JAVA FRAMEWORK FOR SIGNATURE BASED NETWORK INTRUSION DETECTION SYSTEM

Ms. Babita Saharia[1], Prof. Bhaskar P. C [2]

[1]*Student, Department of Technology, Shivaji University, Kolhapur, (India)*
[2] *Departments of Technology, Shivaji University, Kolhapur, (India)*

ABSTRACT

An intrusion detection system (IDS) inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system. This paper presents the full implementation of the intrusion detection system that captures network data as well as provides sufficient means for the decision making process of an administrator. This paper also describes structured IDS model using snort rule base to show acceptable and abusive behaviour, observe and respond to protected systems. The main focus is on building and implementing a signature based suitable intrusion detection model for local network that can monitor computer systems for effective detection of intrusion(attacks from outside the organization) or misuses(attacks from within the organization) over the network.

Keywords: Signature Based, Intrusion Detection System, Snort Rule Based

I. INTRODUCTION

The Internet has become a global platform for communication, commerce and individual expression. With this growth come ever-greater risks as well as opportunities for attackers to illicitly access computers over network. These attacks are generally not foreseen. The danger of computer malware is becoming so serious that every organization is spending a significant amount to protect the computer resources. The conventional security system like firewall is not enough. Intrusion detection systems provide a level of protection beyond the firewall by protecting the network from internal and external security attacks and threats. Intrusion detection systems (IDSs) collect information from a computer or a computer network in order to detect attacks and misuses of the system. Intrusion detection is a security system that monitors computer systems and network traffic and analyzes that traffic in order to detect unwanted activity and events such as illegal and malicious traffic, traffic that violates security policy [1, 2]. Intrusion detection functions include:

- Monitoring and analyzing both user and system activities
- Analyzing system configurations and vulnerabilities
- Assessing system and file integrity
- Ability to recognize patterns typical of attacks
- Analysis of abnormal activity patterns
- Tracking user policy violations

1.1 Types of Intrusion Detection System

There are broadly two types of intrusion detection system. These are host based Intrusion Detection System and network based Intrusion Detection System [9].

- *Host-based IDS*, evaluate information found on a single or multiple host systems, including contents of operating systems, system and application files.
- *Network-based IDS*, evaluate information captured from network communications, analyzing the stream of packets travelling across the network. Packets are captured through a set of sensors.
- Depending on design available for detecting attacks the intrusion detection can broadly categories in two ways : Signature based and anomaly based[6,7]. These two methods share many characteristics, yet are complementary in that they each have strengths where the other has weaknesses.
- *Signature based detection model :* IDS detect intrusions by looking for activity that corresponds to known signatures of intrusions or vulnerabilities. In signature based detection, the observed events are compared against the pre-defined signatures in order to identify possible unwanted traffic
- *Anomaly detection model :* IDS detect intrusions by searching abnormal network traffic. The anomaly-based detection refers to the problem of finding patterns in data that do not conform to expected behaviour.

1.2 System Architectural View

The intrusion detection system discussed in this paper is implemented in java uses WinPcap and Jpcap packages. The live network packets or the packets generated by the traffic generator are captured with the help of WinPcap and Jpcap as shown in figure 1. The packets can be processed directly or it can be saved in a file for further processing.

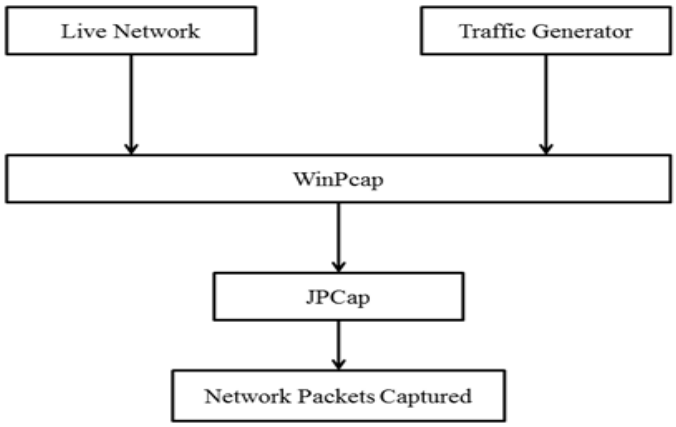

Figure 1: Architecture for Network Monitoring System

WinPcap is an open source library for packet capture and network analysis for the Win32 platforms [8]. Most networking applications access the network through widely used operating system primitives such as sockets. It is easy to access data on the network with this approach since the operating system copes with the low level details (protocol handling, packet reassembly, etc.) and provides a familiar interface that is similar to the one used to read and write files.

