International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 16 (2015) pp 36371-36385
© Research India Publications. http://www.ripublication.com

Network Intrusion Detection using DPI Techniques in Wireless Network

Mr. Satish N. Gujar
JJT,University
Rajastan, India,
satishgujar@gmail.com

Dr. V. M. Thakare
S.G.B. Amravati University,
Amravati
Maharashtra, India
vilthakare@yahoo.co.in

Abstract— Deep Packet Inspection (DPI) is becoming more widely used in virtually all applications or services like Intrusion Detection System (IDS), which operate with or within a network. DPI analyzes all data present in the packet as it passes an inspection to determine the application transported and protocol. Deep packet inspection typically uses regular expression matching as a core operator. Regular expressions (RegExes) are used to flexibly represent complex string patterns in many applications ranging from network intrusion detection and prevention systems (NIDPSs). Regular expressions represent complex string pattern as attack signatures in DPI. It examine whether a packet's payload matches any of a set of predefined regular expressions. There are various techniques developed in DPI for deep packet inspection for regular expression. We survey on these techniques for further improvement in regular expression detection in this paper. We implement technique to block regexp packet such as DOS attack. In the result we found that it is possible to reduce RegExp transaction memory required in network intrusion detection. We implement this technique with possible use of DPI techniques in the wireless network**.**

Keywords— Deep Packet Inspection(DPI); Regular Expression(RegExp); Deterministic Finite Automata(DFA); LaFA; StriFA; CompactDFA; Tcam; DFA/EC; Snort; Br,Sql injection Attack, Xss Attack.

Introduction

In most of the applications Regular expressions (RegExes) are used to flexibly represent complex string patterns in many applications, such as network intrusion detection and prevention systems (NIDPSs), Compilers and DNA multiple sequence alignment [1]. Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are imminent threats of violation of computer security policies and standard security practices. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents in packet, logging information about these incidents, attempting to stop and reporting them to security administrators [3].

Deep Packet Inspection (DPI) is a technology that enables the network owner to analyze internet traffic, throughout the network, in real-time and to differentiate them according to their payload [3]. Traditional packet inspection algorithms have been limited to comparing packets to a set of strings. Newer DPI systems, such as Snort [11], and Bro [10], use rule-sets consisting of regular expressions, these systems are more expressive, efficient, and compact in specifying attack signatures [4].

In Network intrusion detection system techniques such as Bro [10] and Snort [11] and Linux Application Level Packet Classifier (L7 filter) [5] use RegExes to represent attack signatures or packet classifiers. Regular expressions represent complex string pattern as attack signatures in many applications. There is no current regular expression detection system is capable of supporting large RegEx set ; and even larger RegExp sets are expected in future with high speed demand [1]. The LaFA technology based on three observations, first RegExes consist of a variety of different components such as character classes or repetitions, Second is the order of components in a RegEx is preserved in the state machine detecting this RegEx and Third, most RegExes share similar components [1]. LaFA requires less amount of memory due to these three contributions: 1) providing specialized and optimized detection modules to increase resource utilization; 2) systematically reordering the RegEx detection sequence to reduce the number of concurrent operations; 3) sharing states among automata for different RegExes to reduce resource requirements.

The TCAM technology is the first hardware-based RE matching approach that uses ternary content addressable memory (TCAM), TCAM is has been widely deployed in modern networking devices for tasks such as packet classification. StriFA [6] technology presents the stride finite automata; it's a novel finite automata family, used to accelerate both string matching and regular expression matching. Compact DFA [6] proposed method is to compress DFAs by observing that the name used by common DFA encoding is meaningless. This degree of freedom and encode states in such a way that all transitions to a specific state are represented by a single prefix that defines a set of current states. Compact DFA technique applies to a large class of automata, which can be categorized by simple properties. With a TCAM [2] the throughput of compact DFA reaches to 10 Gb/s with low power consumption. This technique uses

Aho–Corasick (AC) algorithm, which uses a deterministic finite automaton (DFA) to represent the pattern set [6].

Extended Character set DFA [3] focused on reducing the memory storage requirement of DFAs, and it can be divided into the following categories: reducing the number of states, reducing the number of transitions, reducing the bits encoding the transitions, and reducing the character-set. Unfortunately, all of these approaches compress DFAs at the cost of increased main memory accesses. This technique propose a novel solution, called deterministic finite automata with extended character-set (DFA/EC), which can significantly decrease the number of states through doubling the size of the character-set. Solution describe in this methodology requires only a single main memory access for each byte in the traffic payload [3].

We implement this project to use deep packet inspection techniques for RegExp matching to improve IDS technique in wireless networks. Implementation and comparatively evolution of existing technique with the new propose technique by considering different parameter such as bandwidth requirement, speed of intrusion detection e.t.c. The implementation details describe in following section.

This project consists of improved RegEx detection technique which will have higher throughput than any other network intrusion detection techniques. We will minimize memory requirements and resource usage by network detection system. The project will be used for regular expression detection of attack signature pattern matching in wireless network. Modules of the projects are survey existing techniques, implement DOS attack filter, built RegEx detection technique which will work on wireless network.

Paper Organization

The rest of the paper organized as follows. An overview of DPI is given in Section 3. Here we describe Detail working of DPI and its levels. Section 4 describes the use of regular expression in DPI. Section 5 gives related DPI techniques limitations in RegExp detection. Ad-hoc network describes In section 6. Dos attacks and its types describes in section 7. Our proposed system details describe in section 8. Section 9 includes advantages of our proposed system. Result and comparison with existing system is given in section 10. Section 11 conclude our work and contains our future work.

Literature Survey and Elaboration & synthesis

Deep Packet Inspection (DPI) is a technology that enables the network owner to analyses internet traffic, through the network, in real-time and to differentiate them according to their payload. Originally the Internet protocols required the network routers to scan only the header of an Internet Protocol (IP) packet. The packet header contains the origin and destination address and other information relevant to moving the packet across the network. The “payload” or content of the packet, which contains the text, images, files or applications transmitted by the user, was not considered to be a concern of the network operator. DPI allows network operators to scan the payload of IP packets as well as the header. Figure 1 [8] shows the domain of packet inspection required in internet protocols and in DPI [8].

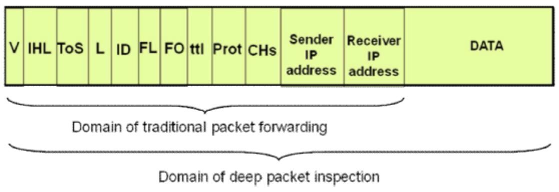

V
IHL
ToS
L
ID
FL
FO
ttl
Prot
CHs
Sender IP address
Receiver IP address
DATA
Domain of traditional packet forwarding
Domain of deep packet inspection

Fig 1. Domain of Deep Packet Inspection [8]

DPI systems use expressions to define patterns of interest in network data streams. The equipment is programmed to make decisions about how to handle the packet or a stream of packets based on the recognition of a regular expression or pattern in the payload. This allows networks to classify and control traffic on the basis of the content, applications, and subscribers [8]. Figure 2 shows the DPI implementation in software and hardware modules.

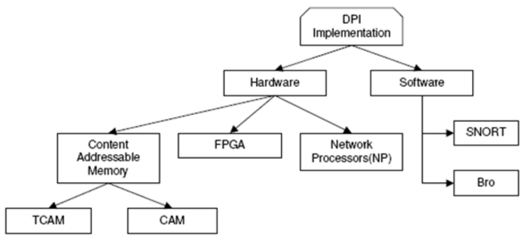


Fig 2. DPI Implementation [7]

Deep
packet inspection typically uses regular expression (RE) matching as a core operator. It examine whether a packet’s payload matches any of a set of predefined regular expressions. REs are fundamentally more expressive, efficient, and flexible in specifying attack signatures. Prior RE matching algorithms are either software base or field-programmable gate array (FPGA) based [1].

