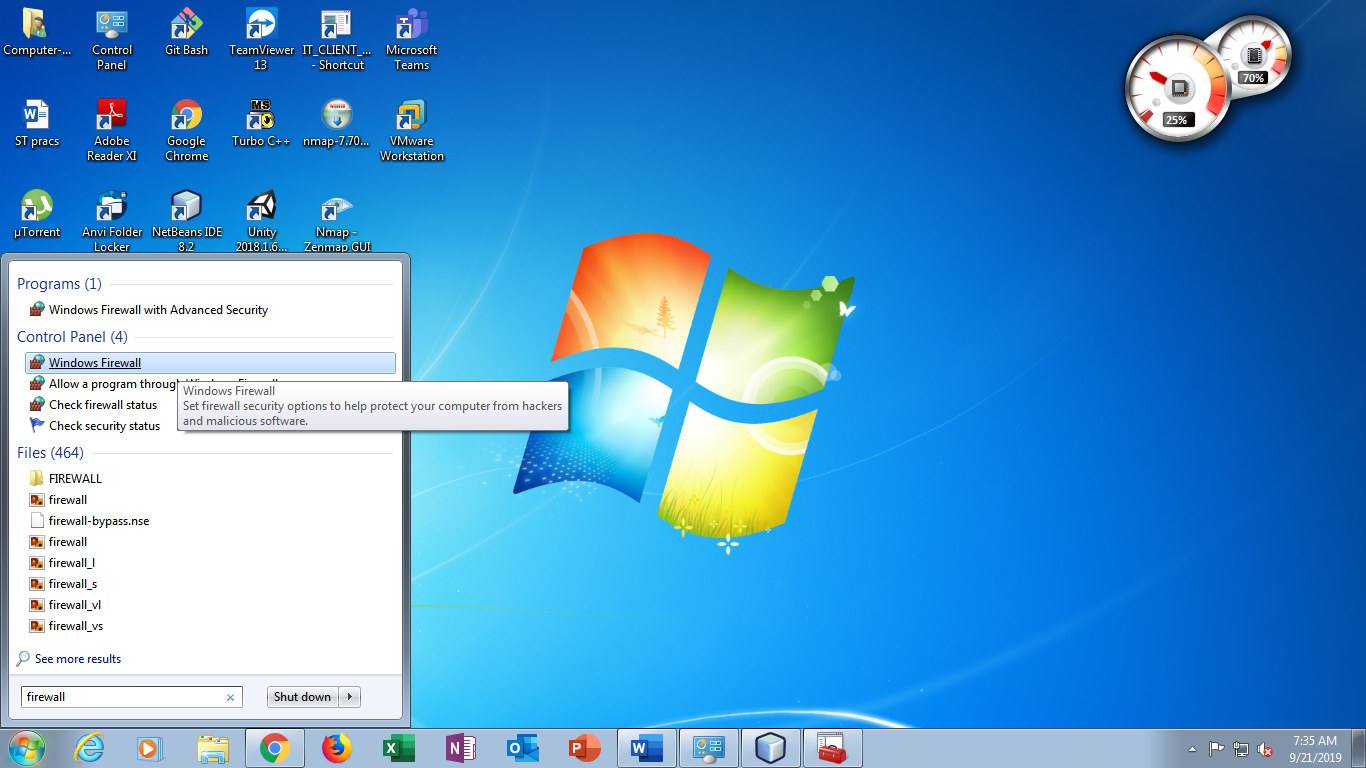


Step 9:

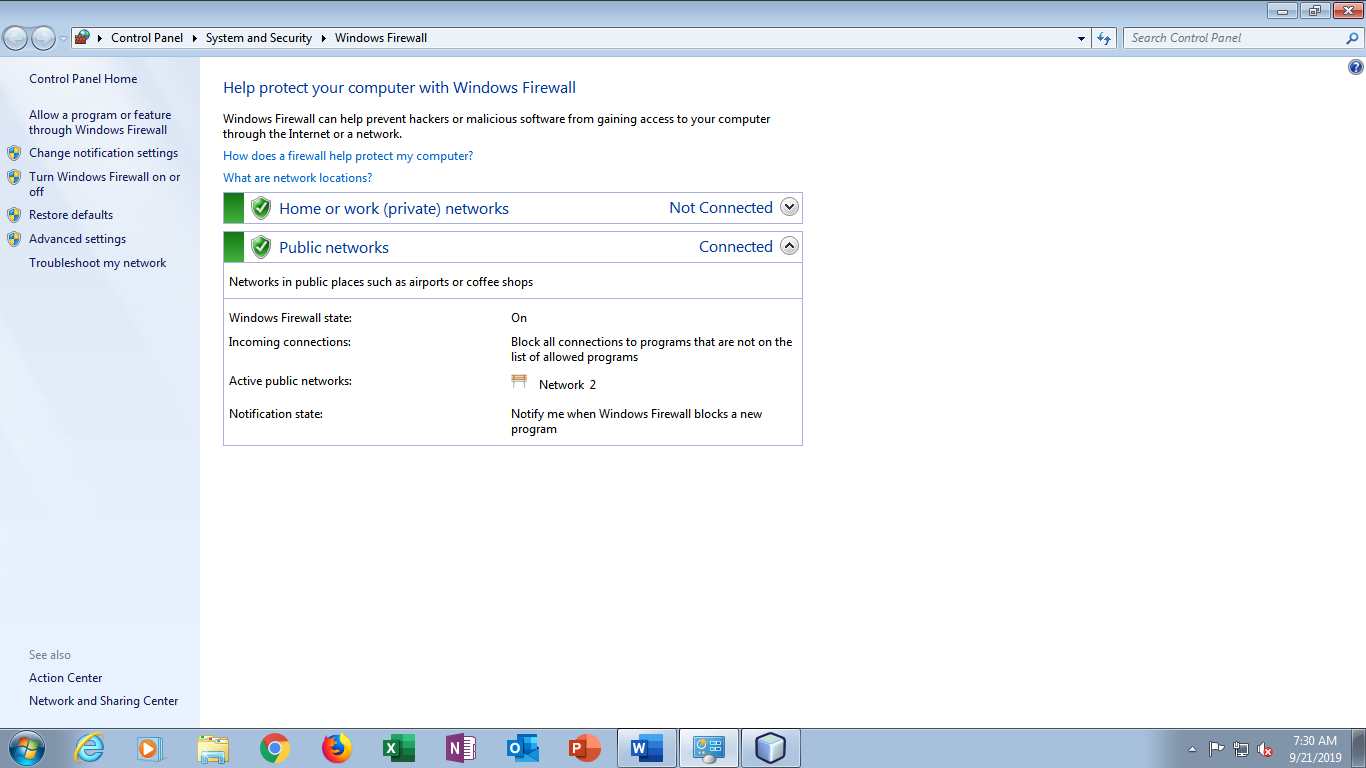


1. **A Website**

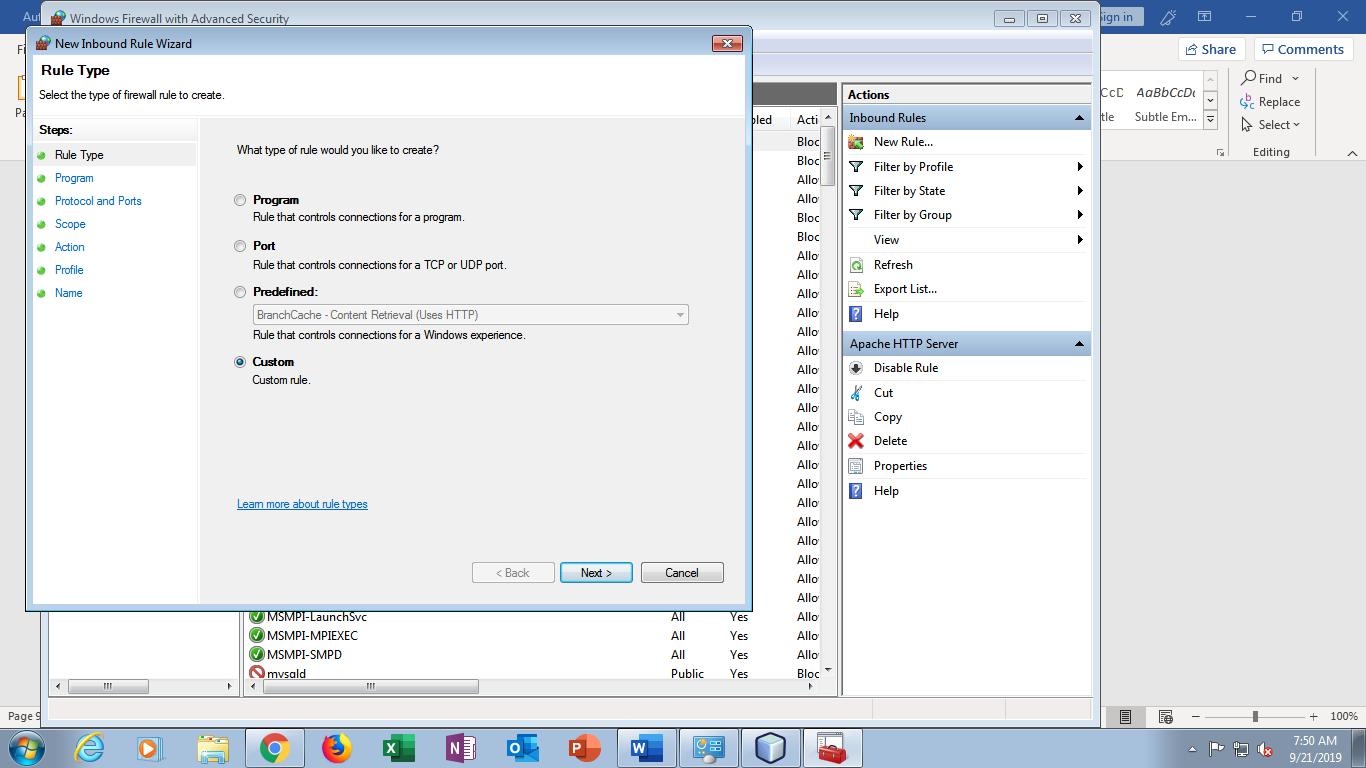
Step 1:



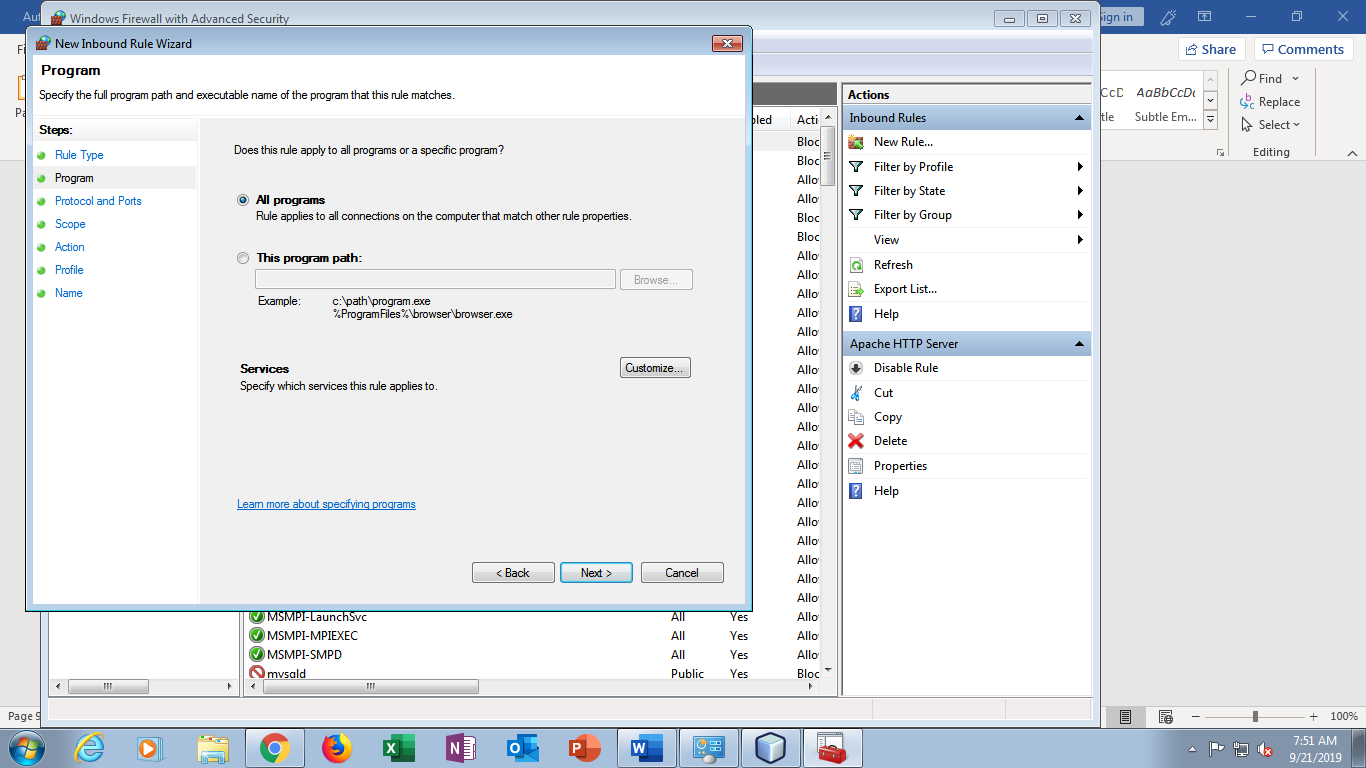
Step 2:



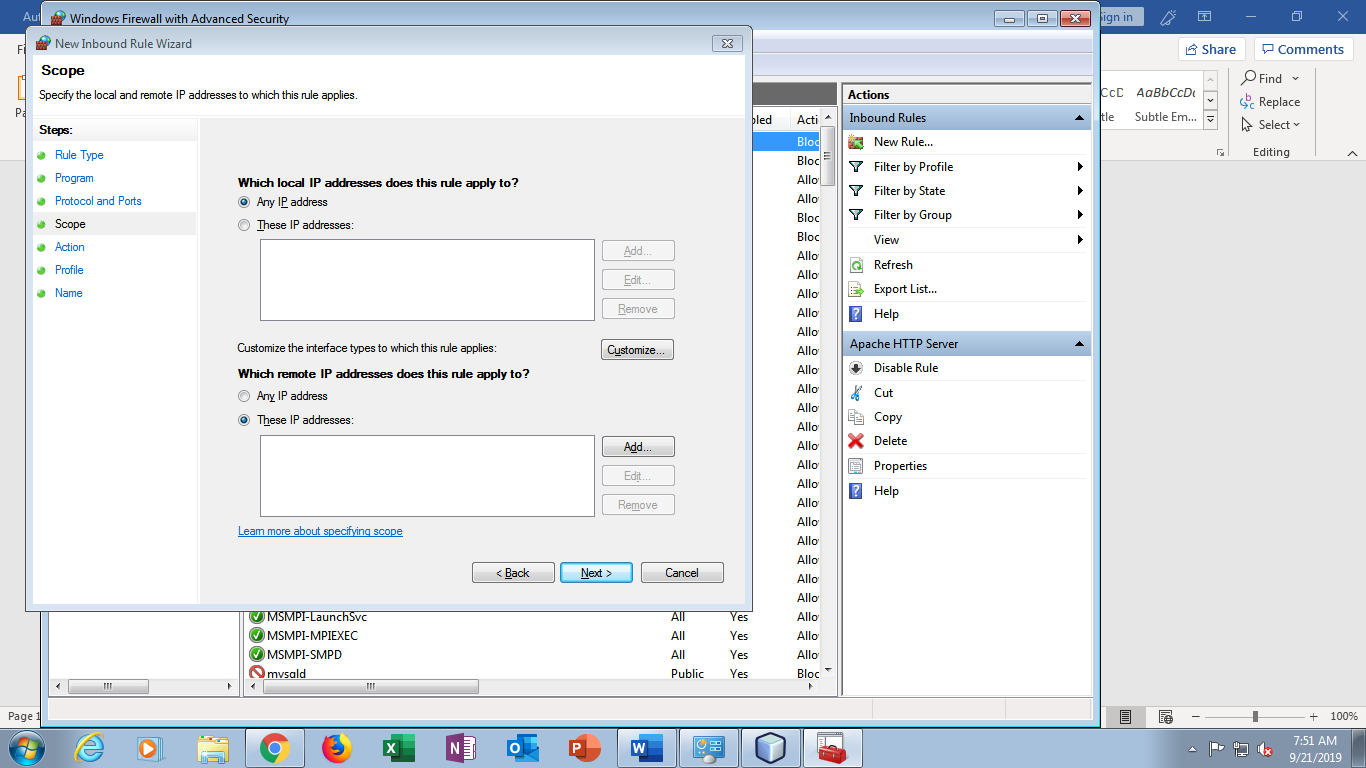
Step 3:



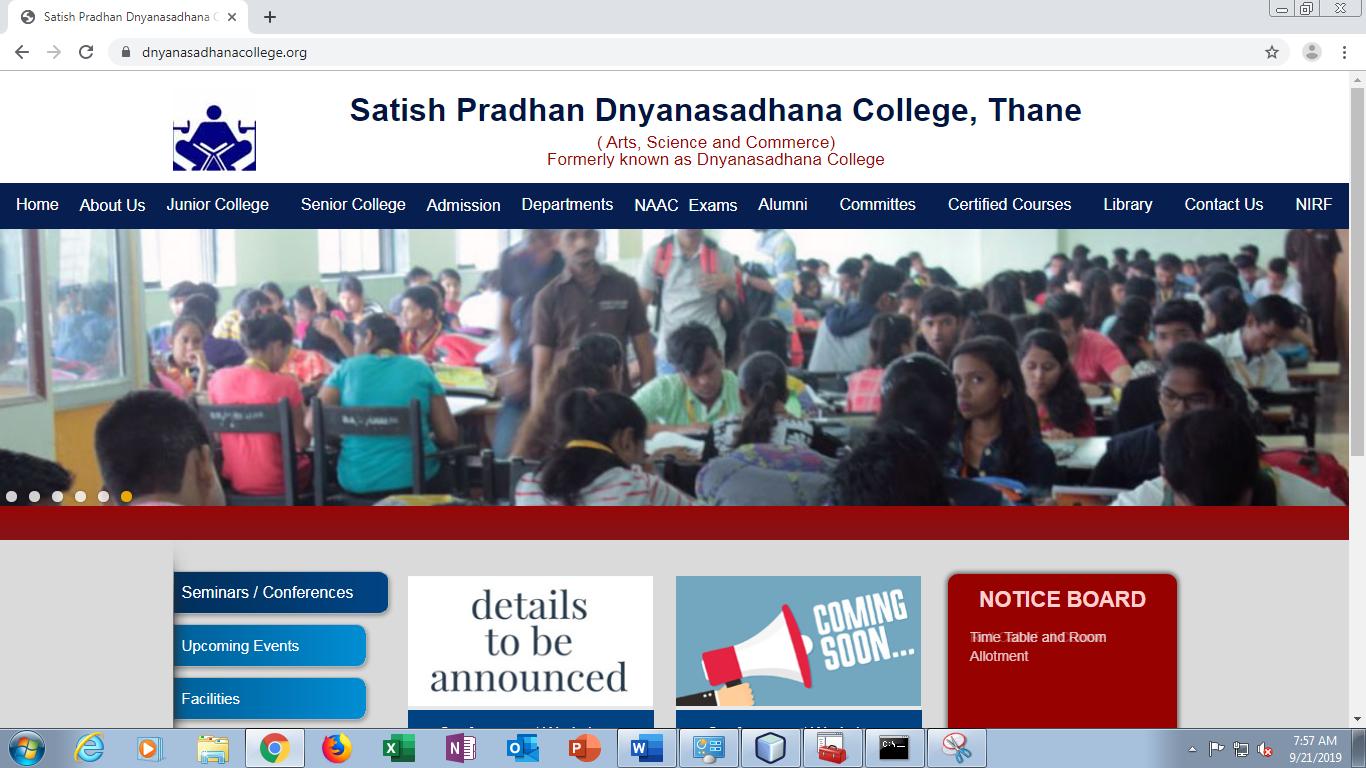
Step 4:



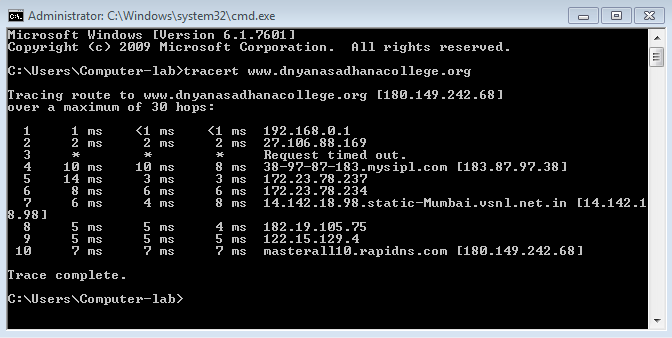
Step 5:



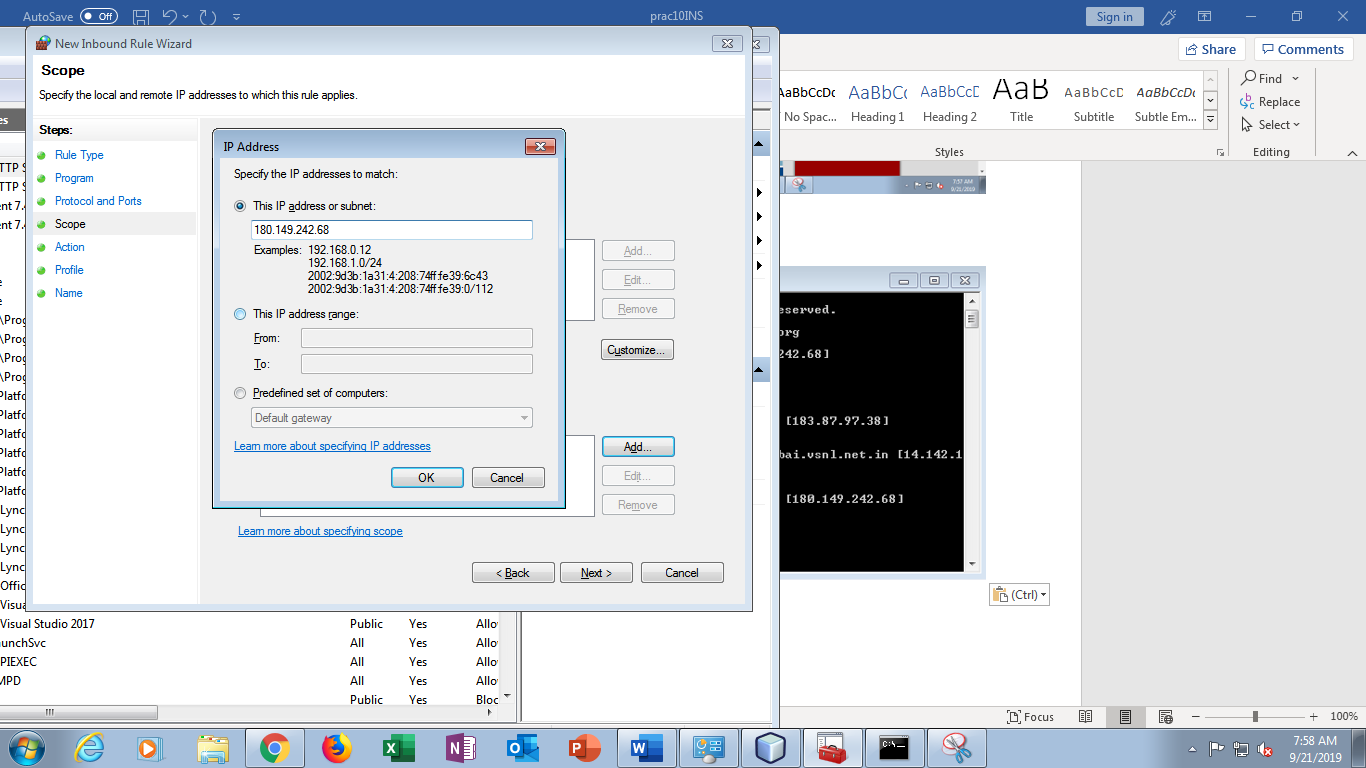
Step 6:



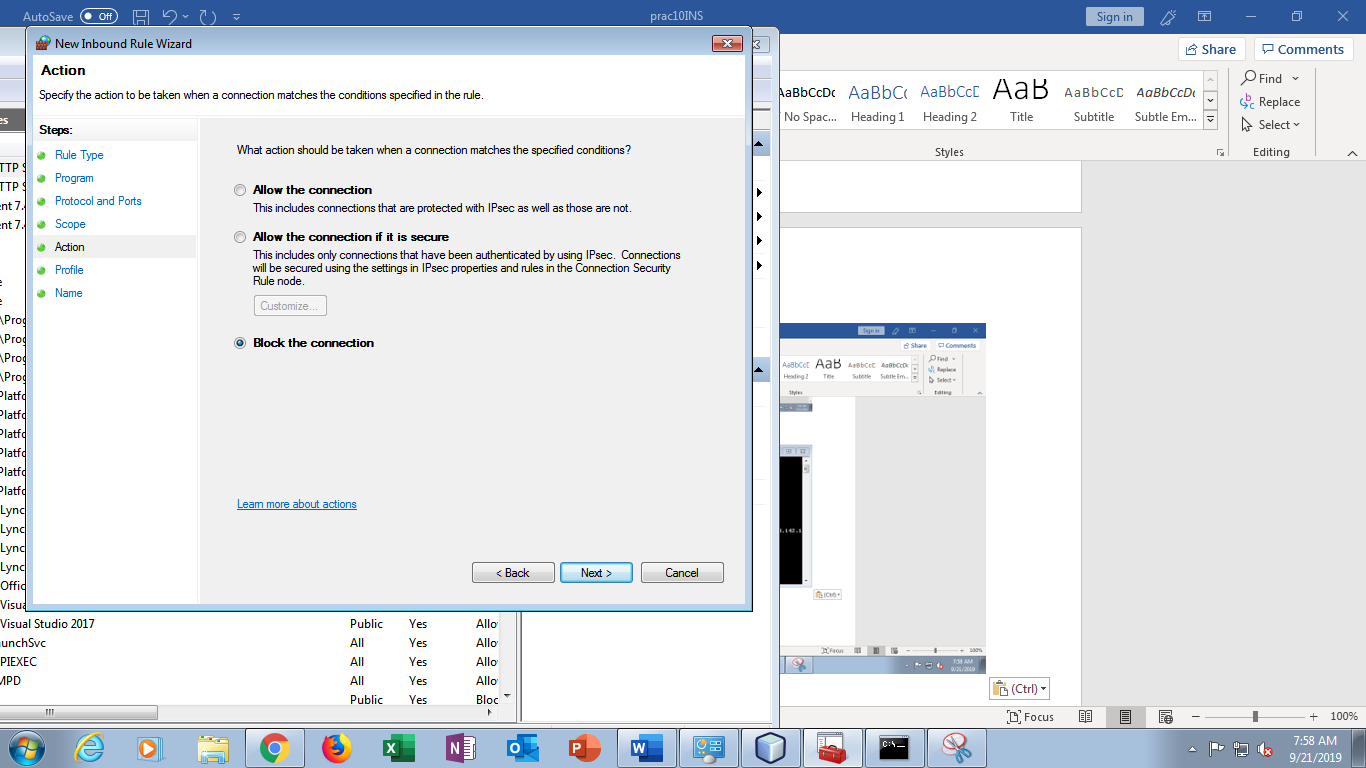
Step 7:



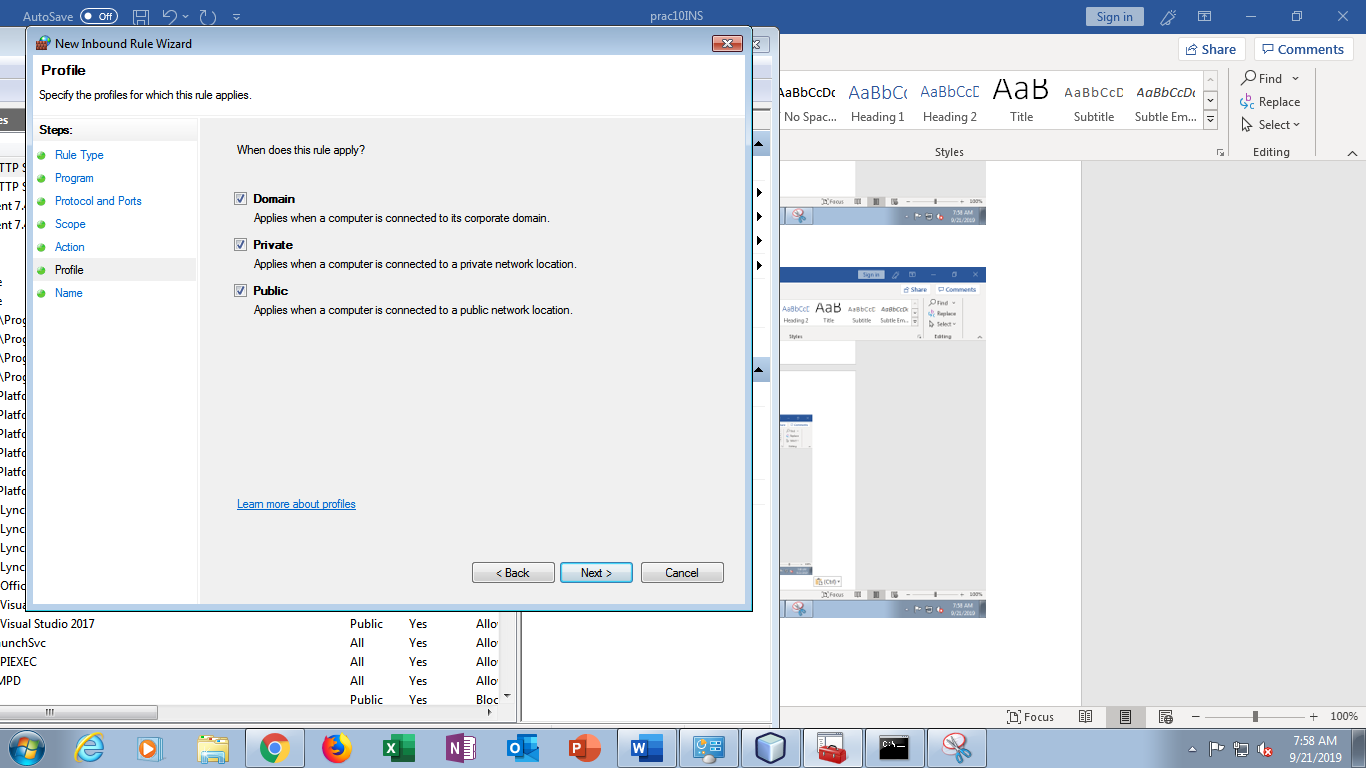
Step 8:



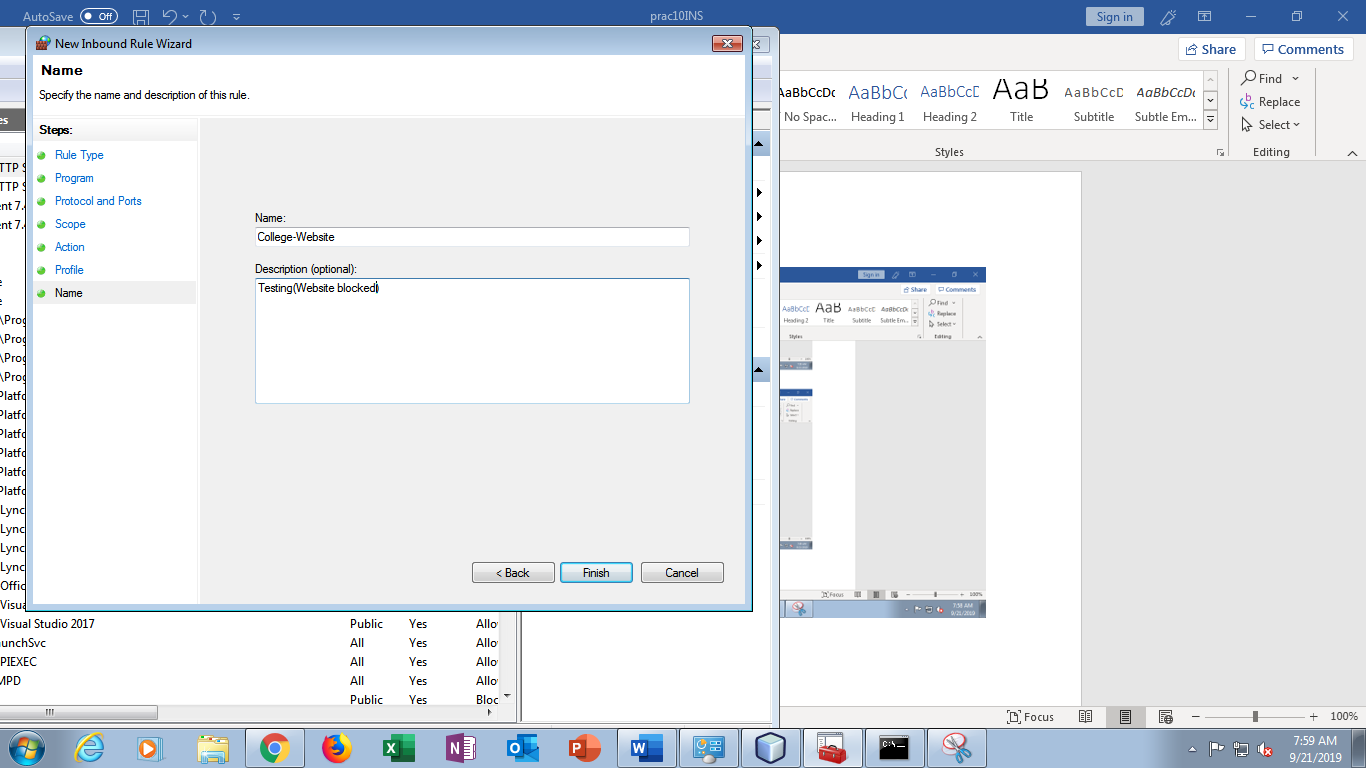
Step 9:



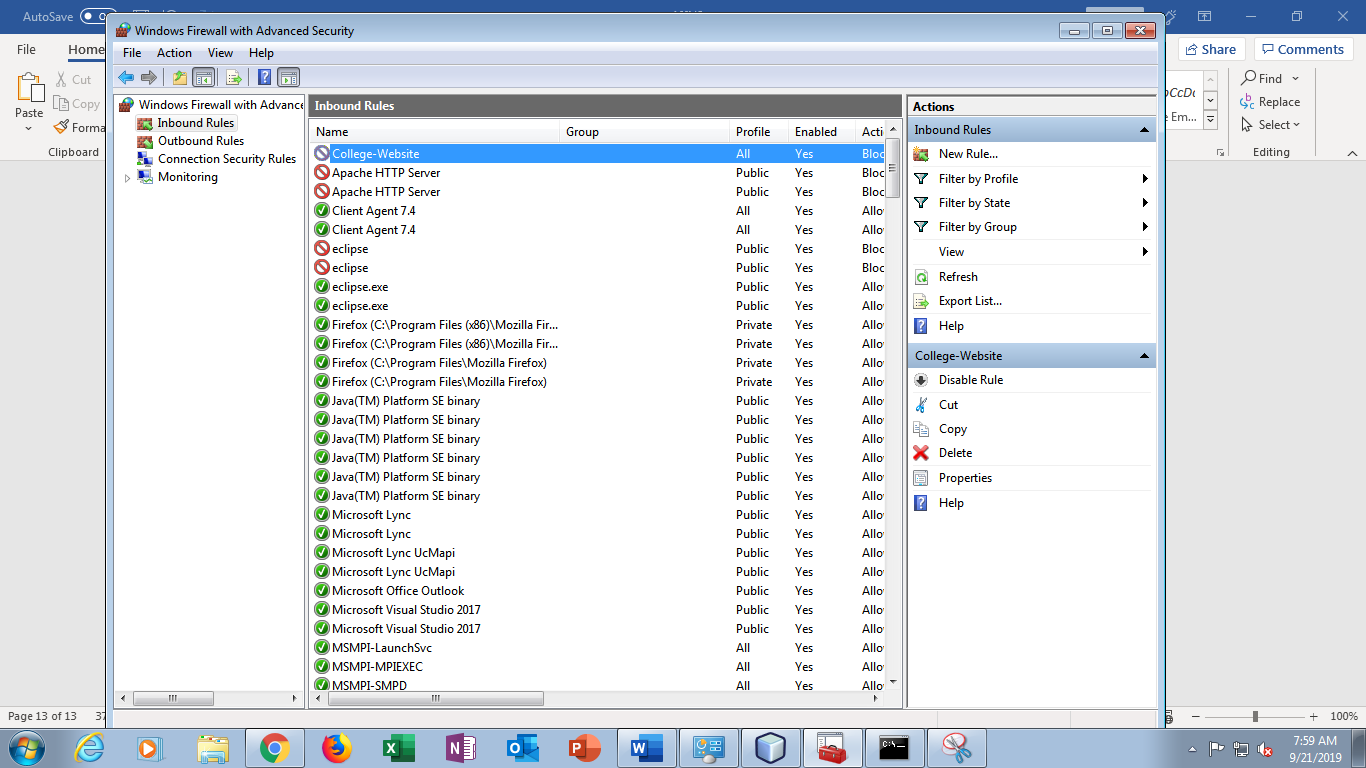
Step 10:



Step 11:



Step 12:



Step 13:

