ANNA UNIVERSITY CHENNAI
UNIVERSITY PRACTICAL EXAMINATION-2024
MAHA BARATHI ENGINEERING COLLEGE
CHINNASALEM-606201

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BACHELOR OF ENGINEERING
R-2021
CCS354 - NETWORK SECURITY LABORATORY

MAHA BARATHI ENGINEERING COLLEGE

CHINNASALEM-606 201

Bonafide Certificate

Certified that this is the bonafide record of work done by Selvan/selvi……………………………Reg No:…………………………..

Year:……………Semester:……………Branch of:………………………...

Degree Examination in the Subject:…………………………..……………

Signature Staff-In Charge

Signature of Head of the Department

Submitted for the University Practical Examination held on……………….

at **MAHA BARATHI ENGINEERING COLLEGE,CHINNASALEM – 606 201**

Signature of Internal Examiner

Date:……………………………………

Signature of External Examiner

Date:……………………………………

CONTENTS

Ex:No:01	IMPLEMENTING SYMMETRIC KEY ALGORITHM

AIM

To implement symmetric encryption cryptography using the Java programming language.

PROCEDURE:

1. **Class SecureRandom:** This class helps generate a secure random number.
2. **Class KeyGenerator:** This class provides the functionality for key generator. The following are the standard KeyGenerator algorithms with the key sizes.
3. **Approach to generate symmetric key:** The following steps can be followed in order to generate a symmetric key.
   - Create a secrete key using *SecureRandom class* in java which is used to generate a random number. This will be used to Encrypt and Decrypt the data.
   - The KeyGenerator class will provide a *getInstance()* method which can be used to pass a string variable which denotes the Key Generation Algorithm. It returns a KeyGenerator Object.
4. **Encryption and Decryption using the symmetric key:** The following steps can be followed in order to perform the encryption and decryption.
   - Create the Initialization vector that is required to avoid repetition during the encryption process. This is basically a random number. The cipher class provides two functionalities the Encryption and Decryption.
   - Finally *doFinal()* method is invoked on cipher which Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation and returns a byte array.

PROGRAM

```
// Java program to implement the// encryption and decryption

import java.security.SecureRandom;

import java.util.Scanner;

import javax.crypto.Cipher;
```

```
import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec .IvParameterSpec;

import javax.xml.bind.DatatypeConverter;

// Creating the symmetric

// class which implements

// the symmetric

public class symmetric {

	private static final String AES   = "AES";

	// We are using a Block cipher(CBC mode)

	private static final String AES_CIPHER_ALGORITHM  =
"AES/CBC/PKCS5PADDING";

	private static Scanner message;

	// Function to create a

	// secret key

	public static SecretKey createAESKey()

		throws Exception

	{

		SecureRandom securerandom    = new SecureRandom();

		KeyGenerator keygenerator = KeyGenerator.getInstance(AES);

		keygenerator.init(256, securerandom);

		SecretKey key = keygenerator.generateKey();

		return key;

	}
```

```
// Function to initialize a vector
// with an arbitrary value
public static byte[] createInitializationVector()
{
    // Used with encryption
    byte[] initializationVector = new byte[16];
    SecureRandom secureRandom  = new SecureRandom();
secureRandom.nextBytes(initializationVector);
    return initializationVector;
}
// This function takes plaintext,
// the key with an initialization
// vector to convert plainText
// into CipherText.
public static byte[] do_AESEncryption( String plainText,
SecretKey secretKey, byte[] initializationVector)   throws Exception
{
    Cipher cipher =
Cipher.getInstance(AES_CIPHER_ALGORITHM);
    IvParameterSpec ivParameterSpec = new IvParameterSpec(
initializationVector);
    cipher.init(Cipher.ENCRYPT_MODE, secretKey,
ivParameterSpec);
    return cipher.doFinal(plainText.getBytes());
}
// This function performs the
```

```
// reverse operation of the
// do_AESEncryption function.
// It converts ciphertext to
// the plaintext using the key.
public static String do_AESDecryption(byte[] cipherText, SecretKey secretKey, byte[] initializationVector)
        throws Exception
{
        Cipher cipher = Cipher.getInstance(AES_CIPHER_ALGORITHM);
        IvParameterSpec ivParameterSpec = new IvParameterSpec( initializationVector);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivParameterSpec);
        byte[] result = cipher.doFinal(cipherText);
        return new String(result);
}

// Driver code
public static void main(String args[])
        throws Exception
{
        SecretKey Symmetrickey = createAESKey();
        System.out.println("The Symmetric Key is :"+ DatatypeConverter.printHexBinary( Symmetrickey.getEncoded()));
        byte[] initializationVector = createInitializationVector();
```

```
String plaintext = "This is the message "+ "I want To Encrypt.";

// Encrypting the message

// using the symmetric key

byte[] cipherText = do_AESEncryption(plainText,
Symmetrickey, initializationVector);

System.out.println("The ciphertext or "+ "Encrypted Message is: "
    + DatatypeConverter.printHexBinary( cipherText));

// Decrypting the encrypted

// message

String decryptedText = do_AESDecryption( cipherText,
Symmetrickey, initializationVector);


System.out.println( "Your original message is: " +
decryptedText);

    }

}
```

OUTPUT:

```
Output
The Symmetric Key is :AD243EE2408A41726D0D977692664A5A5B70117B9416BFF705C706A10F0A8AF0
Please Enter your Message :
This is the message I want To Encrypt.
The ciphertext or Encrypted Message is : 925AAB888938921CE8DF51BC022DC4DCB25C103C2652F5420EA290C8A694E2597A6BC747D00B80
Your original message is : This is the message I want To Encrypt.
```

RESULT:

Thus, the program implements a symmetric key algorithm using java and successfully verified the output.

Ex:No:02(a)	IMPLEMENTING ASYMMETRIC KEY ALGORITHM

AIM

To implement asymmetric key algorithm using the Java programming language.

PROCEDURE:

1. To generate a keypair(public, private). The following steps can be followed in order to generate asymmetric key:
   - We need to first generate public & private key using the *SecureRandom class*. SecureRandom class is used to generate random number.
   - The KeyGenerator class will provide *getInstance()* method which can be used to pass a string variable which denotes the Key Generation Algorithm. It returns KeyGenerator Object. We are using RSA algorithm for generating the keys.
   - Initializing the keyGenerator object with 2048 bits key size and passing the random number.
   - Now, the secret key is generated and if we wish to actually see the generated key which is an object, we can convert it into hexbinary format using DatatypeConverter.
2. **Encryption and Decryption using the asymmetric key:** In the above steps, we have created the public & private keys for Encryption and Decryption. Now, let us implement Asymmetric Encryption using the RSA algorithm. The following steps can be followed in order to implement the encryption and decryption.
   - The cipher class is used for two different modes the encryption and decryption. As Asymmetric encryption uses different keys, we use the private key for encryption and the public key for decryption.
   - The *doFinal()* method is invoked on cipher which encrypts/decrypts data in a single-part operation, or finishes a multiple-part operation and returns byte array.

- Finally we get the Cipher text after Encryption with ENCRYPT_MODE.

PROGRAM

```
// Java program to perform the
// encryption and decryption
// using asymmetric key


package java_cryptography;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.util.Scanner;


import javax.crypto.Cipher;
import javax.xml.bind
	.DatatypeConverter;


public class Asymmetric {


	private static final String RSA
		= "RSA";
	private static Scanner sc;
```

```
// Generating public & private keys
// using RSA algorithm.
public static KeyPair generateRSAKkeyPair()
        throws Exception
{
        SecureRandom secureRandom
                = new SecureRandom();
        KeyPairGenerator keyPairGenerator
                = KeyPairGenerator.getInstance(RSA);

        keyPairGenerator.initialize(
                2048, secureRandom);
        return keyPairGenerator
                .generateKeyPair();
}

// Encryption function which converts
// the plainText into a cipherText
// using private Key.
public static byte[] do_RSAEncryption(
        String plainText,
        PrivateKey privateKey)
        throws Exception
{
```

```
        Cipher cipher
            = Cipher.getInstance(RSA);

        cipher.init(
            Cipher.ENCRYPT_MODE, privateKey);

        return cipher.doFinal(
            plainText.getBytes());
    }

    // Decryption function which converts
    // the ciphertext back to the
    // original plaintext.
    public static String do_RSADecryption(
        byte[] cipherText,
        PublicKey publicKey)
        throws Exception
    {
        Cipher cipher
            = Cipher.getInstance(RSA);

        cipher.init(Cipher.DECRYPT_MODE,
                    publicKey);
        byte[] result
```

```
                = cipher.doFinal(cipherText);

        return new String(result);
    }

    // Driver code
    public static void main(String args[])
        throws Exception
    {
        KeyPair keypair
            = generateRSAKkeyPair();

        String plainText = "This is the PlainText "
                           + "I want to Encrypt using RSA.";

        byte[] cipherText
            = do_RSAEncryption(
                plainText,
                keypair.getPrivate());

        System.out.println(
            "The Public Key is: "
            + DatatypeConverter.printHexBinary(
                keypair.getPublic().getEncoded()));
```

```
        System.out.println(
            "The Private Key is: "
            + DatatypeConverter.printHexBinary(
                keypair.getPrivate().getEncoded()));

        System.out.print("The Encrypted Text is: ");

        System.out.println(
            DatatypeConverter.printHexBinary(
                cipherText));

        String decryptedText
            = do_RSADecryption(
                cipherText,
                keypair.getPublic());

        System.out.println(
            "The decrypted text is: "
            + decryptedText);
    }
}
```

OUTPUT:

```
Javadoc  Declaration  Console
<terminated> Asymmetric [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (Jun 11, 2020, 1:14:25 PM – 1:15:00 PM)
Enter the plainText to Encrypt and Decrypt
This is the PlainText I want to Encrypt using RSA.
public_key is :30820122300D06092A864886F70D01010105000382010F003082010A0282010100B48C76513803517429532347B
private_key is :308204BC020100300D06092A864886F70D0101010500048204A6308204A20201000282010100B48C7651380351
The Encrypted Text is (ciphertext) : 9D873912E423E8EEB42F261EE7F0A29A36BDCA220D11A5E45021534EC85DFB0B499D0
The decrypted text is : This is the PlainText I want to Encrypt using RSA.
```

RESULT:

Thus, the program implements an asymmetric encryption using java and successfully verified the output.

Ex:No:2(b)	IMPLEMENTING KEY EXCHANGE ALGORITHM (DIFFIE-HELLMAN ALGORITHM)

AIM

To Implementation of Key Exchange Algorithm (Diffie-Hellman Algorithm) using java program

PROCEDURE:

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: x =(9^4 mod 23) = (6561 mod 23) = 6

Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and

Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys

Alice: ka = y^a mod p = 65536 mod 23 = 9

Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

PROGRAM

```
// This program calculates the Key for two persons
// using the Diffie-Hellman Key exchange algorithm
class GFG {

    // Power function to return value of a ^ b mod P
    private static long power(long a, long b, long p)
    {
        if (b == 1)
            return a;
        else
            return (((long)Math.pow(a, b)) % p);
    }

    // Driver code
    public static void main(String[] args)
```

```
{
        long P, G, x, a, y, b, ka, kb;

        // Both the persons will be agreed upon the
        // public keys G and P

        // A prime number P is taken
        P = 23;
        System.out.println("The value of P:" + P);

        // A primitive root for P, G is taken
        G = 9;
        System.out.println("The value of G:" + G);

        // Alice will choose the private key a
        // a is the chosen private key
        a = 4;
        System.out.println("The private key a for Alice:"
                                         + a);

        // Gets the generated key
        x = power(G, a, P);

        // Bob will choose the private key b
```

```
        // b is the chosen private key
        b = 3;
        System.out.println("The private key b for Bob:"
                           + b);

        // Gets the generated key
        y = power(G, b, P);

        // Generating the secret key after the exchange
        // of keys
        ka = power(y, a, P); // Secret key for Alice
        kb = power(x, b, P); // Secret key for Bob

        System.out.println("Secret key for the Alice is:"
                           + ka);
        System.out.println("Secret key for the Bob is:"
                           + kb);
    }
}
```

OUTPUT

```
The value of P : 23
The value of G : 9

The private key a for Alice : 4
The private key b for Bob : 3

Secret key for the Alice is : 9
Secret Key for the Bob is : 9
```

RESULT:

Thus, the program implements a Key Exchange Algorithm (DH algorithm) using java and successfully verified the output.

Ex:No:03	IMPLEMENTING DIGITAL SIGNATURES

AIM

To Implementation of Digital Signatures using java program.

PROCEDURE:

Let us implement the digital signature using algorithms SHA and RSA and also verify if the hash matches with a public key.

1. Create a method named Create_Digital_Signature() to implement Digital Signature by passing two parameters input message and the private key. In this method we will get an instance of the signature object passing the signing algorithm and assign it with a private key and finally pass the input this will return byte array.
2. The next step is to generate asymmetric key pair using RSA algorithm and SecureRandom class functions.
3. Finally verifying the signature using public key. Verify_Digital_Signature() method is used to check whether the signature matches by passing it the input, signature, and public key.

PROGRAM:

```
// Java implementation for Generating

// and verifying the digital signature


package java_cryptography;


// Imports

import java.security.KeyPair;
```

```
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.util.Scanner;

import javax.xml.bind.DatatypeConverter;

public class Digital_Signature_GeeksforGeeks {

    // Signing Algorithm
    private static final String
        SIGNING_ALGORITHM
        = "SHA256withRSA";
    private static final String RSA = "RSA";
    private static Scanner sc;

    // Function to implement Digital signature
    // using SHA256 and RSA algorithm
    // by passing private key.
    public static byte[] Create_Digital_Signature(
        byte[] input,
        PrivateKey Key)
```

```
        throws Exception
{
        Signature signature
                = Signature.getInstance(
                        SIGNING_ALGORITHM);
        signature.initSign(Key);
        signature.update(input);
        return signature.sign();
}

// Generating the asymmetric key pair
// using SecureRandom class
// functions and RSA algorithm.
public static KeyPair Generate_RSA_KeyPair()
        throws Exception
{
        SecureRandom secureRandom
                = new SecureRandom();
        KeyPairGenerator keyPairGenerator
                = KeyPairGenerator
                        .getInstance(RSA);
        keyPairGenerator
                .initialize(
                        2048, secureRandom);
```

```
        return keyPairGenerator
            .generateKeyPair();
    }

    // Function for Verification of the
    // digital signature by using the public key
    public static boolean
    Verify_Digital_Signature(
        byte[] input,
        byte[] signatureToVerify,
        PublicKey key)
        throws Exception
    {
        Signature signature
            = Signature.getInstance(
                SIGNING_ALGORITHM);
        signature.initVerify(key);
        signature.update(input);
        return signature
            .verify(signatureToVerify);
    }

    // Driver Code
    public static void main(String args[])
```

```
        throws Exception
    {

        String input
            = "GEEKSFORGEEKS IS A"
            + " COMPUTER SCIENCE PORTAL";
        KeyPair keyPair
            = Generate_RSA_KeyPair();

        // Function Call
        byte[] signature
            = Create_Digital_Signature(
                input.getBytes(),
                keyPair.getPrivate());

        System.out.println(
            "Signature Value:\n "
            + DatatypeConverter
                .printHexBinary(signature));

        System.out.println("Verification: "+ Verify_Digital_Signature( input.getBytes(), signature, keyPair.getPublic()));
    }
}
```

OUTPUT:

```
Signature Value:
2492035AE7782EEB75E18C1C76651384FDE30178DBE806A67DA4C884D52BF15A35CB8D1F
Verification: true
```

RESULT:

Thus, the program implements a Digital Signature Scheme using java and successfully verified the output.

Ex:No:04	INSTALLATION OF WIRESHARK, TCPDUMP AND OBSERVE THE DATA TRANSFERRED IN CLIENT SERVER COMMUNICATION USING TCP/UDP AND IDENTIFY THE TCP/UDP DATAGRAM.

AIM

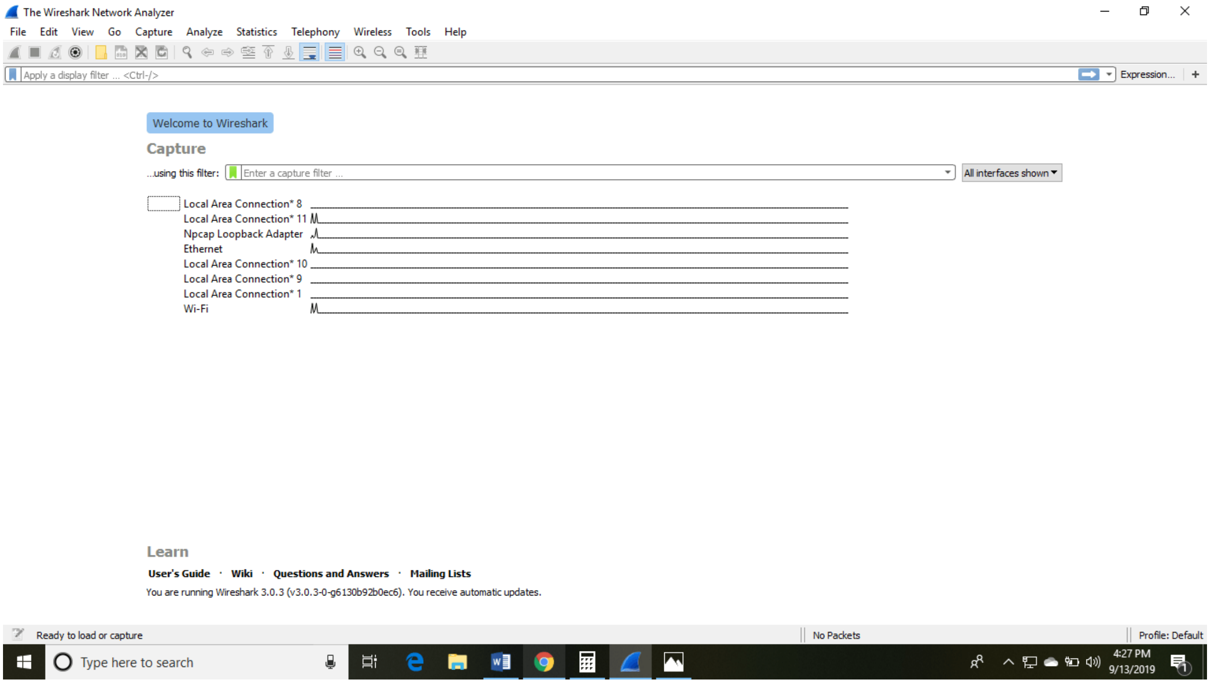
To installation of wire shark, tcpdump observe the data transfer in client server communication using TCP/UDP and identify the TCP/UDP datagram.

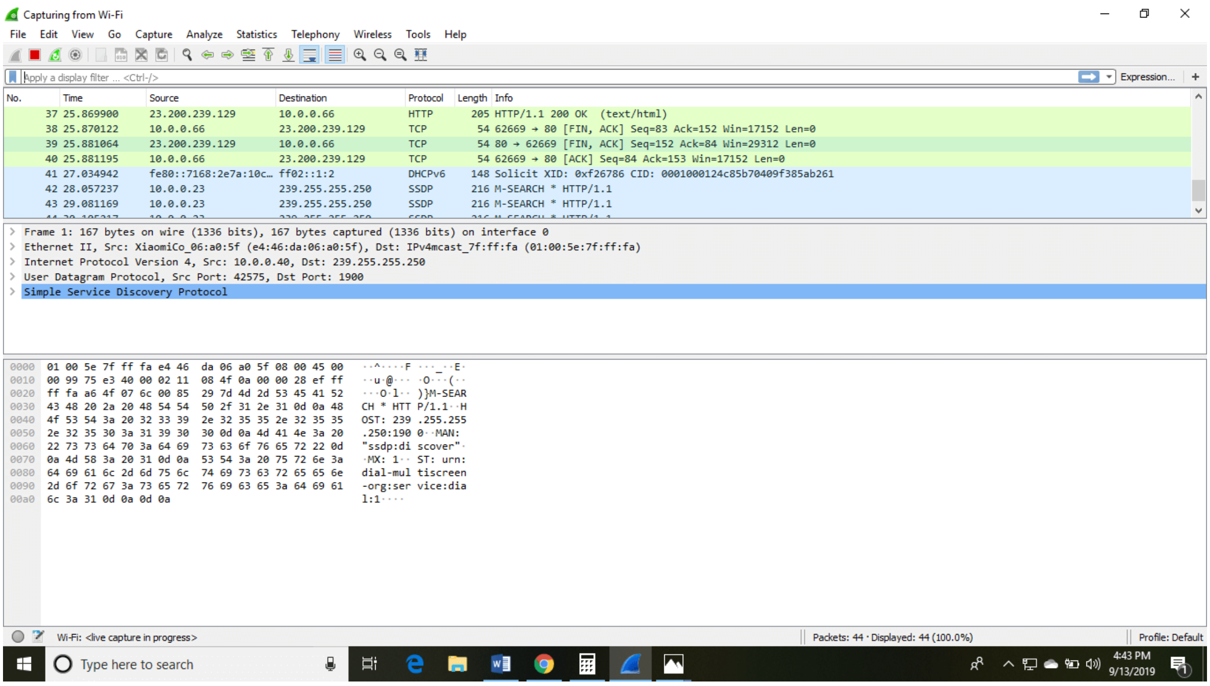
PROCEDURE

Installation of Wire shark Software

Below are the steps to install the Wire shark software on the computer:

1. Open the web browser.
2. Search for 'Download Wire shark.'
3. Select the Windows installer according to your system configuration, either 32-bt or 64-bit. Save the program and close the browser.
4. Now, open the software, and follow the install instruction by accepting the license.
5. The Wire shark is ready for use.





The screen/interface of the Wire shark is divided into five parts:

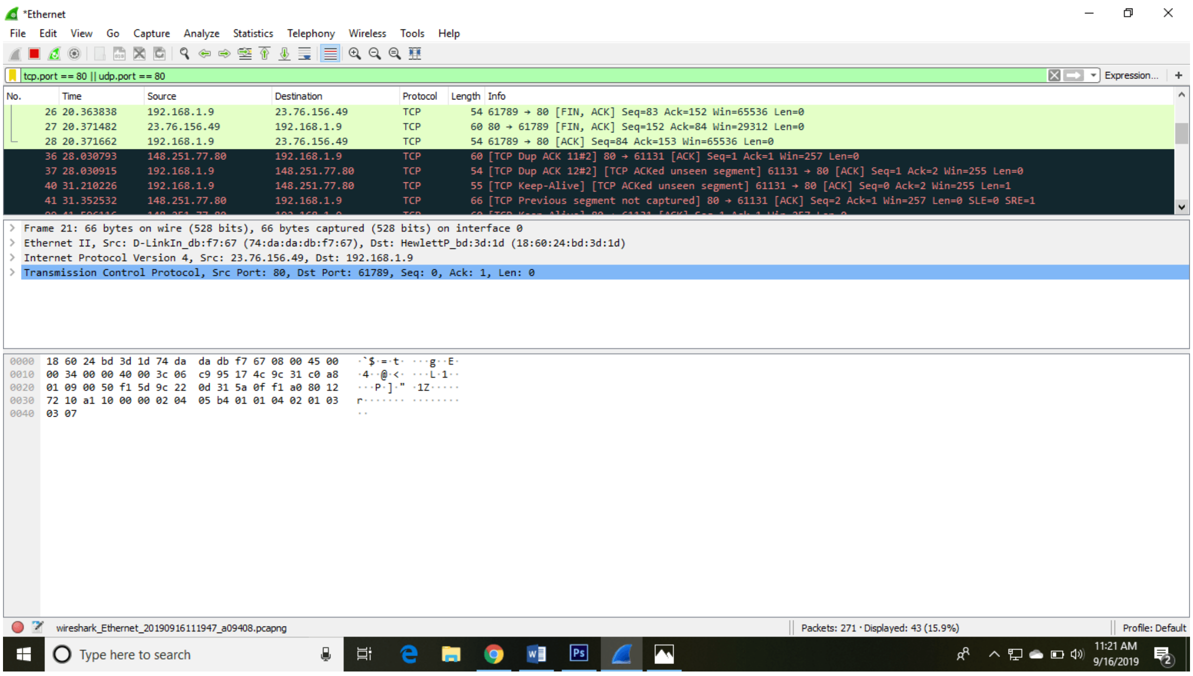
- First part contains a menu bar and the options displayed below it. This part is at the top of the window. File and the capture menus options are commonly used in Wire shark. The capture menu allows to start the capturing process. And the File menu is used to open and save a capture file.

- The second part is the packet listing window. It determines the packet flow or the captured packets in the traffic. It includes the packet number, time, source, destination, protocol, length, and info. We can sort the packet list by clicking on the column name.

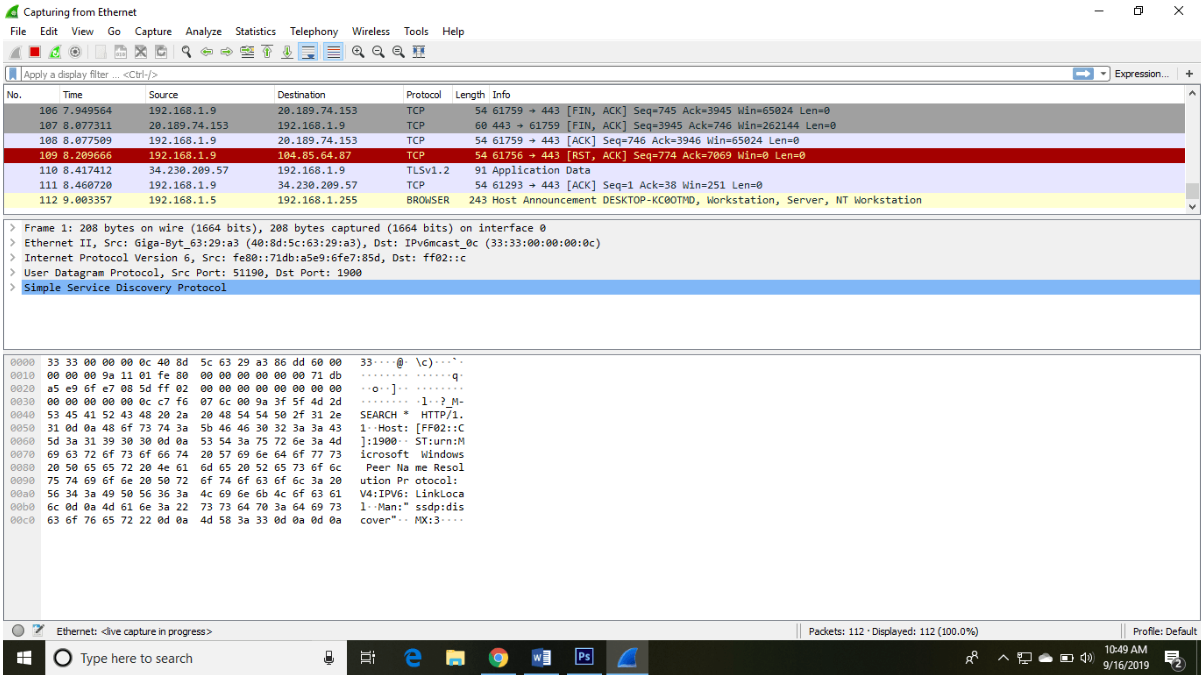
- Next comes the packet header- detailed window. It contains detailed information about the components of the packets. The protocol info can also be expanded or minimized according to the information required.

- The bottom window called the packet contents window, which displays the content in ASCII and hexadecimal format.

- At last, is the filter field which is at the top of the display. The captured packets on the screen can be filtered based on any component according to your requirements. For example, if we want to see only the packets with the HTTP protocol, we can apply filters to that option. All the packets with HTTP as the protocol will only be displayed on the screen, shown below:



After connecting, you can watch the traffic below:



Basic concepts of the Network Traffic

IP Addresses: It was designed for the devices to communicate with each other on a local network or over the Internet. It is used for host or network interface identification. It provides the location of the host and capacity of establishing the path to the host in that network. Internet Protocol is the set of predefined rules or terms under which the communication should be conducted. The types of IP addresses are IPv4 and IPv6.

- IPv4 is a 32-bit address in which each group represents 8 bits ranging from 0 to 255.
- IPv6 is a 128-bit address.