RegExes consist of a variety of different components such as character classes or repetitions [1]. Due to this variety, it is hard to identify a method that is efficient for concurrently detection of all these different components of a RegExp. Most RegExes share similar components. In the traditional FA, a small state machine is used to detect a component in a RegExp. This state machine is duplicated since the similar component may appear multiple times in different RegExes. Furthermore, most of the time, RegExes sharing this component cannot appear at the same time in the input. As a result, the repetition of the same state machine for different RegExes introduces redundancy and limits the scalability of the RegEx detection system. Figure 3 shows example illustrating the transformation from a RegEx set R into the corresponding LaFA technique [1].

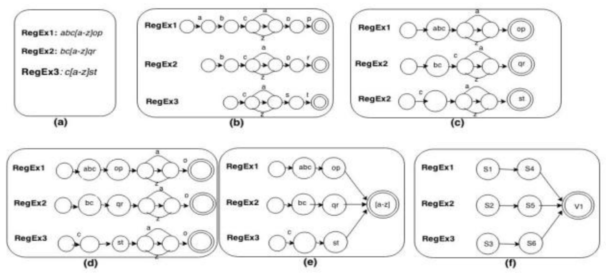


Fig 3: Example illustrating the transformation from a RegEx set R into the corresponding LaFA. (a) RegEx set. (b) NFA corresponding to. (c) Separation of simple strings. (d) Reordering of the detection sequence. (e) Sharing of complex detection modules. (f) LaFA representation of the RegExes [1].

Currently, regular expressions are replacing explicit string patterns as the pattern matching language of choice in packet scanning applications. Their widespread use is due to their expressive power and flexibility for describing useful patterns. For example, in the Linux Application Protocol Classifier (L7-filter), all protocol identifiers are expressed as regular expressions. Similarly, the Snort intrusion detection system has evolved from no regular expressions in its rule set in April 2003 to 1131 out of 4867 rules using regular expressions as of February 2006. Another intrusion detection system, Bro [10], also uses regular expressions as its pattern language [9].

A regular expression describes a set of strings without enumerating them explicitly. Table 1 lists the common features of regular expression patterns used in packet payload scanning. For example, consider a regular expression from the Linux L7-filter for detecting Yahoo traffic: "*^(ymsg|ypns|yhoo).?.?.?.?.?.?.?[lwt].*\xc0\x80*". This pattern matches any packet payload that starts with *ymsg*, *ypns*, or *yhoo*, followed by seven or fewer arbitrary characters, and then a letter *l*, *w* or *t*, and some arbitrary characters, and finally the ASCII letters *c0* and *80* in the hexadecimal form [9].

TABLE 1: Features of Regular Expressions [9]

Syntax	Meaning	Example
^	Pattern to be matched at the start of the input	^AB means the input starts with AB. A pattern without '^', e.g., AB, can be matched anywhere in the input.
I	OR relationship	*A*|*B* denotes *A* or *B*
.	A single character wildcard	
?	A quantifier denoting one or less	*A*? denotes *A*, or an empty sting.
*****	A quantifier denoting zero or more	*A** means an arbitrary number of *A*s.
{}	Repeat	*A*{100} denotes 100 *A*s.
[]	A class of characters	[*lwt*] denotes a letter *l*, *w*, or *t*.
[^]	Anything but not n	[^\n] denotes any character except \n.

.

The Deterministic Finite Automata (DFA) consists of a finite set of input symbols (which are denoted as P), a finite set of states, and a transition function to move from one state to the other denoted as @. In contrast of NFA, DFA has only one active state at any given time [4][9].

The regular expression is required as a need for packet payload inspection to different protocols packets. It introduces a limited DPI system to deal with all packets structures. As the result of this limitation, state-of-art systems have been introduced to replace the string sets of intrusion signature with more expressiveness regular expression (regexp) systems. Therefore, there are several content inspection engines which have partially or fully migrated to regexps including those in Snort [11], Bro [10], and Cisco systems'.

Experimentally, DFA of regexp that contains hundreds of pattern yields to tens of thousands of states which mean memory consumptions in hundreds of megabytes. As a solution of one of the common problems of HW based DPI solutions is the memory access because the memory accesses for the contents of the off chip memory are proportional with the number of bytes in the packet [9].

Related Work

LaFA [1]

LaFA is a novel detection method that resolves scalability issues of the current RegEx detection paradigm. LaFA is finite automata optimized for scalable RegEx detection. LaFA is used for representing a set of RegExes (R={r1;r2;r3;…}) which can also be called a *RegEx database*. An associated LaFA RegEx detection system can be queried with an input , such as a network packet. The system will return *a match* along with a list of matching RegExes if input includes one or more of the RegExes in . Otherwise, a *no* *match* is returned. LaFA facilitate the reduction of memory requirements and detection complexity.

LaFA architecture shown in following figure 3 consist of two blocks, The detection block and The correlation block Detection Block is responsible for detecting the component in the input string. The simple string optimized for exact string matching module used to detect simple string.

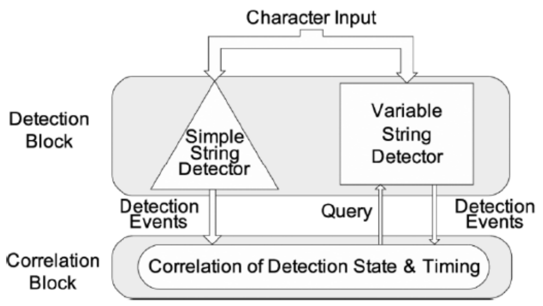


Fig 4: LaFA Architecture [1].

The variable string detector consists of highly optimized variable string detection modules. The correlation block used to track status of RegEx. The correlation block inspects the sequence of components in the input string (i.e., order and timing). The detectors in the detection block communicate their findings to the correlation block by sending detection events. Each detection event consists of a unique ID for the detected component and its location in the input packet.

Small TCAM is the first ternary content addressable memory (TCAM)-based RE matching solution. This technology use a TCAM and its associated SRAM to encode the transitions of the DFA built from an RE set where one TCAM entry might encode multiple DFA transitions. There are three key reasons why TCAM-based RE matching works well given in this research: a small TCAM is capable of encoding a large DFA with carefully designed algorithms. TCAMs facilitate high-speed RE matching because TCAMs are essentially high-performance parallel lookup systems TCAMs are off-the-shelf chips that are widely deployed in modern networking devices; it should be easy to design networking devices that include our TCAM-based RE matching solution[2].

In this experiment, TCAM able to store a DFA with 25 K states in a 0.5-Mb TCAM chip; most DFAs require at most one TCAM entry per DFA state. With variable striding it shows a throughput of up to 18.6 Gb/s is possible.

StriFA technique has been implemented in software and evaluated based on different traces. It undertakes the problem of designing a variable-stride pattern matching engine that can achieve an ultrahigh matching speed with a relatively low memory usage. The technology proposes stride finite automata (StriFA), which can process a variable number of characters at a time. StriFA is designed to be immune to the memory blow-up and byte alignment problems, and therefore, requires much less memory than the previous schemes. Stride deterministic finite automaton (StriDFA) and stride nondeterministic finite automaton (StriNFA) are two basic forms of implementation of StriFA.

The technology results showed that this architecture can achieve about 10-fold increase in speed, with a lower memory consumption compared to traditional NFA/DFA, while maintaining the same detection capabilities.

One of the main advantages of CompactDFA is that it fits into commercially available IP-lookup solutions. They may be used also for performing fast pattern matching. The output of the CompactDFA scheme is a set of compressed rules, such that there is only one rule per state. In the compressed set of rules, a code of a state may match multiple rules. Algorithm of CompactDFA and gives the intuition behind each of its three stages: State Grouping, Common Suffix Tree Construction, and State and Rule Encoding.

CompactDFA evaluates only for the pattern matching process. It uses two common pattern sets: Snort and ClamAV. This methodology can achieve fast pattern matching of 2 Gb/s with low power consumption. This technique shows a reduction of the pattern matching problem to the IP-lookup problem. Due to its small memory and power requirements, this architecture can implement with several TCAM working in parallel. Each TCAM performs pattern matching on a different session, achieving a total throughput of 10 Gb/s and beyond.

Implementation of this technique based on general-purpose processors that are cost-effective and flexible to update. It proposes a novel solution, called deterministic finite automata with extended character-set (DFA/EC), which can significantly decrease the number of states through doubling the size of the character-set. Solution describe in this methodology requires only a single main memory access for each byte in the traffic payload. It performs experiments with several Snort rule-sets. Results show that, compared to DFAs, DFA/ECs are very compact and are over four orders of magnitude smaller in the best cases.

The advantages of a DFA/EC are summarized in the following: A DFA/EC requires only one main memory access for each byte in the packet payload, while significantly reducing storage in terms of table size. A DFA/EC is conceptually simple, easy to implement, and easy to update due to fast construction speed. It maintains two states in runtime: one state for the main DFA, and an additional state for the complementary program. The current runtime state of the main DFA is represented by a DFA state label, and state of the complementary program is represented by a set that contains currently active complementary states. For each byte in the payload, the DFA/EC functions as follows. Firstly, the complementary program calculates the extra bit for the extended character by using the next byte and the current state of the complementary program and the next state of the main DFA and a label is looked-up by using the current state of the main DFA and the extended character, which is composed of the next byte and the extra bit. After that the complementary program calculates its next state by using its current state, the next byte, and the label on the main DFA transition In this implementation, the transition functions of the main DFA, , is implemented by a transition table; and is implemented by the complementary program, which only contains several efficient instructions.

The total minimum memory (storage) requirement of the transition tables in terms of bits and the number of bits is the product of the number of transitions and the number of bits needed to encode each transition measured. It gives result as; the memory bandwidth of DFA/EC can even be smaller than DFA in rule-sets exploit-19 and web-misc-28

A Distributed Denial of Service (DDoS) attack is an attempt to make an online service unavailable by overwhelming it with traffic from multiple sources. They target a wide variety of important resources, from banks to news websites, and present a major challenge to making sure people can publish and access important information.
Attackers build networks of infected computers, known as 'botnets', by spreading malicious software through emails, websites and social media. Once infected, these machines can be controlled remotely, without their owners' knowledge, and used like an army to launch an attack against any target. Some botnets are millions of machines strong. Botnets can generate huge floods of traffic to overwhelm a target. These floods can be generated in multiple ways, such as sending more connection requests than a server can handle, or having computers send the victim huge amounts of random data to use up the target’s bandwidth. Some attacks are so big they can max out a country's international cable capacity.

It is important to differentiate between Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. In a DoS attack, one computer and one internet connection is used to flood a server with packets, with the aim of overloading he targeted server’s bandwidth and resources.

DDoS attack, uses many devices and multiple Internet connections, often distributed globally into what is referred to as a botnet. A DDoS attack is, therefore, much harder to deflect, simply because there is no single attacker to defend from, as the targeted resource will be flooded with requests from many hundreds and thousands of multiple sources.

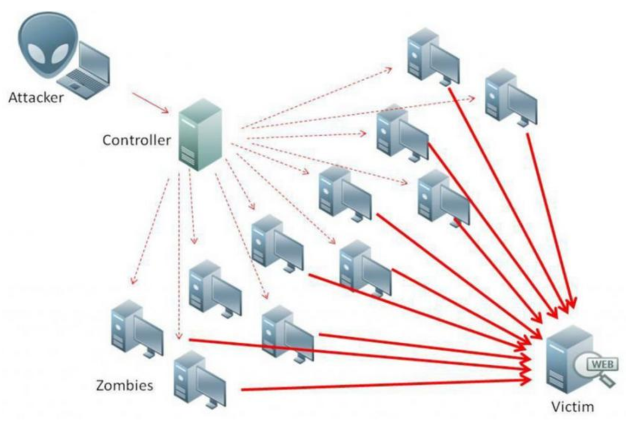


Fig 5: DDOS attack

XSS Attack: Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

XSS attacks may be conducted without using <script></script> tags. Other tags will do exactly the same thing, for example:

<body onload=alert('test1')>

or other attributes like: onmouseover, onerror.

Onmouseove

<b onmouseover=alert('Wufff!')>click me!</b>

Onerror

<img src="http://url.to.file.which/not.exist" onerror=alert(document.cookie);>

SQL Injection Attack: SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via web page input. Injected SQL commands can alter SQL statement and compromise the security of a web application.

Let's say that the original purpose of the code was to create an SQL statement to select a user with a given user id. If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId:
105 or 1=1

SELECT * FROM Users WHERE UserId = 105 or 1=1

The SQL above is valid. It will return all rows from the table Users, since **WHERE 1=1** is always true. The example above seems dangerous, What if the Users table contains names and passwords?

The SQL statement above is much the same as this:

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1

A smart hacker might get access to all the user names and passwords in a database by simply inserting 105 or 1=1 into the input box.

Problem with existing system

From the survey of above techniques we found that, the approach describe in these techniques may require a large number of transitions for some cases, leading to an increase in the number of memory accesses per input byte. In addition, DFA construction is complex and requires significant resources[1]. There is very few network intrusion detection techniques discover in wireless networks.

CompactDFA technique used in architecture requires several TCAM working in parallel, Due to its small memory and power requirements. NBA technologies have some significant limitations. They are delayed in detecting attacks because of their data sources, especially when they rely on flow data from routers and other network devices[3]. DFA/EC does not combine with the existing transition compression and character-set compression techniques, and perform experiments with more rule-sets[4]. One of the problems for StriFA is how to choose an appropriate tag. Since in both the rules and the incoming traffic, the occurrence probabilities of different characters vary from each other, it is a problem to choose an appropriate tag from the rule set [5].Following table shows comparison of existing network intrusion detection techniques.

TABLE 2: Comparison of deep packet intrusion techniques

Intrusion Detection Techniques	Throughput
LaFA	34 Gb/s
CompactDFA	10 Gb/s
Small TCAM	18.6 Gb/s
StriDFA	26.5 Gb/s

Following are objectives of problem we define on the basis of above survey:

1. We can improve network intrusion detection throughput with use of DPI techniques.

2. LaFA technique can be modifying for effective detection of evaluating RegExp on the network.

3. Higher throughput in network intrusion detection can be possible.

4. Above discuss techniques could have better performance in memory requirements, speed of detection, detection of evaluating RegEx detection.