Jpcap is a Java class package which enables to capture and send IP packets from Java application. This package uses WinPcap (packet capture library) [4] and Raw Socket API. This is an open source library for capturing and sending network packets from Java applications. It provides facilities to:

- Capture raw packets live from the wire.
- Save captured packets to an offline file, and read captured packets from an offline file
- Filter the packets according to user-specified rules before dispatching them to the application.
- Send raw packets to the network Jpcap is based on libpcap/winpcap, and is implemented in C and Java.
- This supports Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMP packets.

II. PROPOSED SYSTEM

Using Java, the proposed network intrusion detection system (IDS) which is capable of monitoring traffic to or from a single host on the network (figure 2). The IDS process each packet in the trace, and as rules are matched print an alert (which should include the name of the matched rule) to standard out. One packet may figure into any number of alerts, so your IDS should not stop processing a packet or stream when a single rule is matched. The proposed system consists of following components:

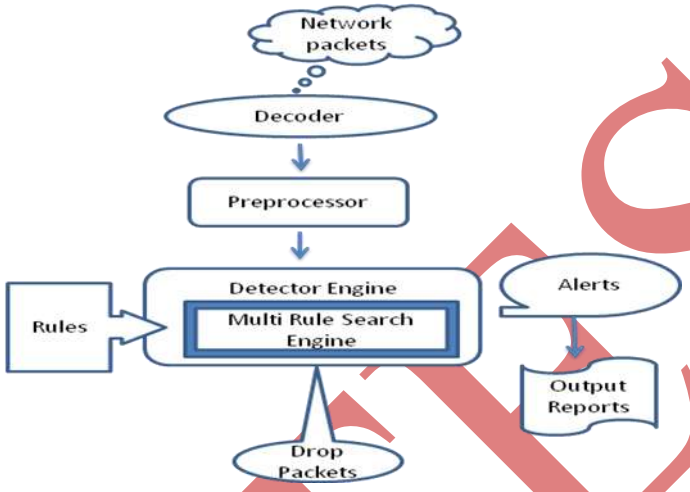

Figure 2: Proposed System Diagram for IDS

2.1 Decoder

The decoder receives the packets from the winpcap packet capturing library and processes them. Formal checker evaluates the packet structure for truncated packet headers and proper checksum, depending on whether it is an Ethernet, ARP, IP, TCP, UDP or ICMP packets.

2.2 Pre-processor

This module takes the packets from the decoder and performs the functions like IP de-fragmentation, building the sessions for reassembly of packets etc.

2.3 Detection Engine

This module captures packets from the network using a packet sniffer, which scans the packet and creates the rule base of the information it contains the following unit.

2.4 Multi Rule Search Engine

This is a rule-based programming environment, a rule engine and scripting environment. It has a fast and efficient algorithm. The detection engine analyzes the packet against snort ***Rules*** (discussed in next section) contained in various files. Intrusion alerts are reported by IDS sensors placed in a network, and they typically have attributes like the type of events, the address of the source and destination host, the time stamp and so on [figure 5 & 6].

2.5 Alerts

Sends the alerts triggered by the Detection Engine to Alert Console in real time.

In addition, dropping packets that match signatures of known attacks or undesirable traffic is a preventive action. In the figure 2 the first phase is the packet capturing mechanism. After packets have been captured in a raw form, they are passed into the packet decoder. The packet decoder translates specific protocol elements into

an internal data structure. After the initial preparatory packet capture and decode is completed, traffic is handled by the preprocessors. Any numbers of pluggable preprocessors either examines or manipulate packets before handing them to the next component: the detection engine. The detection engine performs simple tests on a single aspect of each packet to detect intrusions. The last component is the output plugins, which generate alerts to present suspicious activity A typical response to a detected network attack is to alter the environment of the system under attack. The response mechanisms are intended to allow system administrators to take an active role within their authority to minimize damage caused by a detected attack.