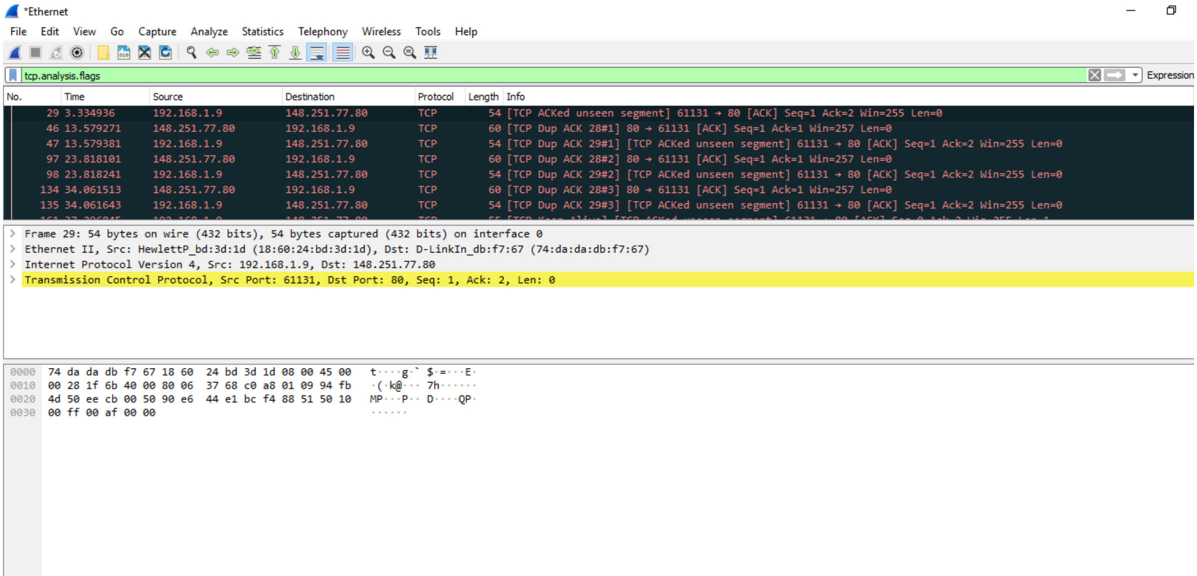

*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.analysis.flags
Expression
No. Time Source Destination Protocol Length Info
29 3.334936 192.168.1.9 148.251.77.80 TCP 54 [TCP ACKed unseen segment] 61131 → 80 [ACK] Seq=1 Ack=2 Win=255 Len=0
46 13.579271 148.251.77.80 192.168.1.9 TCP 60 [TCP Dup ACK 28#1] 80 → 61131 [ACK] Seq=1 Ack=1 Win=257 Len=0
47 13.579381 192.168.1.9 148.251.77.80 TCP 54 [TCP Dup ACK 29#1] [TCP ACKed unseen segment] 61131 → 80 [ACK] Seq=1 Ack=2 Win=255 Len=0
97 23.818101 148.251.77.80 192.168.1.9 TCP 60 [TCP Dup ACK 28#2] 80 → 61131 [ACK] Seq=1 Ack=1 Win=257 Len=0
98 23.818241 192.168.1.9 148.251.77.80 TCP 54 [TCP Dup ACK 29#2] [TCP ACKed unseen segment] 61131 → 80 [ACK] Seq=1 Ack=2 Win=255 Len=0
134 34.061513 148.251.77.80 192.168.1.9 TCP 60 [TCP Dup ACK 28#3] 80 → 61131 [ACK] Seq=1 Ack=1 Win=257 Len=0
135 34.061643 192.168.1.9 148.251.77.80 TCP 54 [TCP Dup ACK 29#3] [TCP ACKed unseen segment] 61131 → 80 [ACK] Seq=1 Ack=2 Win=255 Len=0
Frame 29: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
Ethernet II, Src: HewlettP_bd:3d:1d (18:60:24:bd:3d:1d), Dst: D-LinkIn_db:f7:67 (74:da:da:db:f7:67)
Internet Protocol Version 4, Src: 192.168.1.9, Dst: 148.251.77.80
Transmission Control Protocol, Src Port: 61131, Dst Port: 80, Seq: 1, Ack: 2, Len: 0
0000 74 da da db f7 67 18 60 24 bd 3d 1d 08 00 45 00
0010 00 28 1f 6b 40 00 80 06 37 68 c0 a8 01 09 94 fb
0020 4d 50 ee cb 00 50 90 e6 44 e1 bc f4 88 51 50 10
0030 00 ff 00 af 00 00

Fig (5)

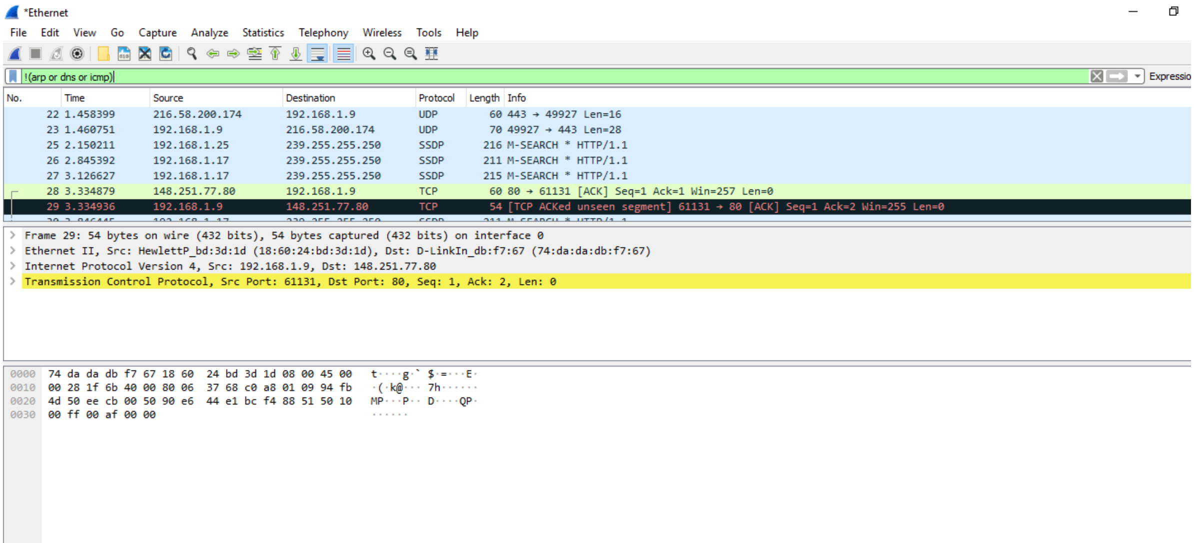

*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
!(arp or dns or icmp)
Expressio
No. Time Source Destination Protocol Length Info
22 1.458399 216.58.200.174 192.168.1.9 UDP 60 443 → 49927 Len=16
23 1.460751 192.168.1.9 216.58.200.174 UDP 70 49927 → 443 Len=28
25 2.150211 192.168.1.25 239.255.255.250 SSDP 216 M-SEARCH * HTTP/1.1
26 2.845392 192.168.1.17 239.255.255.250 SSDP 211 M-SEARCH * HTTP/1.1
27 3.126627 192.168.1.17 239.255.255.250 SSDP 215 M-SEARCH * HTTP/1.1
28 3.334879 148.251.77.80 192.168.1.9 TCP 60 80 → 61131 [ACK] Seq=1 Ack=1 Win=257 Len=0
29 3.334936 192.168.1.9 148.251.77.80 TCP 54 [TCP ACKed unseen segment] 61131 → 80 [ACK] Seq=1 Ack=2 Win=255 Len=0
Frame 29: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
Ethernet II, Src: HewlettP_bd:3d:1d (18:60:24:bd:3d:1d), Dst: D-LinkIn_db:f7:67 (74:da:da:db:f7:67)
Internet Protocol Version 4, Src: 192.168.1.9, Dst: 148.251.77.80
Transmission Control Protocol, Src Port: 61131, Dst Port: 80, Seq: 1, Ack: 2, Len: 0
0000 74 da da db f7 67 18 60 24 bd 3d 1d 08 00 45 00
0010 00 28 1f 6b 40 00 80 06 37 68 c0 a8 01 09 94 fb
0020 4d 50 ee cb 00 50 90 e6 44 e1 bc f4 88 51 50 10
0030 00 ff 00 af 00 00

Fig (6)

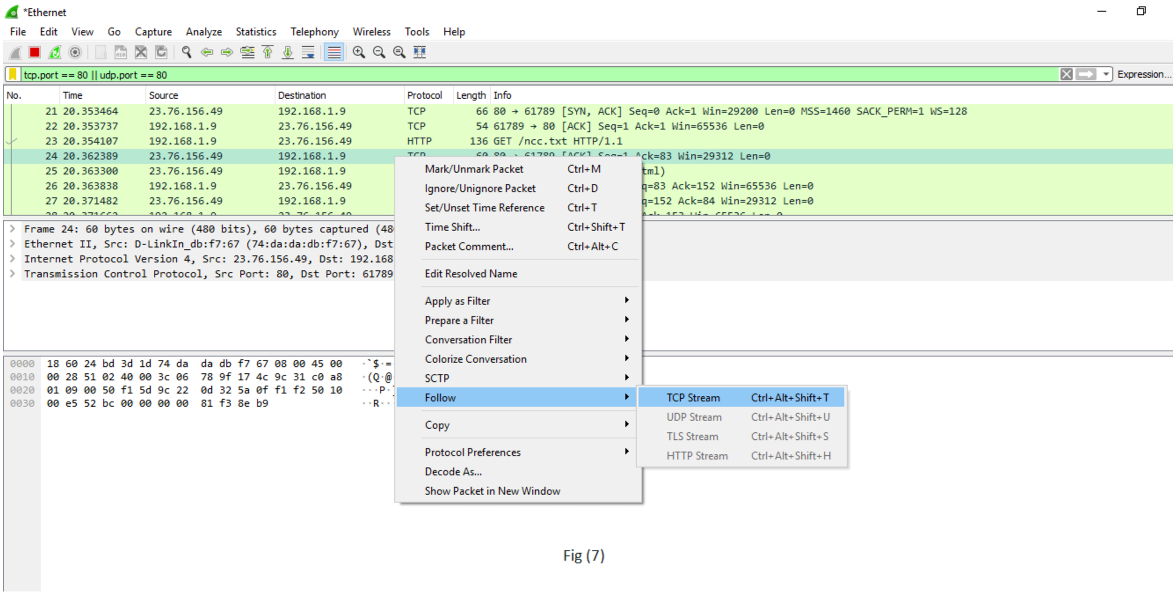


Fig (7)

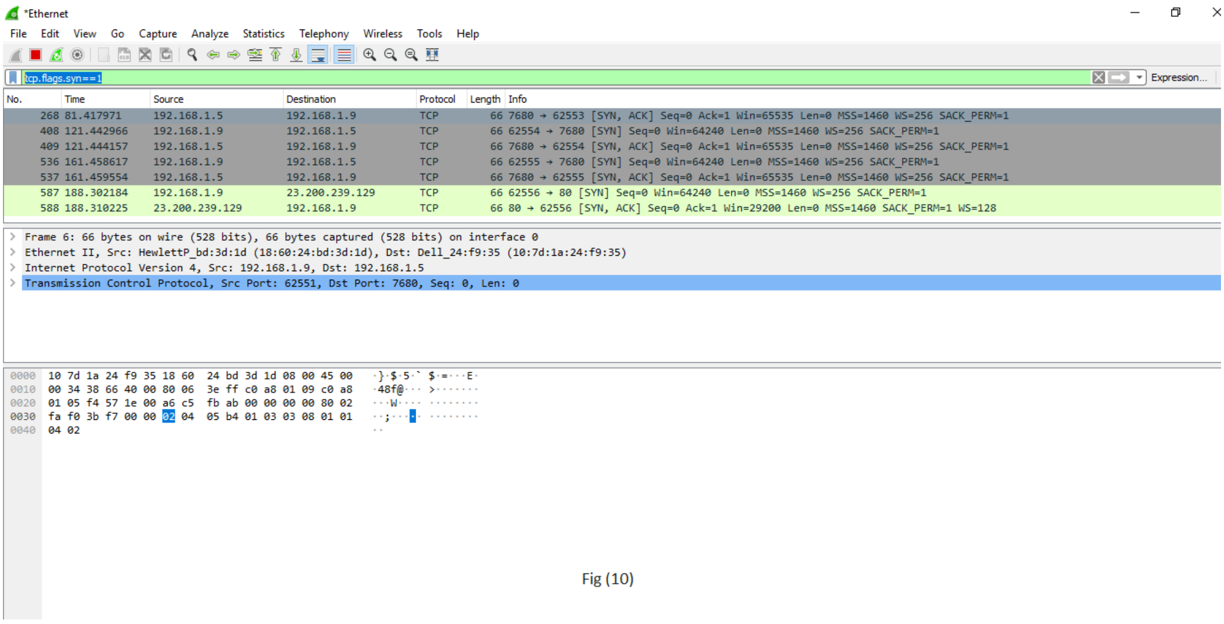
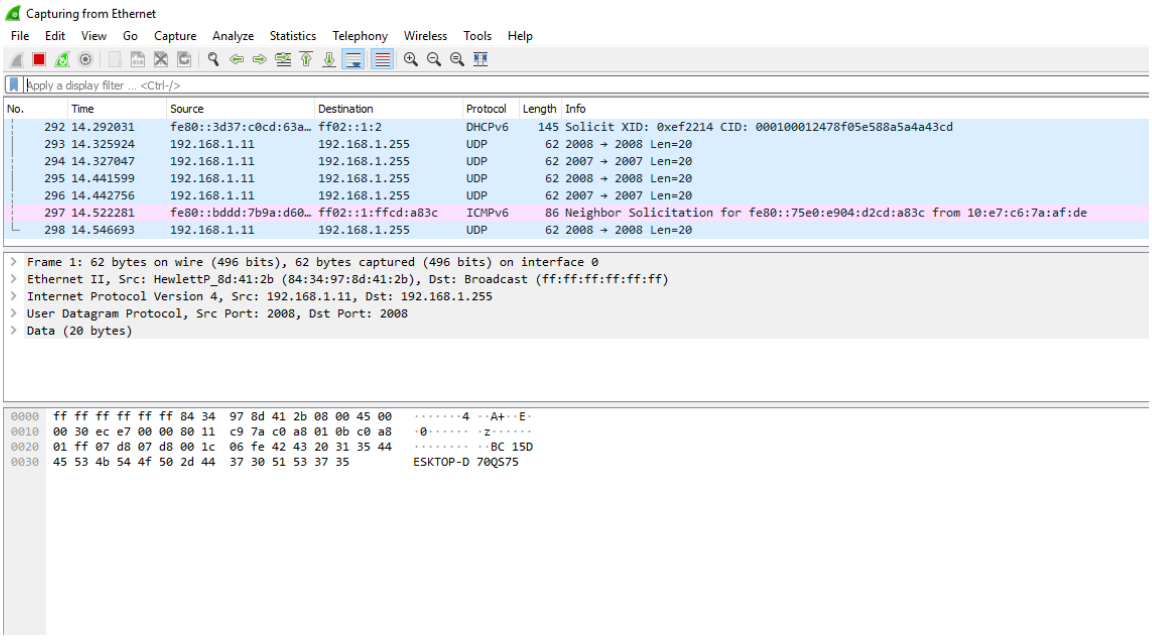