5. There is very few intrusion detection techniques work on wireless network.

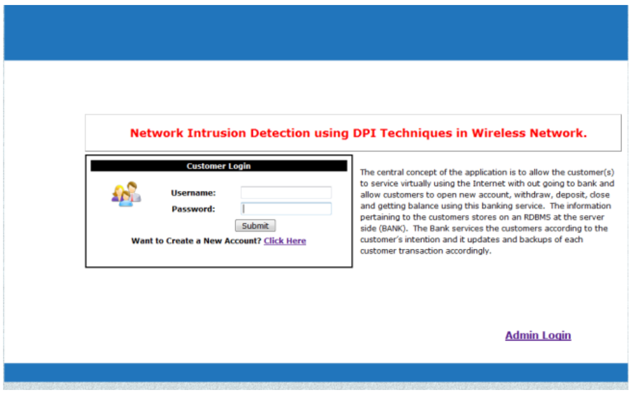
4. PROPOSED METHODOLOGY

System Implementation

This section we include the partitioning of project into different modules. All these modules are explained as follows:

This module includes the registration and log In of client and server. It includes following classes.

- Log In : With this class client can login to the server web site i.e. Banking website
- Register: this class used to register new user for server site.
- Admin Login: Admin can log in and manage the data of website.
- Admin View Log: With this class admin can view all the block IP and type of attacks done and its date and time.
- Unblock: With this class admin can unblock the block IP.


Network Intrusion Detection using DPI Techniques in Wireless Network.

Customer Login

Username:
Password:
Submit

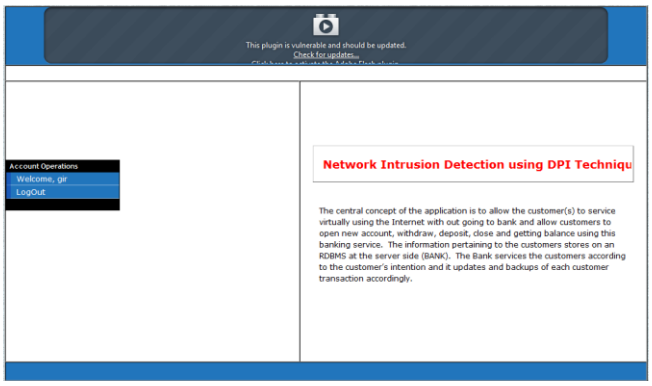
Want to Create a New Account? Click Here

The central concept of the application is to allow the customer(s) to service virtually using the Internet with out going to bank and allow customers to open new account, withdraw, deposit, close and getting balance using this banking service. The information pertaining to the customers stores on an RDBMS at the server side (BANK). The Bank services the customers according to the customer's intention and it updates and backups of each customer transaction accordingly.

Admin Login

Fig 6: User Log in snapshot

User Account Page Snapshot:

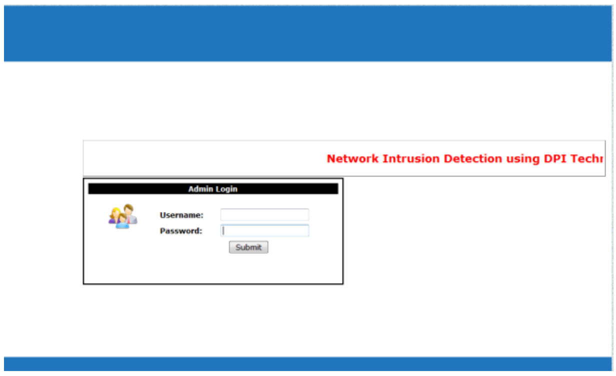


Account Operations
Welcome, gir
LogOut

Network Intrusion Detection using DPI Techniqu

The central concept of the application is to allow the customer(s) to service virtually using the Internet with out going to bank and allow customers to open new account, withdraw, deposit, close and getting balance using this banking service. The information pertaining to the customers stores on an RDBMS at the server side (BANK). The Bank services the customers according to the customer's intention and it updates and backups of each customer transaction accordingly.

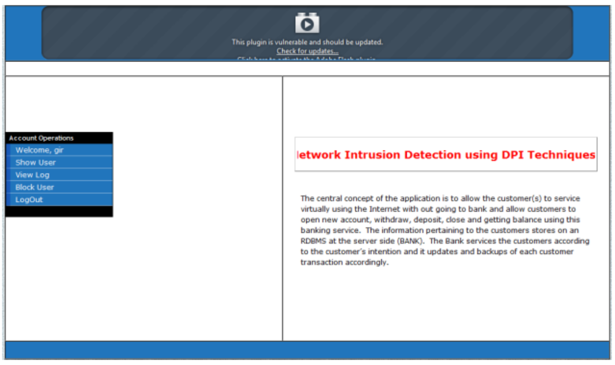
Fig 7: User Account Snapshot


Network Intrusion Detection using DPI Techn

Admin Login

Username:
Password:
Submit

Fig 8: Snapshot for admin log in page



Account Operations
Welcome, gir
Show User
View Log
Block User
LogOut

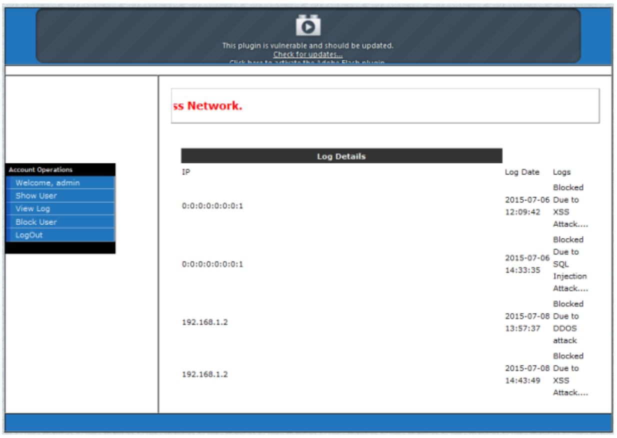
letwork Intrusion Detection using DPI Techniques

The central concept of the application is to allow the customer(s) to service virtually using the Internet with out going to bank and allow customers to open new account, withdraw, deposit, close and getting balance using this banking service. The information pertaining to the customers stores on an RDBMS at the server side (BANK). The Bank services the customers according to the customer's intention and it updates and backups of each customer transaction accordingly.

Fig 9: Snapshot for admin log in page.

DDOS Attack Filter

- Request Count: This class count no. of requests per second (RPS) coming from client.
  - RPS Limit: This class used to save the limit of the request per second any human user can make.
  - Block IP: DDOS filter block the user IP if RPS exceeds the limit.



Account Operations
Welcome, admin
Show User
View Log
Block User
LogOut

ss Network.

Log Details

IP	Log Date	Logs
0:0:0:0:0:0:0:1	2015-07-06 12:09:42	Blocked Due to XSS Attack....
0:0:0:0:0:0:0:1	2015-07-06 14:33:35	Blocked Due to SQL Injection Attack....
192.168.1.2	2015-07-08 13:57:37	Blocked Due to DDOS attack
192.168.1.2	2015-07-08 14:43:49	Blocked Due to XSS Attack....

Fig 10: Snapshot of attack log detail page.

Block IP Page Snapshot:

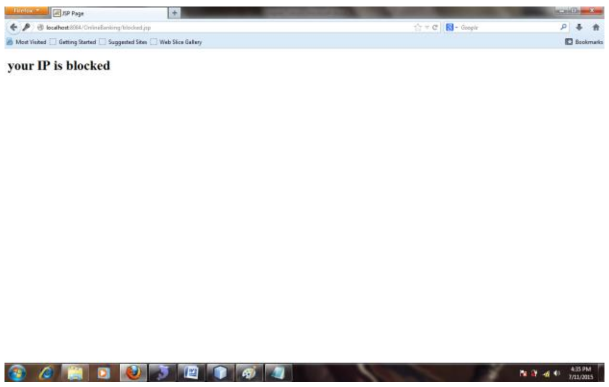
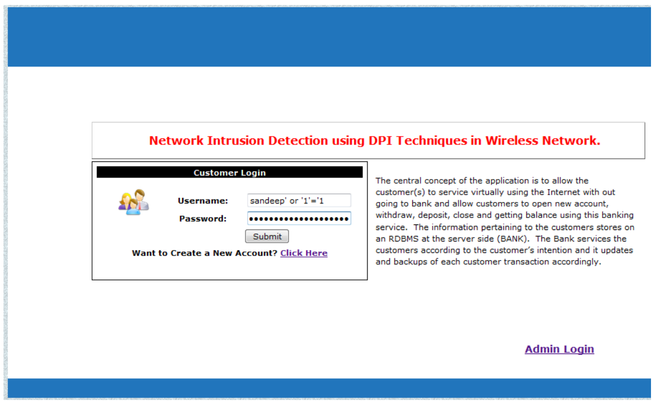

your IP is blocked

Fig 11: Snapshot of Block IP Page

- Pattern Matching: This class used deep packet inspection pattern matching algorithm to detect the attack pattern.
- Block IP: If request packet pattern match to attack pattern saved in database then it block the client IP.


Network Intrusion Detection using DPI Techniques in Wireless Network.

Customer Login

Username: sandeep' or '1'='1

Password: ••••••••••••••••••••

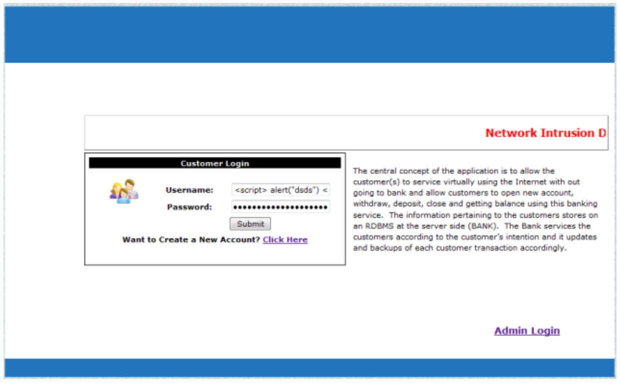
Submit

Want to Create a New Account? Click Here

The central concept of the application is to allow the customer(s) to service virtually using the Internet with out going to bank and allow customers to open new account, withdraw, deposit, close and getting balance using this banking service. The information pertaining to the customers stores on an RDBMS at the server side (BANK). The Bank services the customers according to the customer's intention and it updates and backups of each customer transaction accordingly.

Admin Login

Fig 12: Snapshot of SQL Injection Attack


Network Intrusion D

Customer Login

Username: <script> alert("dsds") <

Password: ••••••••••••••••••••

Submit

Want to Create a New Account? Click Here

The central concept of the application is to allow the customer(s) to service virtually using the Internet with out going to bank and allow customers to open new account, withdraw, deposit, close and getting balance using this banking service. The information pertaining to the customers stores on an RDBMS at the server side (BANK). The Bank services the customers according to the customer's intention and it updates and backups of each customer transaction accordingly.

Admin Login

Fig 13 : Snapshot of XSS attack

- In this project we use Apache Tomcat sever to use database named IDS.
- This database used to save attack pattern and request count per second and access by DDOS filter and DPI module to search for the attack pattern.
- Database used to store account data, user account id and password, list of blocked ip, type and date and time of attack when it occurs

Low-level designs of software system include:

- private classes, private methods, private attributes
- Algorithms.

Low-level design also provides an interface for all classes, public and private methods, including parameters, return values, exceptions thrown and types defined. It describes and justifies the choice of data structures, describes the major alternatives that are considered and why the choice is preferred that is opted.

In our project private classes such as block ip class, account information class only can be access by owner. Owner can login to the server using admin login and see the all information from the site. Admin also have privileges to alter the data saved in database such as unblock the user ip. Client can only access the information about his account, hr does not see any attack information or other user account data. If client try to make attack on the server it can block by dos filter and DPI module.

DOS filter compare the request count per second with the possible no. of the request any human user can make which is saved in database and make the decision to block the ip or not. DPI used to compare packet pattern with the attack pattern saved in database if attack pattern match then it blocks the user op who try to attack on server.

We implement our project on Net beans IDE 7.3. Project is implemented in java language. We used SQL yog enterprises to use Apache Tomcat server. Client server model is used to run this project. To connect server and client we implement Ah-hoc wireless connection between them. We also test our project on internet by connecting internet connection. Data owner & authorized user can login through any machine.

DDOS Attack.java File

```
        package ddosattack;
        import java.io.IOException;
        import java.net.*;

        public class attack {

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

            String str="localhost:8084/OnlineBanking";

            for (int i = 0; i < 200; i++) {
                pingUrl(str);
            }
          }
          public static boolean pingUrl(final String address)
throws IOException {
         try {
          final URL url = new URL("http://" + address);
          final HttpURLConnection urlConn =
(HttpURLConnection) url.openConnection();
```

```
            urlConn.setConnectTimeout(1000 * 10); //
mTimeout is in seconds
            final long startTime = System.currentTimeMillis();
            urlConn.connect();
            final long endTime = System.currentTimeMillis();
            if (urlConn.getResponseCode() ==
HttpURLConnection.HTTP_OK) {
             System.out.println("Time (ms) : " + (endTime -
startTime));
             System.out.println("Ping to "+address +" was
success");
             return true;
             }
            } catch (final MalformedURLException e1) {
             e1.printStackTrace();
            } catch (final IOException e) {
             e.printStackTrace();
            }
            return false;
           }
           }
```

SQLInjectionRegExUtil.java

```
package com.sqlinjection.regularexpressions;

import java.util.ArrayList;
//class for SQL injection checking
public class SQLInjectionRegExUtil
{
 public String
SQLInjectionRegEx="/\\w*((\\%27)|(\\'))((\\%6F)|o|(\\%4F))((
\\%72)|r|(\\%52))exec(\\s|\\+)+(s|x)p\\w+/ix";
   public ArrayList<String> sqlKeywordsFound=new
ArrayList<String>();

   //check for SQL characters in query
   public  boolean checkSQLChars(String query)
   {
      //array for SQL keywords/charcters used in SQL injection
      String[] AttackPattern = {"'","TABLE","table"
,"CREATE", "create", "ALTER", "alter", "shutdown",
"DROP", "drop", "RENAME", "rename", "SELECT", "
select", " INSERT", "insert", " UPDATE", " update", "
DELETE", " delete", " GRANT", " grant", " REVOKE", "
revoke", " char", " int", "@@version", "@@VERSION",
"exec", "update", "select", "EXEC", "union", "UNION",
"WAITFOR", "waitfor", "ORDER BY", "order by", ";", "\"",
"\'", "/*", "*/", "--"};
      int i = 0, fnd;
      boolean chk = false;

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

         fnd = query.indexOf(AttackPattern[i]);//checking for
attack pattern
         if (fnd != -1)
         {
           sqlKeywordsFound.add(AttackPattern[i]+"_");

           System.out.println("ap "+AttackPattern[i]);
            System.out.println("q "+query);
           chk = true; //Attack Pattern Present
           break;
         }
        return chk;
   }
   public static void show()
   {
     String[] AttackPattern = {"'", " CREATE", " create", "
ALTER", " alter", "shutdown", " DROP", " drop", "
RENAME", " rename", " SELECT", " select", " INSERT",
"insert", " UPDATE", " update", " DELETE", " delete", "
GRANT", " grant", " REVOKE", " revoke", " char", " int",
"@@version", "@@VERSION", "exec", "update", "select", "
EXEC", " union", " UNION", " WAITFOR", " waitfor", "
ORDER BY", " order by", ";", "\"", "\'", "/*", "*/", "--"};

      System.out.println("Keyword and characters for SQL
injection");
      for (int i = 0; i < AttackPattern.length; i++) {
         String string = AttackPattern[i];
         System.out.println(""+string);
      }
     }
      public static void main(String[] args) {
      show();
}
}
```