III. SNORT

Snort is an open source network intrusion detection system (NIDS) created by Martin Roesch[3,5]. Snort is a packet sniffer that monitors network traffic in real time, scrutinizing each packet closely to detect a dangerous payload or suspicious anomalies. Snort is based on *libpcap* (for library packet capture), a tool that is widely used in TCP/IP traffic sniffers and analyzers. Snort comes with many signatures enabled in default configuration. A signature takes the form of a specialised program, with raw events as input. Any input triggering a filtering program, or input that matches internal alert conditions, is recognised as an attack. To illustrate, consider the examples (taken from [Roesch]) in Figure 3 and Figure 5. In the first case, this is done by matching any connection with TCP destination port 80, and a specific string in the HTTP segment body. In the second case, the filter checks that the connection destination is one of the nominated destination servers, creates an HTTP data segment by concatenating the current and previous TCP segments, and searches each line for occurrences from the signature set. If any matches are found, a log entry is created with the current time, connection and request details.

```
alert tcp any any -> 192.168.1.10/32 80 (msg: "TTL=100"; \ttl: 100;)
alert icmp any any -> any any (msg: "ICMP Packet found";)
alert ip any any -> any any (msg: "IP Packet detected";)
```

Figure 3: Some Example of Snort Rules

Snort rules [3] are not only simple to write but are capable enough to detect a wide range of suspicious activities in the network. A snort rule can be broken down into two basic parts as in figure 4:

- The rule header
- The rule option

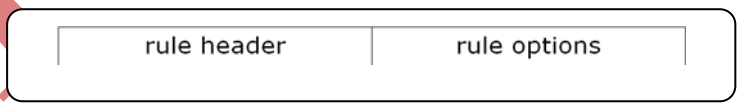

Figure 4: Snort Rule Structure

<rule action > <protocol> <source ip> <source port> <direction> <dest. ip> <dest. port> <rule options>

Figure 5: Snort Rule Header Structure

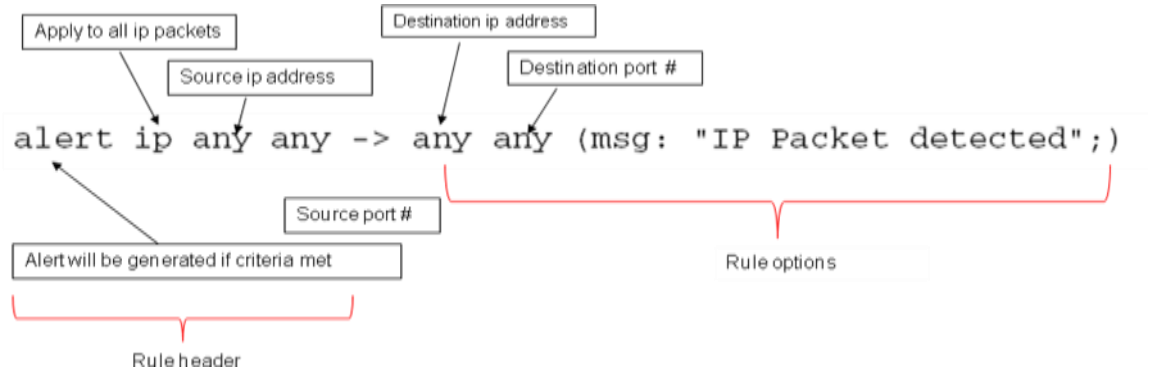

Figure 6: A Typical Example for Snort Rule

Rule header: It specifies the following [figure 6]:

- Rule action
- Protocol
- Source and destination address
- Source and destination port

Rule action: Tells snort what to do whenever a packet matches the rule criteria. There are five default actions available in snort or we can also define our own rule types. Default rule action:

- Alert: An alert is generated using selected alert method and packet is logged
- Log: log the packet
- Pass : ignore the packet
- Activate: alert is generated while turning on another dynamic rule.
- Dynamic: remains silent until an active rule activates it.

Protocol: There are various protocols that snort analyze for suspected behavior for example TCP, UDP, ICMP and IP.

Port numbers: These numbers are defined in many ways including any port, static port definitions, ranges and by negation

Direction operator: Operator -> indicates the direction of the traffic on which rule is applied.

Rule option: This follows the rule header and are enclosed inside a pair of parentheses. There may be one option or many options, separated with a semicolon.