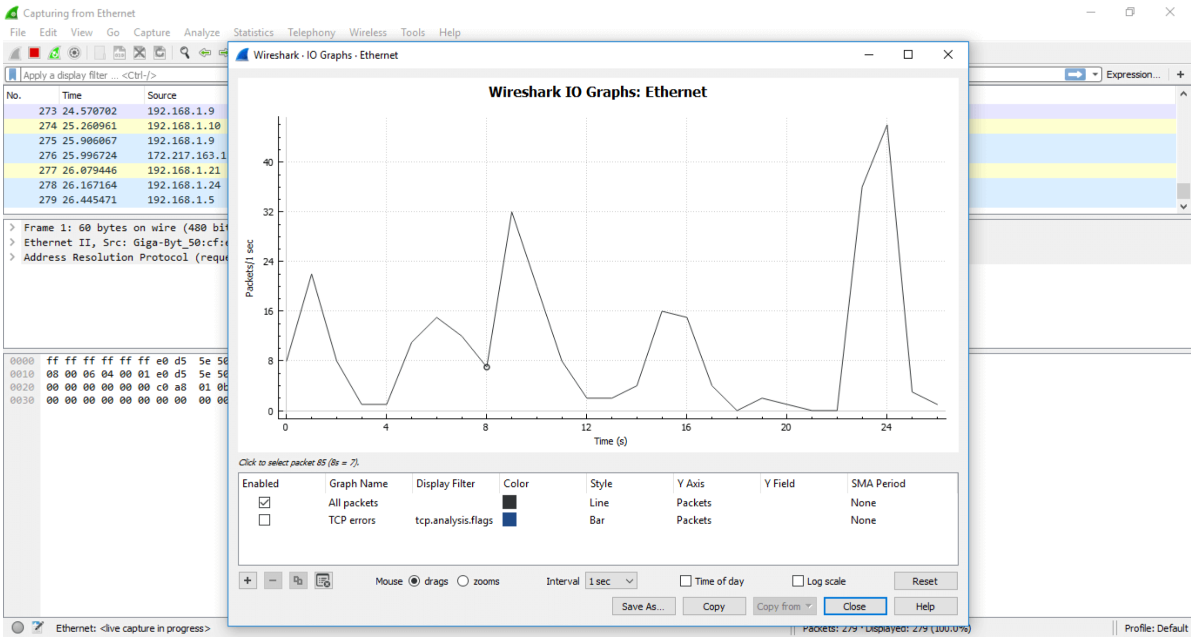
Fig (10)

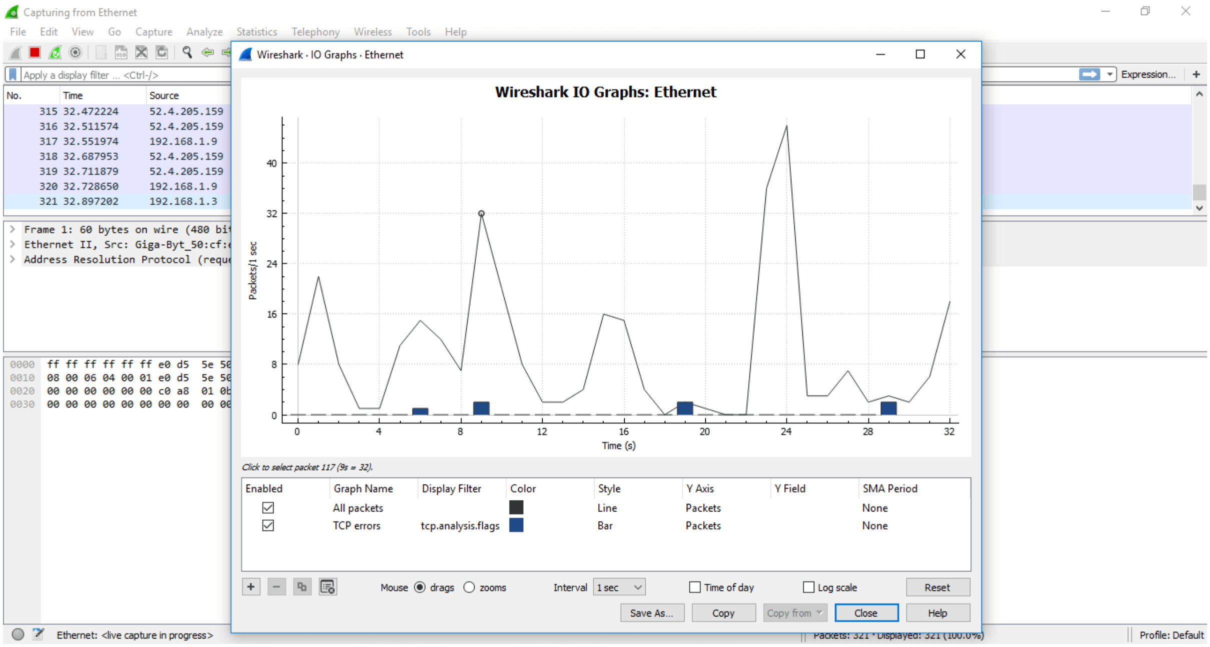
Wireshark packet sniffing

- Open the Wireshark Application.
- Select the current interface. Here in this example, interface is Ethernet that we would be using.
- The network traffic will be shown below, which will be continuous. To stop or watch any particular packet, you can press the red button below the menu bar.


Capturing from Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
292 14.292031 fe80::3d37:c0cd:63a… ff02::1:2 DHCPv6 145 Solicit XID: 0xef2214 CID: 000100012478f05e588a5a4a43cd
293 14.325924 192.168.1.11 192.168.1.255 UDP 62 2008 → 2008 Len=20
294 14.327047 192.168.1.11 192.168.1.255 UDP 62 2007 → 2007 Len=20
295 14.441599 192.168.1.11 192.168.1.255 UDP 62 2008 → 2008 Len=20
296 14.442756 192.168.1.11 192.168.1.255 UDP 62 2007 → 2007 Len=20
297 14.522281 fe80::bddd:7b9a:d60… ff02::1:ffcd:a83c ICMPv6 86 Neighbor Solicitation for fe80::75e0:e904:d2cd:a83c from 10:e7:c6:7a:af:de
298 14.546693 192.168.1.11 192.168.1.255 UDP 62 2008 → 2008 Len=20
Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
Ethernet II, Src: HewlettP_8d:41:2b (84:34:97:8d:41:2b), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 192.168.1.11, Dst: 192.168.1.255
User Datagram Protocol, Src Port: 2008, Dst Port: 2008
Data (20 bytes)
0000 ff ff ff ff ff ff 84 34 97 8d 41 2b 08 00 45 00
0010 00 30 ec e7 00 00 80 11 c9 7a c0 a8 01 0b c0 a8
0020 01 ff 07 d8 07 d8 00 1c 06 fe 42 43 20 31 35 44
0030 45 53 4b 54 4f 50 2d 44 37 30 51 53 37 35
ESKTOP-D 70QS75

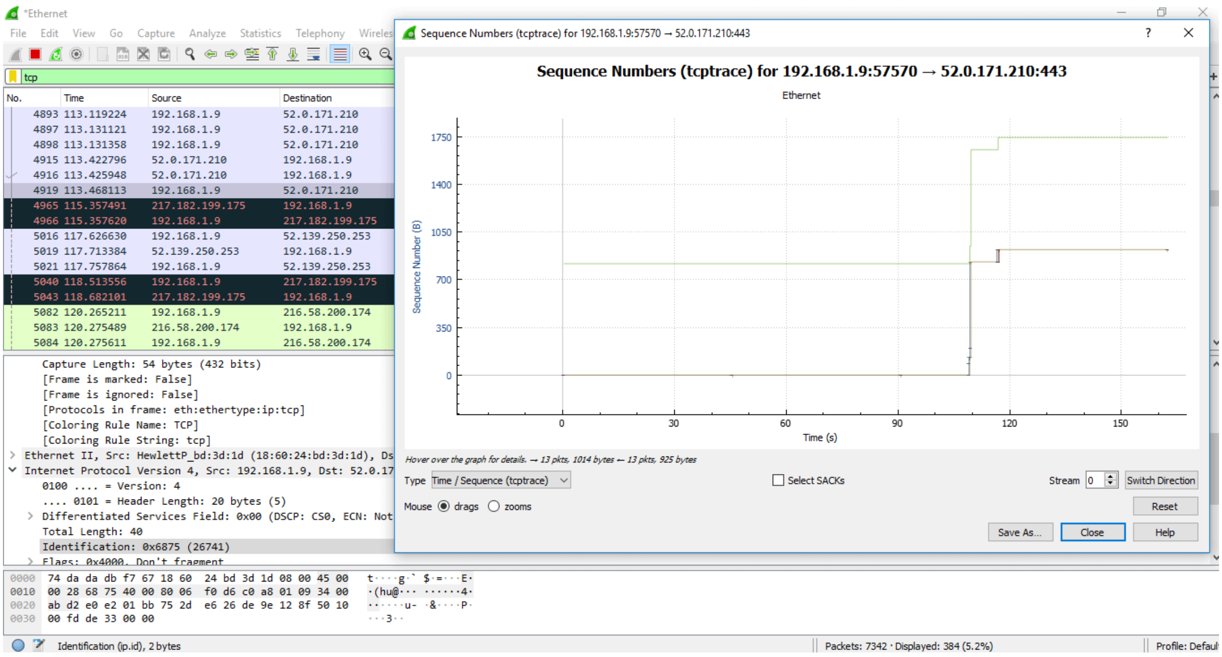
I/O GRAPHS

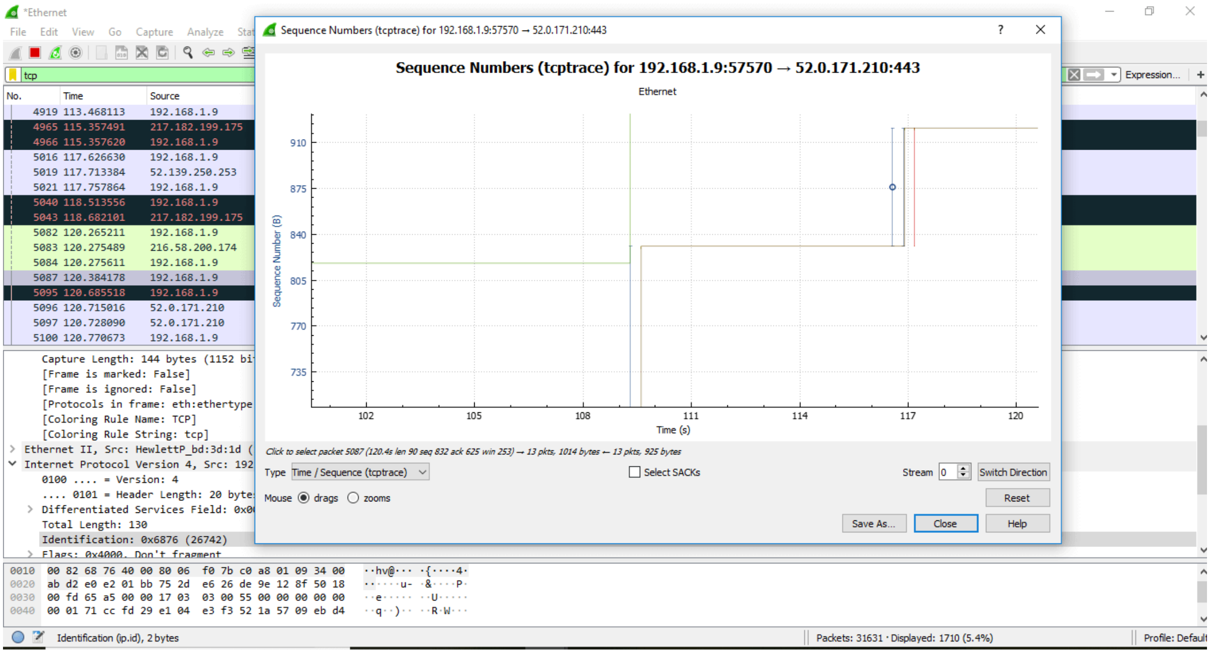

Capturing from Ethernet
Wireshark · IO Graphs · Ethernet
Wireshark IO Graphs: Ethernet
Packets/1 sec
Time (s)
Click to select packet 85 (8s = 7).
Enabled Graph Name Display Filter Color Style Y Axis Y Field SMA Period
All packets Line Packets None
TCP errors tcp.analysis.flags Bar Packets None
Mouse drags zooms Interval 1 sec Time of day Log scale Reset
Save As... Copy Copy from Close Help
Ethernet: <live capture in progress>
Packets: 279 · Displayed: 279 (100.0%)
Profile: Default



the steps to understand the TCP Stream graphs:

- Open the Wireshark. Click on the interface to watch the network traffic.
- Apply the filter as 'tcp.'
- Click on the option 'Statistics 'on the menu bar and select 'TCP Stream graphs' and select 'Time sequence (tcptrace). You can also choose other options in the 'TCP Stream graphs' category depending on your requirements. Now the screen will look as:

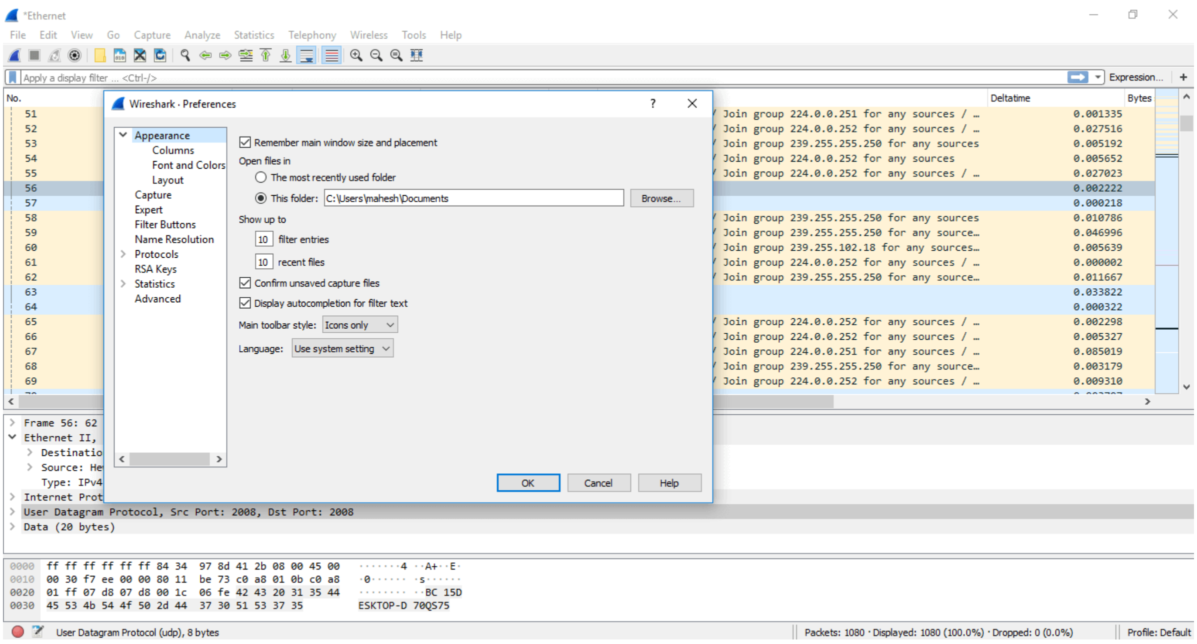




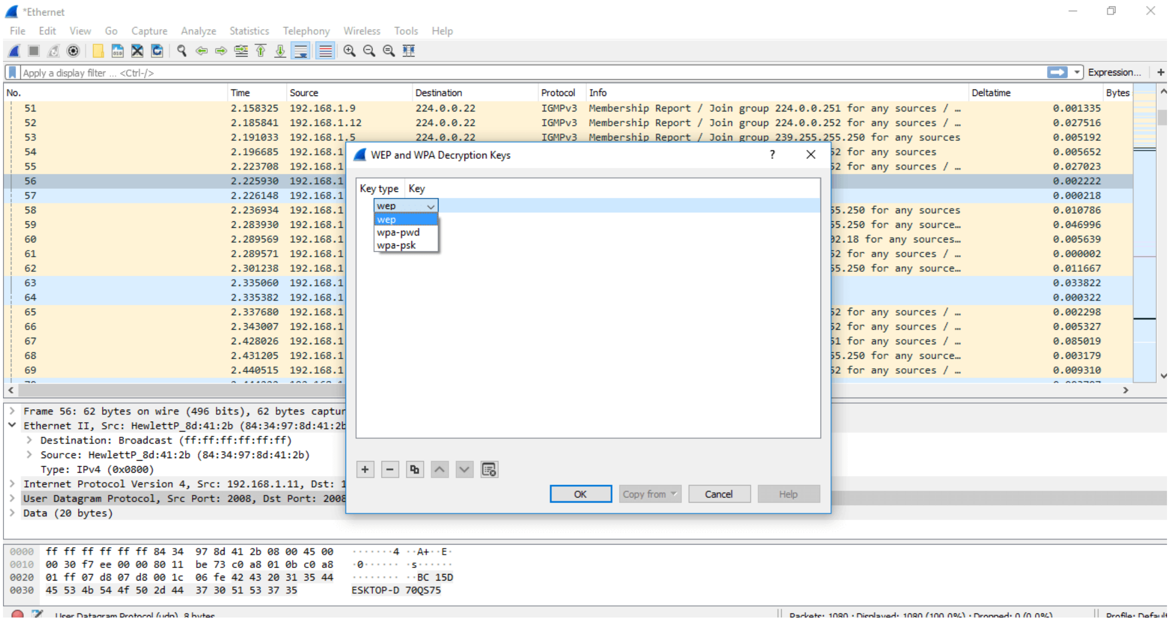
WIRESHARK DECRYPTION

The decryption process is used for the data to be in a readable format. Below are the steps for the decryption process.

- Open the Wireshark and then select the particular interface as explained above.
- Go to the 'Edit' option and select the 'Preferences' option.
- A dialogue will appear as shown below:



- Select the 'Protocol' option in the left column.
- From the drop-down list, select the 'IEEE 802.11' option. Check the box of decryption and click on the Edit option under it.
- A box will appear. Click on the option shown below:



- Select the option wpa-pwd and set the password accordingly.
- The data will be decrypted.
- But the above decryption process is only possible if there is a proper handshake.

RESULT:

Thus, the installation of wire shark, tcpdump observes the data transfer in client server communication using TCP/UDP and identify the TCP/UDP datagram successfully install and output is verified.

Ex:No:05	CHECK MESSAGE INTERGRITY AND CONFIDENTIALITY USING SSL

AIM:

To Check Message Intergrity And Confidentiality Using SSL.

PROCEDURE:

Installing & Configuring HTTP with SSL (HTTPS)
Public Key Cryptography (Asymmetric Cryptography)

In public key cryptography, a matching pair of keys is used; one for encryption and the other for decryption. One of the key is called the public key (can be published or sent over the network and known to all users). The other is called the private key (kept secretly by the owner).

$K_E \neq K_D$

In some public-key algorithms, such as RSA, both keys can be used for encryption. In other algorithms, one key is for encryption only and the other for decryption.

Handshaking - Key Exchange

Once the ciphersuit to be used are negotiated and agree-upon, the client and server will establish a session key:

1. The client uses server's public key to encrypt a secret and sends to the server.
2. Only the server has the matching private key to decrypt the secret (not the Eavesdroppers).
3. The client and server then use this secret to generate a session key independently and simultaneously.

This session key would then be used for secure communication for this particular communication session

1. The client generates a 48-byte (384-bit) random number called pre_master_secret, encrypts it using the verified server's public key and sends it to the server.

2. Server decrypts the pre_master_secret using its own private key. Eavesdroppers cannot decrypt the pre_master_secret, as they do not possess the server's private key.
3. Client and server then independently and simultaneously create the session key, based on the pre_master_secret, client_random and server_random. Notice that both the server and client contribute to the session key, through the inclusion of the random number exchange in the hello messages. Eavesdroppers can intercept client_random and server_random as they are sent in plaintext, but cannot decrypt the pre_master_secret.
4. In a SSL/TLS session, the session key consists of 6 secret keys (to thwart crypto-analysis). 3 secret keys are used for client-to-server messages, and the other 3 secret keys are used for server-to-client messages. Among the 3 secret keys, one is used for encryption (e.g., DES secret key), one is used for message integrity (e.g., HMAC) and one is used for cipher initialization. (Cipher initialization uses a random plaintext called Initial Vector (IV) to prime the cipher pump.)
5. Client and server use the pre_master_secret (48-byte random number created by the client and exchange securely), client_random, server_random, and a pseudo-random function (PRF) to generate a master_secret. They can use the master_secret, client_random, server_random, and the pseudo-random function (PRF) to generate all the 6 shared secret keys. Once the secret keys are generated, the pre_master_secret is no longer needed and should be deleted.
6. From this point onwards, all the exchanges are encrypted using the session key.
7. The client sends Finished handshake message using their newly created session key. Server responds with a Finished handshake message.

Message Exchange

Client and server can use the agreed-upon session key (consists of 6 secret keys) for secure exchange of messages.

Sending messages:

1. The sender compresses the message using the agreed-upon compression method (e.g., PKZip, gzip).

2. The sender hashes the compressed data and the secret HMAC key to make an HMAC, to assure message integrity.
3. The sender encrypts the compressed data and HMAC using encryption/decryption secret key, to assure message confidentiality.

Retrieve messages:

1. The receiver decrypts the ciphertext using the encryption/decryption secret key to retrieve the compressed data and HMAC.
2. The receiver hashes the compressed data to independently produce the HMAC. It then verifies the generated HMAC with the HMAC contained in the message to assure message integrity.
3. The receiver un-compresses the data using the agreed-upon compression method to recover the plaintext.

OUTPUT

> **openssl s_client ?**

(Display the available options)

The following command turns on the debug option and forces the protocol to be TLSv1:

> **openssl s_client -connect localhost:443 -CAfile ca.crt -debug -tls1**

```
Loading 'screen' into random state - done
CONNECTED(00000760)

write to 00988EB0 [009952C8] (102 bytes => 102 (0x66))
0000 - 16 03 01 00 61 01 00 00-5d 03 01 40 44 35 27 5c   ....a...]..@D5'\
0010 - 5a e8 74 26 e9 49 37 e2-06 3b 1c 6d 77 37 d1 ae   Z.t&.I7..;.mw7..
0020 - 44 07 86 47 98 fa 84 1a-8d f4 72 00 00 36 00 39   D..G......r..6.9
0030 - 00 38 00 35 00 16 00 13-00 0a 00 33 00 32 00 2f   .8.5.......3.2./
0040 - 00 07 00 66 00 05 00 04-00 63 00 62 00 61 00 15   ...f.....c.b.a..
0050 - 00 12 00 09 00 65 00 64-00 60 00 14 00 11 00 08   .....e.d.`......
0060 - 00 06 00 03 01                                    .....
0066 - <SPACES/NULS>

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
0000 - 16 03 01 00 2a                                    ....*
```

```
read from 00988EB0 [00990ABD] (42 bytes => 42 (0x2A))
0000 - 02 00 00 26 03 01 40 44-35 27 cc ef 2b 51 e1 b0   ...&..@D5'..+Q..
0010 - 44 1f ef c4 83 72 df 37-4f 9b 2b dd 11 50 13 87   D....r.7O.+..P..
0020 - 91 0a a2 d2 28 b9 00 00-16                        ....(....
002a - <SPACES/NULS>

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
0000 - 16 03 01 02 05                                    .....

read from 00988EB0 [00990ABD] (517 bytes => 517 (0x205))
0000 - 0b 00 02 01 00 01 fe 00-01 fb 30 82 01 f7 30 82   ..........0...0.
0010 - 01 60 02 01 01 30 0d 06-09 2a 86 48 86 f7 0d 01   .`...0...*.H....
0020 - 01 04 05 00 30 4d 31 0b-30 09 06 03 55 04 06 13   ....0M1.0...U...
0030 - 02 55 53 31 10 30 0e 06-03 55 04 0b 13 07 74 65   .US1.0...U....te
0040 - 73 74 31 30 31 31 0c 30-0a 06 03 55 04 03 13 03   st1011.0...U....
0050 - 63 68 63 31 1e 30 1c 06-09 2a 86 48 86 f7 0d 01   chc1.0...*.H....
0060 - 09 01 16 0f 63 68 63 40-74 65 73 74 31 30 31 2e   ....chc@test101.
0070 - 63 6f 6d 30 1e 17 0d 30-34 30 32 32 36 30 36 35   com0...040226065
0080 - 36 35 34 5a 17 0d 30 35-30 32 32 35 30 36 35 36   654Z..0502250656
0090 - 35 34 5a 30 3b 31 0b 30-09 06 03 55 04 06 13 02   54Z0;1.0...U....
00a0 - 55 53 31 0c 30 0a 06 03-55 04 03 13 03 63 68 63   US1.0...U....chc
00b0 - 31 1e 30 1c 06 09 2a 86-48 86 f7 0d 01 09 01 16   1.0...*.H.......
00c0 - 0f 63 68 63 40 74 65 73-74 31 30 31 2e 63 6f 6d   .chc@test101.com
00d0 - 30 81 9f 30 0d 06 09 2a-86 48 86 f7 0d 01 01 01   0..0...*.H......
00e0 - 05 00 03 81 8d 00 30 81-89 02 81 81 00 cd e4 9e   ......0.........
00f0 - 7c b6 d2 34 4e d3 53 46-25 c7 53 88 25 60 e6 46   |..4N.SF%.S.%`.F
0100 - db 64 3a 73 61 92 ac 23-92 cd 2c 94 a9 8f c6 7f   .d:sa..#..,.....
0110 - 47 73 c0 d9 8d 34 b7 2c-dd c9 86 bd 82 6f ce ac   Gs...4.,.....o..
0120 - d8 e2 ba 0f e5 f5 3a 67-2c 89 1a 1b 03 eb 21 85   ......:g,.....!.
0130 - 28 e3 29 98 84 ed 46 75-82 fa 0f 30 a3 a9 a5 71   (.)...Fu...0...q
0140 - 46 4c d6 0d 17 c4 19 fd-44 fb e2 18 46 a6 9d ab   FL......D...F...
0150 - 91 de 6b a1 7f fe 30 06-28 5d d8 d3 29 00 c3 1d   ..k...0.(]..)...
0160 - 4c 13 00 61 8f f3 85 51-f5 68 d8 69 25 02 03 01   L..a...Q.h.i%...
0170 - 00 01 30 0d 06 09 2a 86-48 86 f7 0d 01 01 04 05   ..0...*.H.......
0180 - 00 03 81 81 00 29 fd bf-5a ed 70 8f 53 a4 e9 14   .....)..Z.p.S...
0190 - 4c 5e ba 84 c6 54 1b f2-c0 3c c4 30 0f 7f 12 80   L^...T...<.0....
```

```
01a0 - 4e 01 b7 fd 39 50 f1 41-0d d8 aa 77 d9 87 25 1a   N...9P.A...w..%.
01b0 - 1e e2 97 88 4f 53 75 c8-70 22 6a 01 61 0f 51 3e   ....OSu.p"j.a.Q>
01c0 - 13 19 9c 64 f2 76 14 e8-85 25 23 a2 11 c4 8c f8   ...d.v...%#.....
01d0 - 23 2c d1 c3 d3 71 3a e6-71 54 10 07 dc 72 ff ee   #,...q:.qT...r..
01e0 - e8 3e cf 8e 77 73 e9 9f-f5 9a 90 60 4d a0 aa 03   .>..ws.....`M...
01f0 - 32 1f 11 6f 2e 9a 5f 3c-77 05 22 0c 81 bf 29 96   2..o.._ 5 (0x5))
0000 - 16 03 01 01 8d                                    .....

read from 00988EB0 [00990ABD] (397 bytes => 397 (0x18D))
0000 - 0c 00 01 89 00 80 e6 96-9d 3d 49 5b e3 2c 7c f1   .........=I[.,|.
0010 - 80 c3 bd d4 79 8e 91 b7-81 82 51 bb 05 5e 2a 20   ....y.....Q..^*
0020 - 64 90 4a 79 a7 70 fa 15-a2 59 cb d5 23 a6 a6 ef   d.Jy.p...Y..#...
0030 - 09 c4 30 48 d5 a2 2f 97-1f 3c 20 12 9b 48 00 0e   ..0H../..< ..H..
0040 - 6e dd 06 1c bc 05 3e 37-1d 79 4e 53 27 df 61 1e   n.....>7.yNS'.a.
0050 - bb be 1b ac 9b 5c 60 44-cf 02 3d 76 e0 5e ea 9b   .....\`D..=v.^..
0060 - ad 99 1b 13 a6 3c 97 4e-9e f1 83 9e b5 db 12 51   .....<.N.......Q
0070 - 36 f7 26 2e 56 a8 87 15-38 df d8 23 c6 50 50 85   6.&.V...8..#.PP.
0080 - e2 1f 0d d5 c8 6b 00 01-02 00 80 11 3f 5f fa e4   .....k......?_..
0090 - 79 9a 0b d9 e0 67 37 c4-2a 88 22 b0 95 b7 a7 be   y....g7.*.".....
00a0 - 93 79 9d 51 ae 31 47 99-df 47 dd 80 5e 3d 2a 4a   .y.Q.1G..G..^=*J
00b0 - 29 8b fd c1 63 5e 48 e8-e3 fd ac 95 1b 3a 5f 75   )...c^H......:_u
00c0 - 98 2d 3c 9c ba 68 18 7b-be 38 2c 69 3d 41 b7 c3   .-<..h.{.8,i=A..
00d0 - 08 a1 da b0 a8 a4 fe 9a-d6 1e 56 ff 4c 8c 6e 6b   ..........V.L.nk
00e0 - 18 f1 ec 9d 22 a9 90 27-c1 c6 2c 0e bd 0e 13 d4   ...."..'..,.....
00f0 - fd b2 c9 8f 6f bb 8e 06-e0 b5 1f f7 87 03 5f a8   ....o........_.
0100 - 12 4f bb ce ba f1 76 fb-80 08 37 00 80 30 99 ad   .O....v...7..0..
0110 - 9b fc 3a 14 6b a8 2c c5-fe 7b bd 1c 92 ec 19 a6   ..:.k.,..{......
0120 - 75 2d 69 4e f4 9f 74 60-5d d4 3e 06 97 38 bc b5   u-iN..t`].>..8..
0130 - 0e 3c 1f f2 99 e6 55 4a-36 42 a8 f2 b7 32 2a 1e   .<....UJ6B...2*.
0140 - a3 87 b3 f3 79 43 28 d1-7a 0d db 7c 11 26 f3 68   ....yC(.z..|.&.h
0150 - b1 73 b6 78 4b f3 22 20-e4 f7 27 08 ab 74 92 92   .s.xK." ..'..t..
0160 - 79 26 61 40 1e e9 90 11-e8 b1 cf 99 d9 9f c7 68   y&a@...........h
0170 - 48 e8 f2 a5 d5 d7 0e e1-88 9a bd 0f 40 85 af 2d   H..........@..-
0180 - da 76 3a 10 6e b9 38 4d-37 9c 41 c8 9f            .v:.n.8M7.A..

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
0000 - 16 03 01 00 04                                    .....
```

```
read from 00988EB0 [00990ABD] (4 bytes => 4 (0x4))
0000 - 0e                                               .
0004 - <SPACES/NULS>

write to 00988EB0 [00999BE0] (139 bytes => 139 (0x8B))
0000 - 16 03 01 00 86 10 00 00-82 00 80 63 c2 3c 69 26   ...........c...dU.....]n..
0030 - 05 f1 db 44 f3 13 a8 24-3a 76 0e 3e 1a 6e 55 0c   ...D...$:v.>.nU.
0040 - 31 9b 04 99 30 ff 8f d2-8d 8e 0d b1 67 ac 43 ee   1...0.......g.C.
0050 - b2 3f d3 c7 c5 33 81 e1-3f d2 47 6f 5d 8a fb 4c   .?...3..?.Go]..L
0060 - 62 c7 23 b3 f7 ad 3c a9-0c 87 4a 08 07 55 ba 06   b.#...<...J..U..
0070 - 34 18 0c 5f d9 35 f0 2b-90 9a 9d 6b 87 62 41 0f   4.._.5.+...k.bA.
0080 - b3 47 74 5f 5b b8 59 5a-b2 21 dd                  .Gt_[.YZ.!.

write to 00988EB0 [00999BE0] (6 bytes => 6 (0x6))
0000 - 14 03 01 00 01 01                                ......

write to 00988EB0 [00999BE0] (45 bytes => 45 (0x2D))
0000 - 16 03 01 00 28 0f 31 83-e0 f8 91 fa 33 98 68 46   ....(.1.....3.hF
0010 - c0 60 83 66 28 fe d3 a5-00 f0 98 d5 df 22 72 2d   .`.f(........"r-
0020 - e4 40 9b 96 3b 4c f9 02-13 a7 e7 77 74            .@..;L.....wt

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
0000 - 14 03 01 00 01                                   .....

read from 00988EB0 [00990ABD] (1 bytes => 1 (0x1))
0000 - 01                                               .

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
0000 - 16 03 01 00 28                                   ....(

read from 00988EB0 [00990ABD] (40 bytes => 40 (0x28))
0000 - d4 0b a6 b7 e8 91 09 1e-e4 1e fc 44 5f 80 cc a1   ...........D_...
0010 - 5d 51 55 3e 62 e8 0f 78-07 f6 2f cd f9 bc 49 8d   ]QU>b..x../...I.
0020 - 56 5b e8 b2 09 2c 18 52-                         V[...,.R
---
```

```
Certificate chain
 0 s:/C=US/CN=chc/emailAddress=chc@test101.com
   i:/C=US/OU=test101/CN=chc/emailAddress=chc@test101.com
---

Server certificate
-----BEGIN CERTIFICATE-----
MIIB9zCCAWACAQEwDQYJKoZIhvcNAQEEBQAwTTELMAkGA1UEBh
MCVVMxEDAOBgNV
BAsTB3Rlc3QxMDExDDAKBgNVBAMTA2NoYzEeMBwGCSqGSIb3DQEJ
ARYPY2hjQHRl
c3QxMDEuY29tMB4XDTA0MDIyNjA2NTY1NFoXDTA1MDIyNTA2NTY1
NFowOzELMAkG
A1UEBhMCVVMxDDAKBgNVBAMTA2NoYzEeMBwGCSqGSIb3DQEJA
RYPY2hjQHRlc3Qx
MDEuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDN5J58
ttI0TtNTRiXH
U4glYOZG22Q6c2GSrCOSzSyUqY/Gf0dzwNmNNLcs3cmGvYJvzqzY4roP5f
U6ZyyJ
GhsD6yGFKOMpmITtRnWC+g8wo6mlcUZM1g0XxBn9RPviGEamnauR3mu
hf/4wBihd
2NMpAMMdTBMAYY/zhVH1aNhpJQIDAQABMA0GCSqGSIb3DQEBBA
UAA4GBACn9v1rt
cI9TpOkUTF66hMZUG/LAPMQwD38SgE4Bt/05UPFBDdiqd9mHJRoe4peIT
1N1yHAi
agFhD1E+ExmcZPJ2FOiFJSOiEcSM+CMs0cPTcTrmcVQQB9xy/+7oPs+Od3
Ppn/Wa
kGBNoKoDMh8Rby6aXzx3BSIMgb8plq3LOxiu
-----END CERTIFICATE-----

subject=/C=US/CN=chc/emailAddress=chc@test101.com
issuer=/C=US/OU=test101/CN=chc/emailAddress=chc@test101.com
---

No client certificate CA names sent
---
```

```
SSL handshake has read 1031 bytes and written 292 bytes
---

New, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
Server public key is 1024 bit
SSL-Session:
    Protocol  : TLSv1
    Cipher    : EDH-RSA-DES-CBC3-SHA
    Session-ID:
    Session-ID-ctx:
    Master-Key:
57FDDAF85C7D287F9F9A070E8784A29C75E788DA2757699B
                20F3CA50E7EE01A66182A71753B78DA218916136D50861AE
    Key-Arg   : None
    Start Time: 1078211879
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
---
```