XSSAttackRegExUtil.java:

```
package co.xss.regularexpressionutil;
import java.util.regex.Pattern;
import javax.print.DocFlavor;

public class XSSRegExUtil {

   public String RegExForXSS="/((\\%3C)|<)((\\%2F)|\V)*[a-
z0-9\\%]+((\\%3E)|>)/ix";

   private String cleanXSS(String value)
   {
       value = value.replaceAll("<", "& lt;").replaceAll(">", "&
gt;");
       value = value.replaceAll("\\(", "& #40;").replaceAll("\\)",
"& #41;");
       value = value.replaceAll("'", "& #39;");
       value = value.replaceAll("eval\\((.*)\\)", "");
       value                                                    =
value.replaceAll("[\\\"\\\'][\\s]*javascript:(.*)[\\\"\\\']", "\"\"");
       value = value.replaceAll("script", "");
       return value;
    }

    private static Pattern[] patterns = new Pattern[]{
       // Script fragments
       Pattern.compile("<script>(.*?)</script>",
Pattern.CASE_INSENSITIVE),
       // src='...'
       Pattern.compile("src[\r\n]*=[\r\n]*\\\'(.*?)\\\'",
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL),
       Pattern.compile("src[\r\n]*=[\r\n]*\\\"(.*?)\\\"",
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL),
       // lonely script tags
       Pattern.compile("</script>",
Pattern.CASE_INSENSITIVE),
       Pattern.compile("<script(.*?)>",
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL),
       // eval(...)
       Pattern.compile("eval\\((.*?)\\)",
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL),
       // expression(...)
       Pattern.compile("expression\\((.*?)\\)",
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL),
       // javascript:...
       Pattern.compile("javascript:",
Pattern.CASE_INSENSITIVE),
       // vbscript:...
       Pattern.compile("vbscript:",
Pattern.CASE_INSENSITIVE),
       // onload(...)=...
       Pattern.compile("onload(.*?)=",
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL)
    };

    public XSSBean stripXSS(String value)
    {
        String org=value;
       if (value != null) {
          // NOTE: It's highly recommended to use the ESAPI
library and uncomment the following line to
          // avoid encoded attacks.
          // value = ESAPI.encoder().canonicalize(value);

          // Avoid null characters
          value = value.replaceAll("\0", "");

             for (Pattern scriptPattern : patterns){
             value = scriptPattern.matcher(value).replaceAll("");
          }
       }
       boolean flag=false;
       if(!org.equals(value))
       {
       flag=true;
       }
       XSSBean xb=new XSSBean();
       xb.setIsXSS(flag);
       xb.setValue(value);
       return xb;
    }
     public static void main(String[] args) {

       XSSBean s=new XSSRegExUtil().stripXSS("<script");
        System.out.println("v "+s.isIsXSS());
         System.out.println("v "+s.getValue());
    }
}
```

ddosattackdetection.java:

```
package com.DDOS.jpcap;
import com.DDOS.dao.DdosDetectDao;
import com.DDOS.dao.SourceCheckandCountDao;
import java.awt.Toolkit;
import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;
public class ddosattackdetection
{
    Toolkit toolkit;
    Timer timer;
    DdosDetectDao dddao=new DdosDetectDao();
    SourceCheckandCountDao                     sccdao=new
SourceCheckandCountDao();
```

```
   public boolean checkSuspectedIPWithnum_periodAndTH()
   {
      boolean flag=false;
      ArrayList alist=null;
      int num_period=dddao.getnum_periodThreshold();
      int th=dddao.getThThreshold();
    //  int max_th=dddao.getMax_ThThreshold();
     // int node_th=dddao.getNode_ThThreshold();
      //System.out.println("Threshold :"+threshold);
      alist=dddao.getsuspectedIPAddress(num_period,th);
      String ip="";
      int alen=alist.size();
      for(int i=0; i<alen;i++)
      {
         ip=alist.get(i).toString();
         sccdao.updateSourceIPStatus(ip, 1);

         System.out.println("IP "+ip+" Added to Source black
list ");
      }
      return flag;
   }

   public boolean checkSuspectedIPWithMax_Th()
   {
      boolean flag=false;
      ArrayList alist=null;
      //int num_period=dddao.getnum_periodThreshold();
     // int th=dddao.getThThreshold();
      int max_th=dddao.getMax_ThThreshold();
     // int node_th=dddao.getNode_ThThreshold();
      //System.out.println("Threshold :"+threshold);

alist=dddao.getsuspectedIPAddressWithMax_Threshold(max_
th);
      String ip="";
      int alen=alist.size();
      for(int i=0; i<alen;i++)
      {
         ip=alist.get(i).toString();
         sccdao.updateSourceIPStatus(ip, 1);

         System.out.println("IP "+ip+" Added to Source black
list ");
      }
      return flag
  }
   public void AttackDetectionModule()
   {
          toolkit = Toolkit.getDefaultToolkit();
      timer = new Timer();
      timer.schedule(new ddosattackdetection.RemindTask(),
                              0,       //initial delay
                              60*1000);   //(100*1000)subsequent
rate
    }
   class RemindTask extends TimerTask
   {

      public void run()
      {
            toolkit.beep();
         checkSuspectedIPWithnum_periodAndTH();

         sccdao.TruncatePacketCount();
         //sccdao.ResetPacketCount();
      }
   }
}
```

Mathematical Model for simple regular expression matching

Consider

R=regular expression

R= abc[a-z]op

P=pattern with which we match the input packet regular expression.

If

R=P (Regular expression pattern match)

i.e R(i)=P(i)

R(ii)=P(ii)

R(iii)=P(iii) . . . . .

Then the packet blocked

Else the packet forward to the server.

In our approach we divide regular expression in two division

S= set if simple variables in regular expression.

C= set of complex variables in regular expression.

Consider

R= adc[a-z]op

We divide this regular expression

S= abcop

C= [a-z]

If

S=P (Pattern Matching)

C=P (Pattern Matching)

Then the packet is block

Otherwise packet forwarded to the server.

Figure 2.7 shows RegEx components based on their detection complexity and depth.

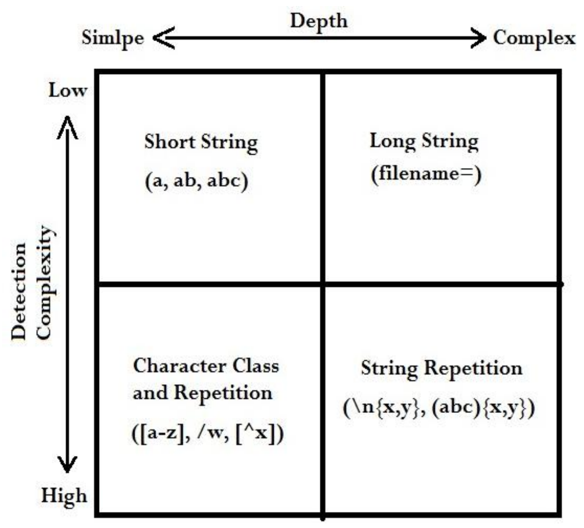


Fig 14 : RegEx components based on their detection complexity and depth

In DDos filter we count number of request per 40 seconds from one IP address, if the no requests are greater than 100 then we block the IP address. Thus any human can not send hundred requests per 40 seconds

Suppose

r = request count per 40 seconds (time stamp) from particular IP.

t= time stamp which we set 40 seconds.

c = no of request any human can possibly send to server here we set (100).

If

r > c in time t

Ip address blok, (No further request from this IP processed)

If not

Then allow IP to communicate with server.

Architecture Diagram

Figure 4.2.1 shows architecture diagram for our project. It shows client and server transaction which done through DOS filter and DPI module to avoid attack on server such as DOS attach and Sql Injection Attack.

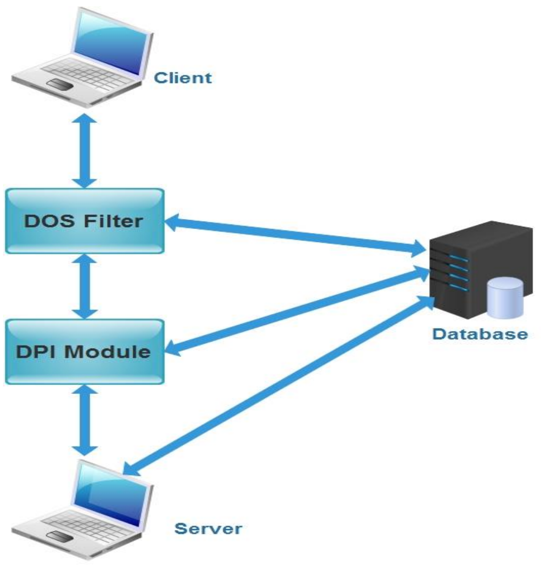


Fig 15: Architecture Diagram

ER Diagram means Entity Relationship Diagram. The Entities are mapped to the tables in the application. An entity-relationship (ER) diagram is a UML diagram that shows relationships between entities in a database. In figure 4.2.2 client, DOS filter, DPI module, server and data base are entities. Connection line shows the relationship between them, and ovals represent attributes of entities.

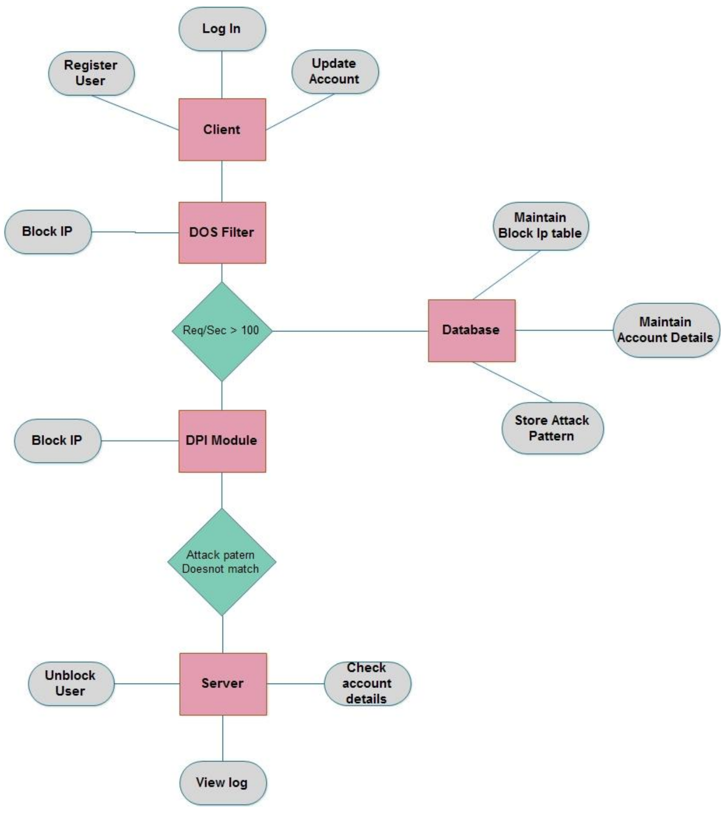


Fig 16: E-R diagram for our project

Activity diagram shows the execution and flow of the system, and not how it is assembled. Activity diagrams contain activities, made up of smaller actions. When used for software modeling, activities typically represent an event occur after any function call. Figure 4.3.a shows activity diagram for our project. Figure 4.3.b shows activity diagram for DOS filter and DPI module.

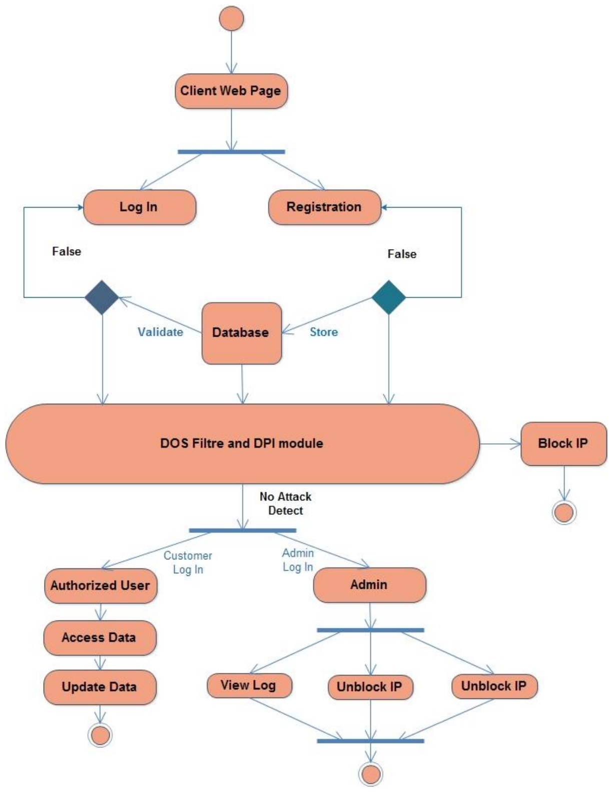


Fig 17: Activity Diagram for our Project

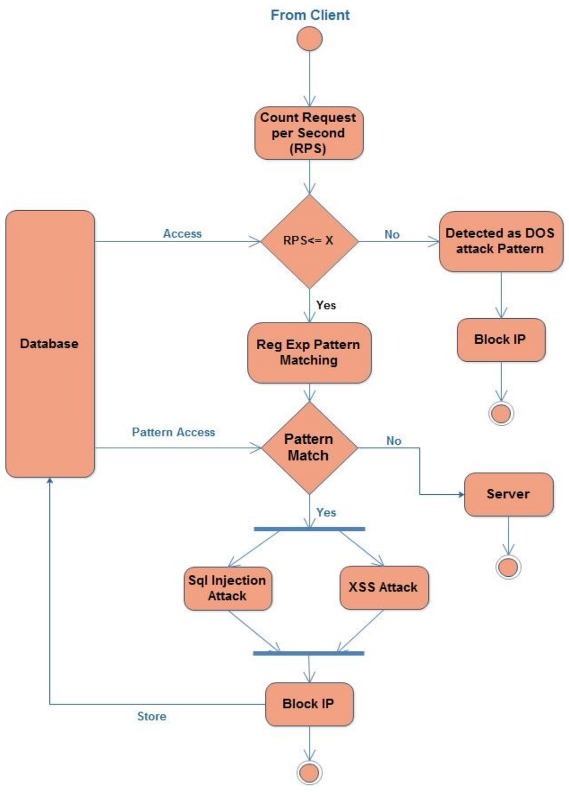


Fig 18: DOS and DPI Module Activity Diagram

The system is "decomposed in lower-level DFD (Level 1)" into a set of processes, data stores, and the data flows between these processes and storage of data.