IV. CLASS DIAGRAM FOR PROPOSED SYSTEM

The simple class diagram in figure 7 shows all the classes and their relationship. The class layout can be superimposed onto the system model to see how the model and the class diagram is related. The main class provides an interface to interact with the system. A thread process for Control Unit acting as a controller to direct packets to different event analyzer is also started. Another thread process the Sensor Unit is also created to sniff the network constantly. Concurrently, the controller retrieves a packet captured by Sensor Unit and sends to Analyzer Unit for high-level analysis to generate information to create an event. Analyzer Unit uses Packet Analyzer to inspect the packet's header and type to extract useful data. The analyzed packet or event is delivered to three event analyzer. The PortScan unit analyze the event to display port status on the monitored system. DisplayTable shows the detail of incoming and outgoing packets. The Alert checks the event for security problem by comparing it with rules stored in RuleData unit. When a problem is detected, Alert creates

AlertInfo to log the event and triggers CounterMeasure unit to log the event into a file and generate a HTML page for presentation of events stored in log file.

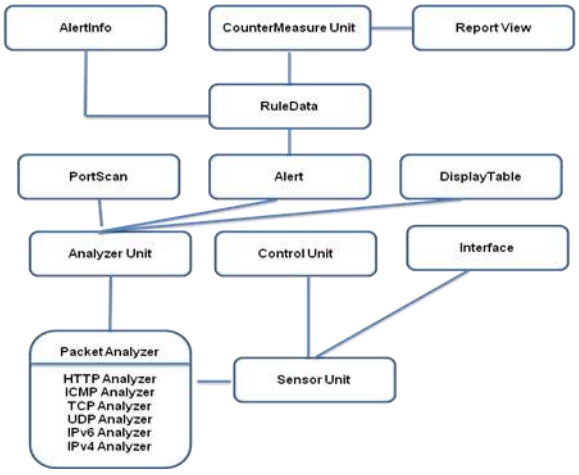

Figure 7: Class Diagram of the Proposed IDS

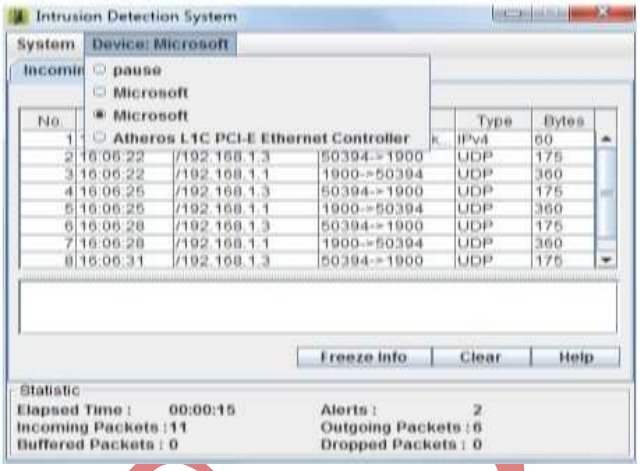

Figure 8: Menu Bar For Network Device Selection

The interface unit provides friendly interaction between the user and the system. JSwing which the part of Java foundation classes (JFC) software is used to set GUI components with a pluggable look and feel. Most parts of this class is defining how each JSwing component are drawn on screen and action to perform when an event has occurred. The request to generate Report requires lots of processing from storage device. The Device menu bar in figure 8 list the network adapter and lets the user select one f the detected network adapter to monitor. The `pause' temporary stops the sensor from capturing any packets from the network until a network adapter is selected. The method setDeviceMenu() is called from the constructor and the return value of Jpcap.getDeviceDescription() is passed into the method [figure 9]. Jpcap will interact with WinPcap/LibPcap to retrieve a list of detected and supported network adapter in the system. If Jpcap returns `Unknown', it means that either the detected adapter is not supported or WinPcap/LibPcap has not been installed properly. The monitor (sensor) box is similar to a packet sniffer which monitors network adapter and makes a copy of all packets on the monitored network adapter. The packets are interpreted and events containing useful and relevant information is generated and provided to the rest of the system. The monitor box consist of 3 major component : Sensor, Packet Analyzer and Buffer Storage. The task of coordinating between classes to handle the packet sniffed by the sensor is done by Control unit the controller class.

```
Private NetworkInterface devices []=jpcap.getDeviceList();
String dnames[]=new String [devices.length];
      for(int i=0; i<devices.length;i++)
        dnames[i]=devices[i].description;
      setDeviceMenu(dnames);
.....
.....
Public void setDeviceMenu (String as [])
   {
     AbstractAction abstractaction = new AbstractAction() {

        Public void actionPerformed (ActionEvent actionevent)
        {
          JRadioButtonMenuItem jradiobuttonmenuitem = (JRadioButtonMenuItem) actionevent.getSource();
          deviceMenu.setText ("Device:" + jradiobuttonmenuitem.getActionCommand ());
          SetDevicename (jradiobuttonmenuitem.getActionCommand());
```