GET /test.html HTTP/1.0

```
write to 00988EB0 [009952C8] (82 bytes => 82 (0x52))
0000 - 17 03 01 00 18 74 fa 45-35 2d b1 24 59 cf ad 96   .....t.E5-.$Y...
0010 - 34 30 01 7d be 8e 70 f9-41 62 11 f1 36 17 03 01   40.}..p.Ab..6...
0020 - 00 30 56 61 ba 2d d3 58-5d e6 6a 83 78 07 87 7a   .0Va.-.X].j.x..z
0030 - db b2 a7 40 c7 6d c1 4a-20 3b 82 7d aa 15 e8 65   ...@.m.J ;.}...e
0040 - 3b 92 bd c8 20 e9 9d 41-f1 77 51 d9 ae 31 c4 2c   ;... ..A.wQ..1.,
0050 - 32 5a                                             2Z

write to 00988EB0 [009952C8] (58 bytes => 58 (0x3A))
0000 - 17 03 01 00 18 39 2f df-43 75 91 13 34 1b 12 04   .....9/.Cu..4...
0010 - 7d ef 8d e1 86 54 4f 67-c8 1d cd 07 a4 17 03 01   }....TOg........
0020 - 00 18 53 d9 22 9d eb 6e-8b 79 f8 e4 82 2f ba ea   ..S."..n.y.../..
0030 - 03 a5 3f 12 85 2e 9f 64-ff dc                    ..?....d..

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
0000 - 17 03 01 01 48                                    ....H
```

```
read from 00988EB0 [00990ABD] (328 bytes => 328 (0x148))
0000 - bd eb 8b 9c 01 ac 73 30-8f ca a4 8b 2a 6f bd 02   ......s0....*o..
0010 - d7 fc 71 18 61 47 f2 1d-70 8b 10 7d 98 28 a4 50   ..q.aG..p..}.(.P
0020 - f3 0f 42 e8 c5 e1 3e 53-34 bd c7 62 34 1b 5e 8c   ..B...>S4..b4.^.
0030 - 99 2d 89 c6 b3 f0 19 96-22 97 43 b8 8f 9d 76 42   .-......".C...vB
0040 - 95 a5 7c db 3b 22 dd 57-29 8d e8 d4 28 3e 89 d8   ..|.;".W)...(>..
0050 - 46 e5 dc 35 51 56 f8 44-d1 82 44 a0 65 b0 93 22   F..5QV.D..D.e.."
0060 - 4b 0a eb 07 26 c9 2a e2-45 4c de 07 0c bb 3e c6   K...&.*.EL....>.
0070 - bc 37 94 cd ec 94 2f 35-76 37 13 4d 0f 88 9c b1   .7..../5v7.M....
0080 - d7 1c 58 8a 35 5b 32 bc-12 2b 9c e6 5b d4 86 bd   ..X.5[2..+..[...
0090 - 39 fc 99 18 79 ec f7 53-db 59 74 49 da 07 69 54   9...y..S.YtI..iT
00a0 - f4 66 aa 36 34 39 f9 0b-87 50 9e 76 db 9f d0 44   .f.649...P.v...D
00b0 - 0c 0d e7 65 80 9b b8 51-56 3d d0 db aa 55 ff ca   ...e...QV=...U..
00c0 - 74 38 24 c1 8c d7 32 cf-ab 03 b3 59 29 0f 80 18   t8$...2....Y)...
00d0 - 6a d4 e0 7e fd 41 8c f7-1d 81 12 a7 00 b3 71 39   j..~.A........q9
00e0 - 78 1e 3c 17 42 d4 99 22-69 7b 2d 09 ef d8 6e f4   x.<.B.."i{-...n.
00f0 - 64 f6 61 34 72 8c 89 f5-a8 ea 1c b1 0d 08 ff 17   d.a4r...........
0100 - 51 3e 46 2b 38 75 61 6a-1e 34 f4 14 14 38 0d 5e   Q>F+8uaj.4...8.^
0110 - 6e ba db ef 83 88 ee a5-2c 18 5a 0c 27 e3 d9 19   n.......,.Z.'...
0120 - 6c a3 12 c0 a1 3d e1 14-96 d3 1a f9 c9 f2 aa d6   l....=..........
0130 - 12 d5 36 ae 36 f2 18 f5-df c6 ef 34 d7 7d 2b 70   ..6.6......4.}+p
0140 - 99 88 47 93 91 09 56 b1-                          ..G...V.

HTTP/1.1 200 OK
Date: Tue, 02 Mar 2004 07:18:08 GMT
Server: Apache/1.3.29 (Win32) mod_ssl/2.8.16 OpenSSL/0.9.7c
Last-Modified: Sat, 07 Feb 2004 10:53:25 GMT
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>Home page on main server</h1>

read from 00988EB0 [00990AB8] (5 bytes => 5 (0x5))
```

```
0000 - 15 03 01 00 18                              .....

read from 00988EB0 [00990ABD] (24 bytes => 24 (0x18))
0000 - a5 47 51 bd aa 0f 9b e4-ac d4 28 f2 d0 a0 c8 fa   .GQ.......(.....
0010 - 2c d4 e5 e4 be c5 01 85-                          ,.......

closed

write to 00988EB0 [009952C8] (29 bytes => 29 (0x1D))
0000 - 15 03 01 00 18 d4 19 b9-59 88 88 c0 c9 38 ab 5c   ........Y....8.\
0010 - 98 8c 43 fd b8 9e 14 3d-77 5e 4c 68 03            ..C....=w^Lh.
```

RESULT:

Thus, the check message intergrity and confidentiality using SSL can verified the output successfully.

Ex:No:06	EXPERIMENT EAVESDROPPING, DICTIONARY ATTACKS, MITM ATTACK

AIM

To experiment eavesdropping, dictionary attack, MITM attack.

PROCEDURE

Man in the Middle (MITM) against Diffie-Hellman:

A malicious Malory, that has a MitM (man in the middle) position, can manipulate the communications between Alice and Bob, and break the security of the key exchange.

1. Selected public numbers p and g, p is a prime number, called the "modulus" and g is called the base.
2. Selecting private numbers.
   let Alice pick a private random number a and let Bob pick a private random number b, Malory picks 2 random numbers c and d.
3. Intercepting public values,
   Malory intercepts Alice's public value (g^a(mod p)), block it from reaching Bob, and instead sends Bob her own public value (g^c(modp)) and Malory intercepts Bob's public value (g^b(mod p)), block it from reaching Alice, and instead sends Alice her own public value (g^d (modp))
4. Computing secret key
   Alice will compute a key $S_1=g^{da}$(mod p), and Bob will compute a different key, $S_2=g^{cb}$(mod p)
5. If Alice uses S_1 as a key to encrypt a later message to Bob, Malory can decrypt it, re-encrypt it using S_2, and send it to Bob. Bob and Alice won't notice any problem and may assume their communication is encrypted, but in reality, Malory can decrypt, read, modify, and then re-encrypt all their conversations.

PROGRAM:

```
import java.util.Random;
```

```
import java.util.Scanner;


public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();


        System.out.print("Enter a prime number : ");
        int p = scanner.nextInt();
        System.out.print("Enter a number : ");
        int g = scanner.nextInt();


        class A {
            private int n;


            public A() {
                this.n = random.nextInt(p) + 1;
            }


            public int publish() {
                return (int) Math.pow(g, n) % p;
```

```java
    }


    public int compute_secret(int gb) {
        return (int) Math.pow(gb, n) % p;
    }
}


class B {
    private int a;
    private int b;
    private int[] arr;


    public B() {
        this.a = random.nextInt(p) + 1;
        this.b = random.nextInt(p) + 1;
        this.arr = new int[]{a, b};
    }
    public int publish(int i) {
        return (int) Math.pow(g, arr[i]) % p;
    }
```

```java
        public int compute_secret(int ga, int i) {
            return (int) Math.pow(ga, arr[i]) % p;
        }
    }
    A alice = new A();
    A bob = new A();
    B eve = new B();
    System.out.println("Alice selected (a) : " + alice.n);
    System.out.println("Bob selected (b) : " + bob.n);
    System.out.println("Eve selected private number for Alice (c) : " + eve.a);
    System.out.println("Eve selected private number for Bob (d) : " + eve.b);
    int ga = alice.publish();
    int gb = bob.publish();
    int gea = eve.publish(0);
    int geb = eve.publish(1);
    System.out.println("Alice published (ga): " + ga);
    System.out.println("Bob published (gb): " + gb);
    System.out.println("Eve published value for Alice (gc): " + gea);
    System.out.println("Eve published value for Bob (gd): " + geb);
  }
}
```

Output:

Enter a prime number (p) : 227

Enter a number (g) : 14

Alice selected (a) : 227

Bob selected (b) : 170

Eve selected private number for Alice (c) : 65

Eve selected private number for Bob (d) : 175

Alice published (ga): 14

Bob published (gb): 101

Eve published value for Alice (gc): 41

Eve published value for Bob (gd): 32

Alice computed (S1) : 41

Eve computed key for Alice (S1) : 41

Bob computed (S2) : 167

Eve computed key for Bob (S2) : 167

RESULT

Thus, the above program experiment eavesdropping, dictionary attacks, MITM attacks are executed successfully and output are verified.