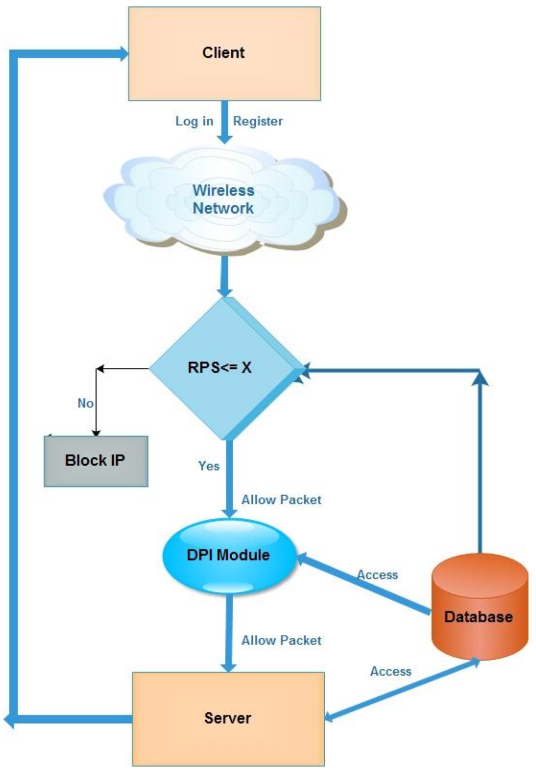


Fig 19: DFD level 1 Diagram (DOS filter)

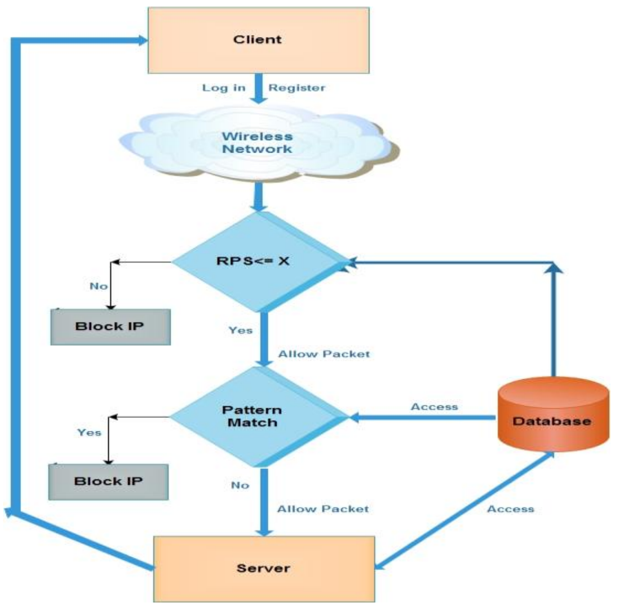


Fig 20: DFD level 2 Diagram (Dpi Module)

Test Procedures Results and Discussion

TABLE 3: Test Plan for Project

Sr. No	Content	Explanation
1.	Name of the product	Network Intrusion Detection Using Dpi Techniques In Wireless Network.
2.	Prepared By	Mr. Satish N. Gujar JJT,University, Rajastan, India,
3.	Introduction	Testing is carried out mainly for finding all defects present in the system and to prevent a defective product reaching the customers. Testing is also carried out to convince the customer that product is fulfilling the specifications and functional requirement of customer.
4.	Objectives	To test all the functionality of a module. Test correctness of input and output. Test the component point of view estimation accuracy.
5.	Scope	Test the functionality for traditional development and Component based software development.
6.	Testing Strategy	Unit Testing Module Testing Performance Testing User acceptance Testing Beta Testing.
7.	Approval	Dr. Vilas M. Thakare, S.G.B. Amravati University, Amravati Maharashtra, India

Advantages of proposed system

With this proposed methodology improve network intrusion detection throughput with use of DPI techniques. The Dos filter remove or blocked all dos attack malicious packet it filters the dos attack packets due to which it improve malicious packet detection.

- Speed of intrusion detection and prevention is increased.
- Authorized user can access data without any security risk.
- Memory requirement for server security is reduced.

Result and Analysis

This section we discuss on the result of our project. After all the test cases we test our program for various DDOS attack on our server site web page. It results in machine IP block as we expected. We test our project for Sql injection and XSS attack for testing DPI technique pattern matching by inserting various Sql injection and XSS attack, our project blocked every attack. We also tested admin privileges to unlock any client which would be block earlier. Admin can unblock any client from the database by accessing admin login.

Admin can also see the information about attack that what kind of attack occurs at which date and time, Server can store the list of attack according to its time and date.

Conclusion and Future Enhancement

In existing systems, there are some limitations Intrusion detection and prevention system. In our project DDOS filter blocks the DDOS attacks so that the intrusion detection module only face packet without DDOS attack pattern, hence it automatically increase intrusion detection and prevention speed.

In DDOS filter we implement technique to count request per second and set the value that cannot be achieve by any human user, If the request per second count is greater than predefined value system block clients ip, so that the attacker cannot try to attack on server again. Intrusion detection module use Deep Packet Inspection pattern matching technique to detection of intrusion containing packets and prevent it.

We implement our project with the use of wireless network; we tested it with ad-hoc wireless network and internet as client server module.

Future Enhancement

In future we can improve intrusion detection by using various emerging attack patterns in intrusion detection module.

References

[1] Masanori Bando, N. Sertac Artan, and H. Jonathan Chao., "*Scalable Lookahead Regular Expression Detection System for Deep Packet Inspection*", IEEE Transactions on Networking, Vol. 20, No. 3, June 2012.

[2] Chad R. Meiners, Jignesh Patel, Eric Norige, Alex X. Liu, and Eric Torng., "*Fast Regular Expression Matching Using Small TCAM*", IEEE/Acm Transactions On Networking, Vol. 22, No. 1, February 2014.

[3] Tiwari Nitin, Solanki Rajdeep Singh and Pandya Gajaraj Singh, "*Intrusion Detection and Prevention System (IDPS) Technology- Network Behavior Analysis System (NBAS)*", ISCA Journal of Engineering Sciences, Vol. 1(1), 51-56, July 2012.

[4] Cong Liu, Yan Pan, Ai Chen, and Jie Wu., "*A DFA with Extended Character-Set for Fast Deep Packet Inspection*", IEEE Transactions On Computers, Vol. 63, No. 8, August 2014.

[5] Xiaofei Wang, Yang Xu, Junchen Jiang, Olga Ormond, Bin Liu, and Xiaojun Wang, "*StriFA: Stride Finite Automata for High-Speed Regular Expression Matching in Network Intrusion Detection Systems*", IEEE Systems Journal, Vol. 7, No. 3, September 2013.

[6] Anat Bremler-Barr, DavidHay, and Yaron Koral, "CompactDFA: Scalable Pattern Matching Using Longest Prefix Match Solutions", IEEE/Acm Transactions On Networking, Vol. 22, No. 2, April 2014.

[7] Tamer AbuHmed, Abedelaziz Mohaisen, and DaeHun Nyang., "*A Survey on Deep Packet Inspection for Intrusion Detection Systems*", Information Security Research Laboratory, Inha University, Incheon 402-751, Korea, March 2008.

[8] Klaus Mochalski, and Hendrik Schulze,"*White paper on Deep Packet Inspection*", ITU-T study groups com13.

[9] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz, "*Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection*", ACM 580-0/06/0012, December 3–5, 2006.

[10] Bing Chen, Lee, J., and Wu, A.S., "*Active event correlation in Bro IDS to detect multi-stage attacks*", Fourth IEEE International Workshop on Information Assurance, 13-14 April 2006.

[11] Rafeeq Ur Rehman, "*Intrusion Detection Systems with Snor*t", ISBN 0-13-140733-3, Library of Congress Cataloging-in-Publication Data, Prentice Hall PTR Upper Saddle River, New Jersey 07458.