Ex:No:07	EXPERIMENT WITH SNIFF TRAFFIC USING ARP POISONING

AIM

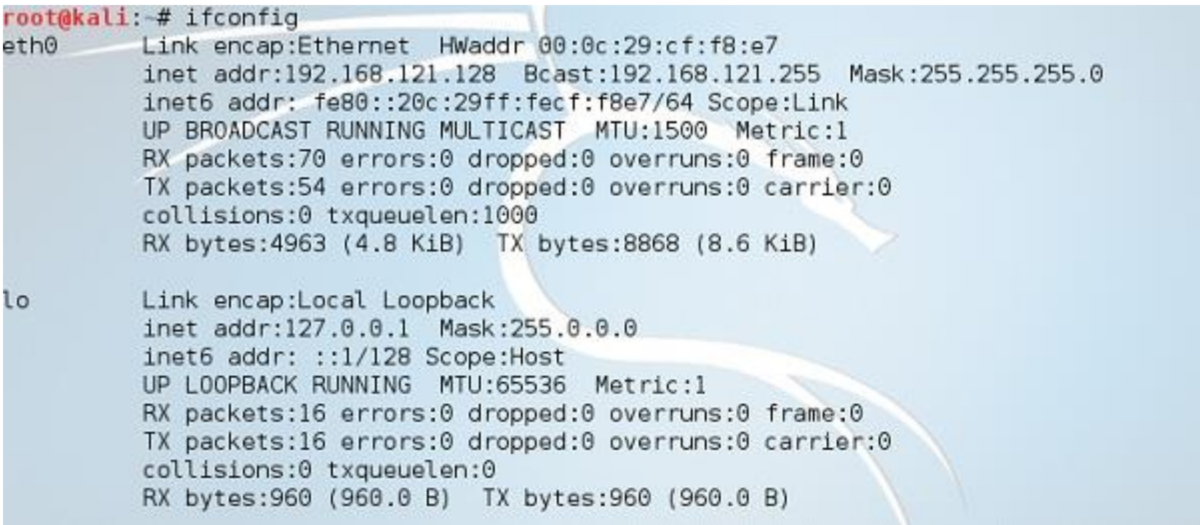
To experiment with sniff traffic using ARP poisoning.

PROCEDURE

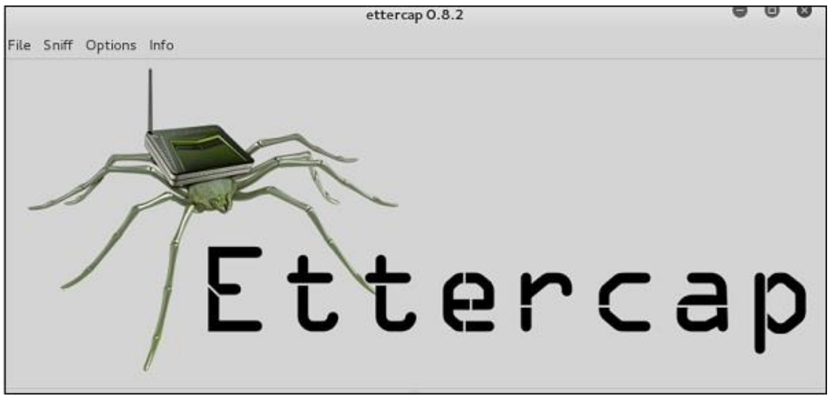
Step 1 − Install the VMware workstation and install the Kali Linux operating system.

Step 2 − Login into the Kali Linux using username pass "root, toor".

Step 3 − Make sure you are connected to local LAN and check the IP address by typing the command **ifconfig** in the terminal.



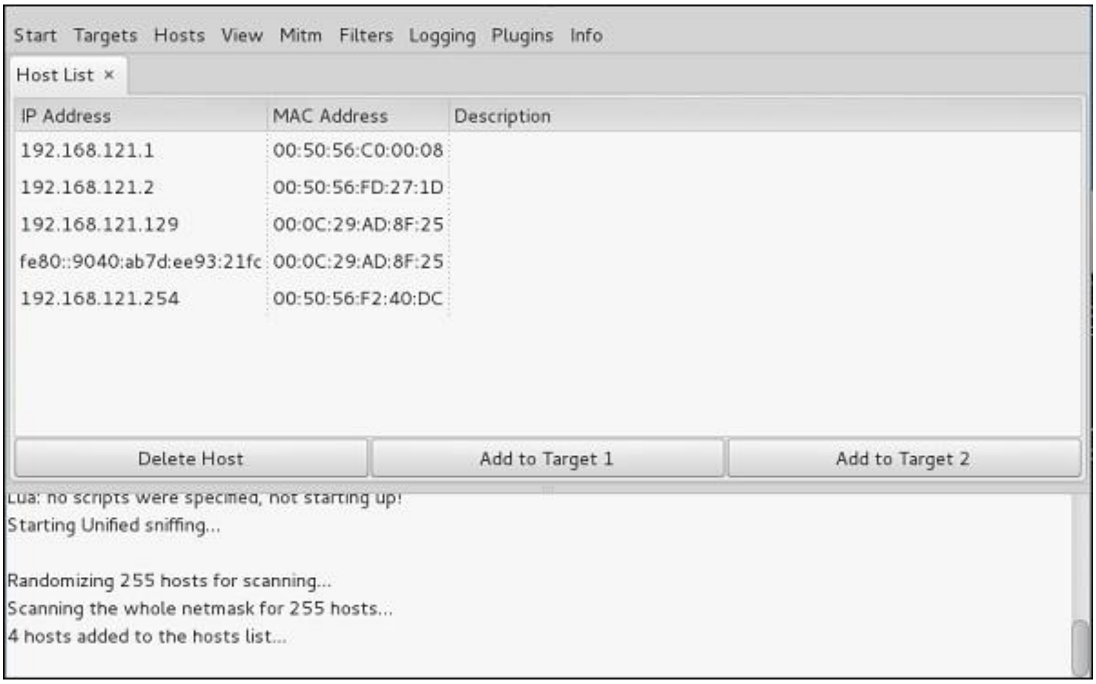
Step 4 − Open up the terminal and type "Ettercap –G" to start the graphical version of Ettercap.



Step 5 – Now click the tab “sniff” in the menu bar and select “unified sniffing” and click OK to select the interface. We are going to use “eth0” which means Ethernet connection.

Step 6 – Now click the “hosts” tab in the menu bar and click “scan for hosts”. It will start scanning the whole network for the alive hosts.

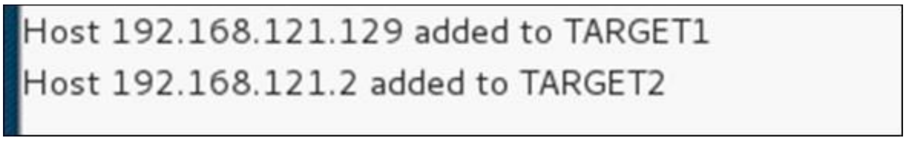
Step 7 – Next, click the “hosts” tab and select “hosts list” to see the number of hosts available in the network. This list also includes the default gateway address. We have to be careful when we select the targets.



Step 8 – Now we have to choose the targets. In MITM, our target is the host machine, and the route will be the router address to forward the traffic. In an MITM attack, the attacker intercepts the network and sniffs the packets. So, we will add the victim as “target 1” and the router address as “target 2.”

In VMware environment, the default gateway will always end with "2" because "1" is assigned to the physical machine.

Step 9 − In this scenario, our target is "192.168.121.129" and the router is "192.168.121.2". So we will add target 1 as **victim IP** and target 2 as **router IP**.



Step 10 − Now click on "MITM" and click "ARP poisoning". Thereafter, check the option "Sniff remote connections" and click OK.

Step 11 − Click "start" and select "start sniffing". This will start ARP poisoning in the network which means we have enabled our network card in "promiscuous mode" and now the local traffic can be sniffed.

Note − We have allowed only HTTP sniffing with Ettercap, so don't expect HTTPS packets to be sniffed with this process.

Step 12 − Now it's time to see the results; if our victim logged into some websites. You can see the results in the toolbar of Ettercap.

RESULT:

Thus, the above experiment with sniff traffic using ARP poisoning are executed successfully and output are verified.

Ex:No:08	DEMONSTRATE INTRUSION DETECTION SYSTEM USING ANY TOOL

AIM

To demonstrate intrusion detection system using any tool (SNORT).

PROCEDURE

In Windows:

- **Step-1:** Download SNORT installer from https://www.snort.org/downloads/snort/Snort_2_9_15_Installer.exe
- **Step-1:** Execute the Snort_2_9_15_Installer.exe

Different SNORT Modes:

1. **Sniffer Mode –**
   To print TCP/IP header use command **./snort -v**
   To print IP address along with header use command **./snort -vd**
2. **Packet Logging –**
   To store packet in disk you need to give path where you want to store the logs. For this command is**./snort -dev -l ./SnortLogs**.
3. **Activate network intrusion detection mode –**

To start this mode use this command **./snort -dev -l ./SnortLogs -h 192.127.1.0/24 -c snort.conf**

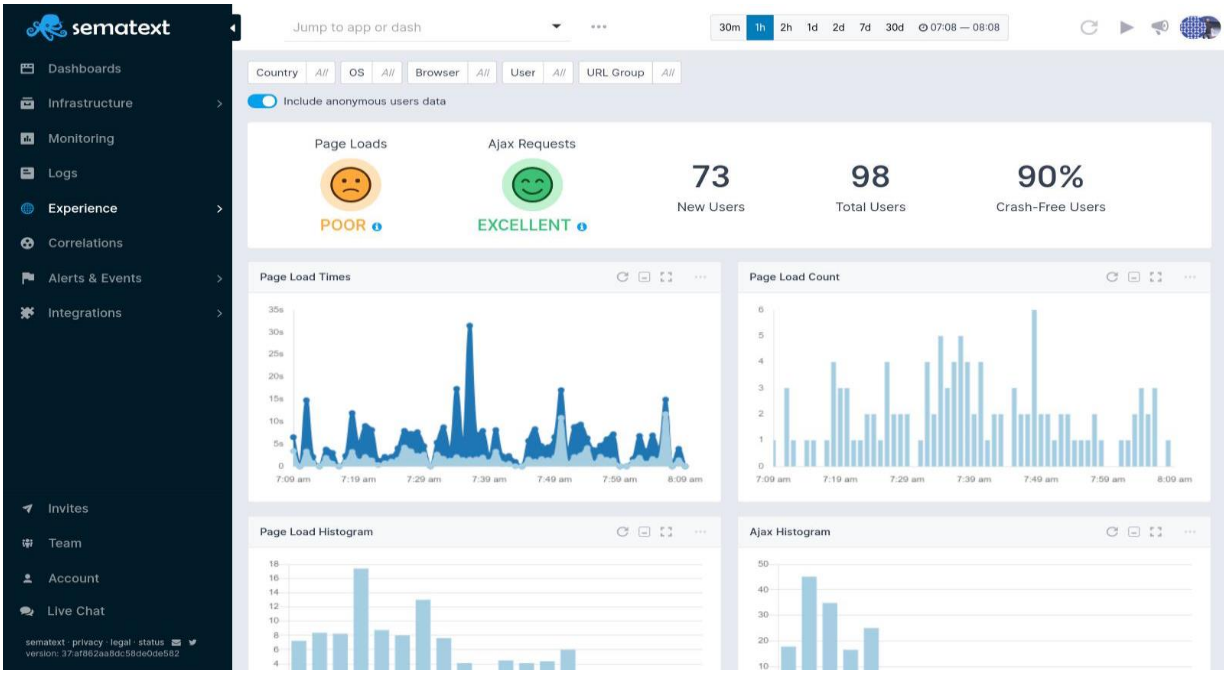
RESULT

Thus the above demonstrate intrusion system using SNORT are installed successfully and output are verified.

Ex:No:09	EXPLORE NETWORK MONITORING TOOL

PROCEDURE

1. Sematext Experience



Sematext Experience is a real user monitoring solution that offers **100% visibility** into your website or web app that affects your users' experience.

Here is what puts Sematext on the top of our list:

- Easy installation
- Single page application support
- Individual session performance
- Inspect Page load events
- Monitor your Apdex score
- Real-time automatic alerts

Sematext Experience allows you to inspect **individual sessions** to get **page-level specifics**. This helps assess the user's satisfaction to prevent customer loss due to poor performance.

Furthermore, you can set up alerts for **Apdex score**, **script errors**, and **page load time** and receive **real-time notifications** whenever performance anomalies are detected. this, in turn, will enable you to troubleshoot issues faster.

SEMATEXT EXPERIENCE

Sematext Experience was designed so DevOps and BizOps can work together. Having easy access to all your actionable data provides your whole team with in-depth insights. With this data, effectual decisions can be made with ease to ensure your customers are always satisfied.

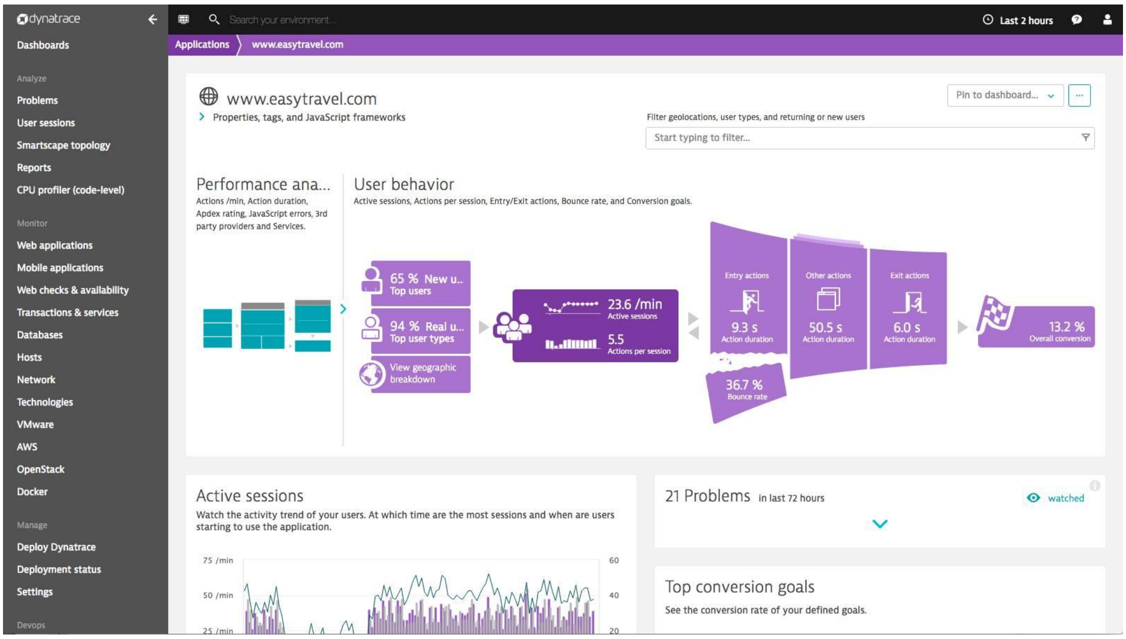
Pricing

- From $9/mo

Pros

- Combine the power of metrics, logs, and end-user monitoring under one roof with Sematext Cloud
- First-class support for popular frontend frameworks such as React, Ember, and Angular
- URL grouping for both page-load events and HTTP requests
- Powerful cost control using data sampling
- Has a solution for synthetic monitoring
- Error tracking

2. Dynatrace RUM



Part of Dynatrace's digital experience monitoring toolset, Dynatrace RUM is a powerful website monitoring service that offers complete real-time visibility of customer experience. You can monitor the activity of all mobile and web application users across all devices and browsers to assess and improve user satisfaction.

With Dynatrace RUM you can also collect business-relevant metrics, allowing you to correlate performance issues with potential business impact.

Features

- Map the whole user journey
- Replay individual customer sessions
- Business-relevant, user transaction monitoring
- Real-time AI-based analysis

Pricing

- Available on request

Pros

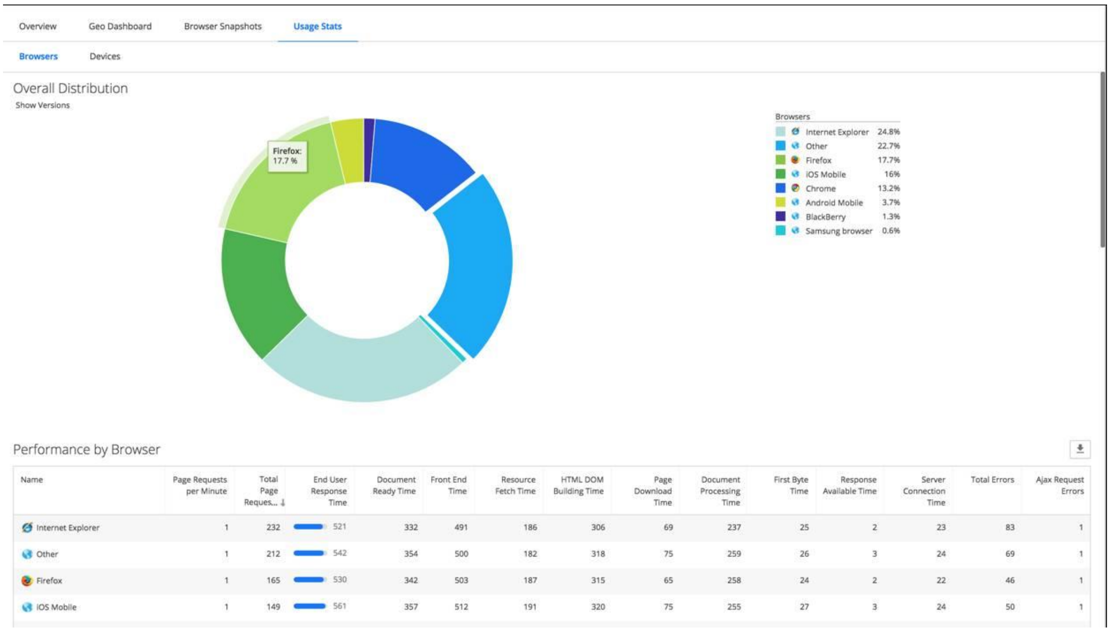
- Intuitive non-technical dashboard usability
- Interactive interfaces and visual reports for ROI tracking

- Mobile monitoring breakdowns

Cons

- Reportedly pricey
- The UI can be overwhelming at first

3. AppDynamics Browser RUM



AppDynamics's RUM tool tracks customers' journey to provide full visibility into their interaction with your webapp. You receive browser-user insights to help you optimize web experiences. Self-learning algorithms use the app's behavior to dynamically baseline web metrics with automatic anomaly detection and resolution.

Features

- Real-time intelligent alerting
- Backend and frontend monitoring in same solution
- Business transaction correlation
- Browser snapshot waterfalls
- Dynamic performance baselining

Pricing

- Available in two options: Lite (free) version and Pro version. Pricing available on request

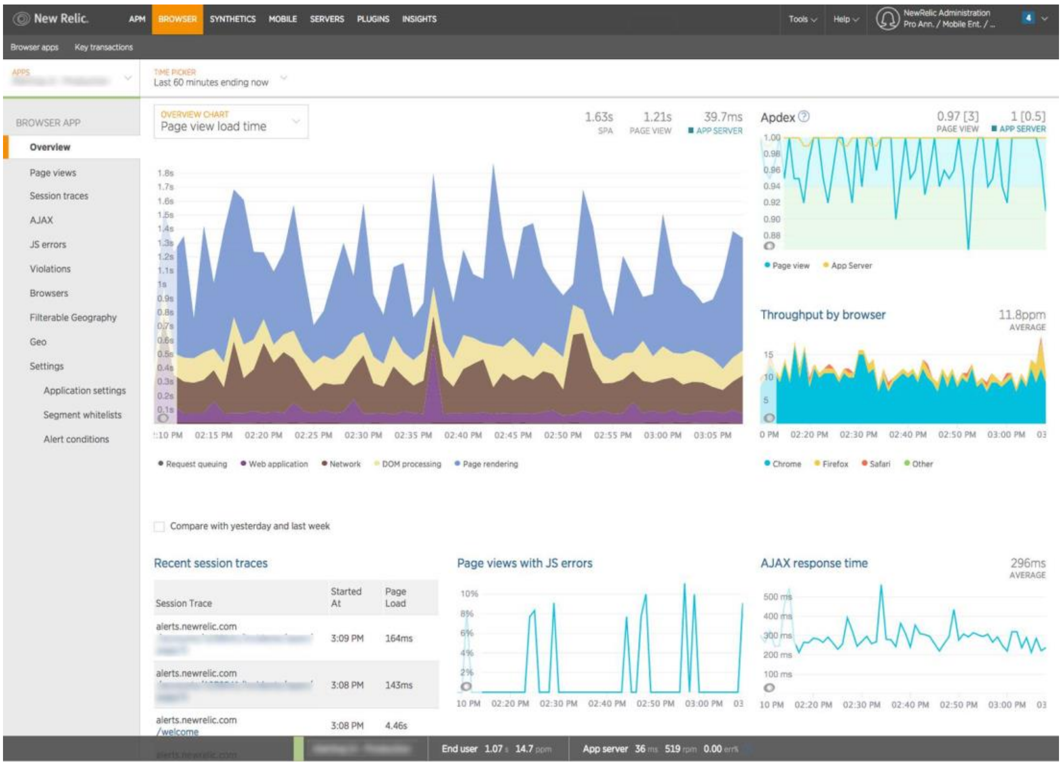
Pros

- Free training
- Self-learning platform

Cons

- Reportedly pricey

4. New Relic Browser



New Relic is mostly known for their APM tool, but they completed their monitoring tools set with a RUM solution, New Relic Browser.

New Relic Browser has advanced RUM features that give you access to insights from the users' perspective by focusing on browser performance. It monitors the entire life cycle of a page or a view, from the moment users enter the app until they disconnect.

Features

- Browser Pageviews and Page Load Times
- Java Errors and Instance details
- AJAX Timing and Call Reports
- Browser Session Traces
- Filterable Geography Analytics
- Route changes in apps with single page application (SPA) architecture
- Individual session performance

Pricing

- Pricing information available on request. Also has a free (Lite) version with fewer features

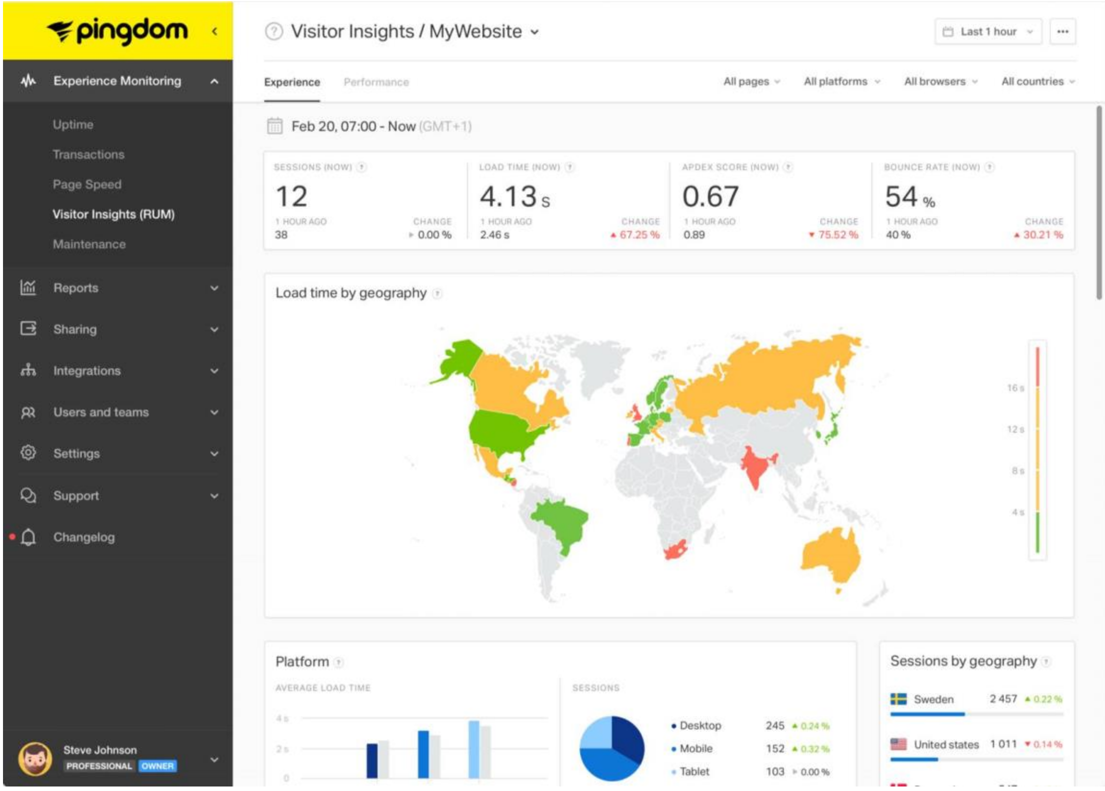
Pros

- Synthetic monitoring option available

Cons

- Most features are available for Pro accounts only
- Reports are not very comprehensive
- Missing detailed HTTP resources metrics

5. Pingdom



Pingdom is a unified performance monitoring tool that brings together transaction, uptime, and real user monitoring.

Pingdom allows you to filter data from specific users to get greater insights on the regional performance of your website and make optimizations to deliver a better experience to your most valuable users. It's highly scalable, allowing you to monitor millions of pageviews without compromising your data.

Features

- Tailored incident management
- Real-time data and alerting
- Website and server monitoring
- Mobile accessibility

Pricing

- The basic setup starts at $10/month, up to $199 – $15,000

Pros

- Customizable, fast and comprehensive alerting and reporting
- Synthetic and end user monitoring
- Notifications to multiple destinations (text message, email)

Cons

- Expensive if you increase volume or scale up as there is no data sampling available
- No error tracking or error management

RESULT:

Thus, the above process are explore network monitoring tools and view the output

Ex:No:10	STUDY TO CONFIGURE FIREWALL, VPN

AIM

To study to configure firewall, VPN using Google cloud services.

PROCEDURE

Google Cloud firewall rules

Google Cloud firewall rules apply to packets sent to and from virtual machine (VM) instances within your VPC network and through Cloud VPN tunnels.

Consolegcloud

1. In the Google Cloud console, go to the **VPN tunnels** page.
2. Go to VPN tunnels
3. Click the VPN tunnel that you want to use.
4. In the **VPN gateway** section, click the name of the VPC network. This action directs you to the **VPC network details** page that contains the tunnel.
5. Click the **Firewall rules** tab.
6. Click **Add firewall rule**. Add a rule for TCP, UDP, and ICMP:
   - **Name:** Enter allow-tcp-udp-icmp.
   - **Source filter:** Select **IPv4 ranges**.
   - **Source IP ranges:** Enter a **Remote network IP range** value from when you created the tunnel. If you have more than one peer network range, enter each one. Press the **Tab** key between entries. To allow traffic from all source IPv4 addresses in your peer network, specify 0.0.0.0/0.
   - **Specified protocols or ports:** Select tcp and udp.
   - **Other protocols:** Enter icmp.
   - **Target tags:** Add any valid tag or tags.
7. Click **Create**.

If you need to allow access to IPv6 addresses on your VPC network from your peer network, add an `allow-ipv6-tcp-udp-icmpv6` firewall rule.

Click **Add firewall rule**. Add a rule for TCP, UDP, and ICMPv6:

- **Name:** Enter allow-ipv6-tcp-udp-icmpv6.
- **Source filter:** Select **IPv6 ranges**.
- **Source IP ranges:** Enter a **Remote network IP range** value from when you created the tunnel. If you have more than one peer network range, enter each one. Press the **Tab** key between entries. To allow traffic from all source IPv6 addresses in your peer network, specify ::/0.
- **Specified protocols or ports:** Select tcp and udp.
- **Other protocols**: Enter 58. 58 is the protocol number for ICMPv6.
- **Target tags:** Add any valid tag or tags.

Click **Create**.

CONCLUSION

The purpose of this study was to explore the role of the firewall in network security. This was done by researching five more specific problems. Two of them were concerned with the relationship between firewalls and network services, and it is in this area we believe this study makes its foremost contribution. With regard to the question about firewall configurations, our results are in line with findings from other studies, not least those by Wool. Realistically, we do not consider our results to be that revolutionary nor reliable. VPNs allow users or corporations to connect to remote servers, branch offices, or to other companies over a public internetwork, while maintaining secure communications. In all these cases, the secure connection appears to the user as a private network communication-despite the fact that this communication occurs over a public internetwork. VPN technology is designed to address issues surrounding the current business trend towards increased telecommuting and widely distributed global operations, where workers must be able to connect to central resources and communicate with each other.