Figure 9: Source Code for Extracting Network Devices

The sensor is a thread which runs concurrently with the rest of the system and capture packets by monitoring a network adapter on a computer. When the thread process of Sensor is created and started, the run method will be automatically called. In the run method, the thread will wait until a network adapter had been selected. Once the instance of Jpcap has been created, the thread will begin to sniff the network for packets by using the method run()[figure 10]. This method invokes the library routine in WinPcap/LibPcap to capture packets. Once a packet had been caught, the receivePacket() method will be invoked. Here, the captured packet is passed to the controller for storage in buffer. The sensor can capture packets faster, so a buffer is needed to store these raw packets captured by the sensor. The Linked List ('LinkedList packets') is used to implemented a First-In-First-Out stack buffer. The first packet on the buffer will be removed and processed by the packet analyzer first. The method addPackets() is called by Sensor unit to add the captured packet into the Linked List array for storage. When Control unit instance is created, a thread process is created and the run method is called[figure11]. The run method will constantly checks if there are packet in the linked list storage then deletes the first packet in the FIFO queue to make way for new packets.

```
Public void run ()
   {
     System.out.println("Done");
     While (thisSensorThread!=null){
       synchronized (this)
       {
         try
         {
           while(jpcap==null)
           {
             wait();
           }
         }
         catch(InterruptedException
interruptedexception){}
       }
       jpcap.processPacket(-1, this);
}
}
public void receivePacket(final Packet packet1)
   {
     String info=packet1.toString();
     controller.addPackets(packet1);
   }
```

Figure10: Source Code For Sensor Unit

```
synchronized void addPackets(Packet packet)
   {
     if(packet == null)
     {
       return;
     }else      {
       packets.add(packet);
       return;
     }  }
public void run()
   {
.....
//Remove the oldest packet queue if buffer limit is reached
if(packets.size() > 0)
        {
      For(; packets.size() >= Config.SENSOR_BUFFER_LIMIT - 1;
      display.add_DropCount())
          {
            if(Config.DEBUG_TRAFFIC)
            {
              System.out.print("-");
            }
            packets.removeFirst();
          }//Remove packet from queue for event analysis
```

Figure 11: Source Code For Control Unit

The method analyze_packet() [figure12] moves the packet into Analyze unit for packet analysis. A high-level event will be generated and will delivered into three different event analyzer; the Table1(for incoming packet) and Table2 (for outgoing packets) which are objects from DisplayTable. It will display the detail of the event or packet in a table. The Table3 object from PortScan will take in the target IPaddress and shows the port status of the system. The Table4 object is generated from Alert which checks the event for problem and generate an alert when found.

Now Each of these raw packet analyzer is able to extract the header data belonging to its packet type. The PacketAnalyzer is an abstract class which forces method header name in all the raw packet analyzer. The selected header data will be stored in a hash-table in Analyzer unit. This information is used by event analyzer for further analysis.

```
synchronized void analyze_packet(Packet packet)
  {
//   Add the records  of packets in the tables
     if(!analyzer.analyze_packet(packet))
     {
        return;
     }
 if(analyzer.matchLocalIP(analyzer.get("addressIP_src"))>0)
     {  //Outgoing packects
        display.Table2.addRow(analyzer, analyzer.get("addressIP_dst"));
        display.Table3.addRow(analyzer.get("port_src"), analyzer.get("port_dst"));
        display.Table4.scanPacket(analyzer.get("addressIP_dst"), analyzer, packet.data);
        display.add_OutCount();
     } else
     {   //incoming packects
        display.Table1.addRow(analyzer, analyzer.get("addressIP_src"));
        display.Table3.addRow(analyzer.get("port_dst"), analyzer.get("port_src"));
        display.Table4.scanPacket(analyzer.get("addressIP_src"), analyzer, packet.data);
        display.add_InCount();
     }
     `
```

Figure12: Source Code for Control Unit to Analyse Packets

The Event Analysis boxes as in figure 13 analyze input from monitor sensor box which generates event. The packets are send to different event analyzer for analyzing. Useful and relevant information will be extracted to interpret an event. These events are then stored on their respective storage box for future references. When a problem had been detected, the countermeasure box will be triggered. The system port number is extracted from the event and delivered into the port analyzer. If the port analyzer cannot find the system port number in its buffer, it will assume a new system port had been opened.

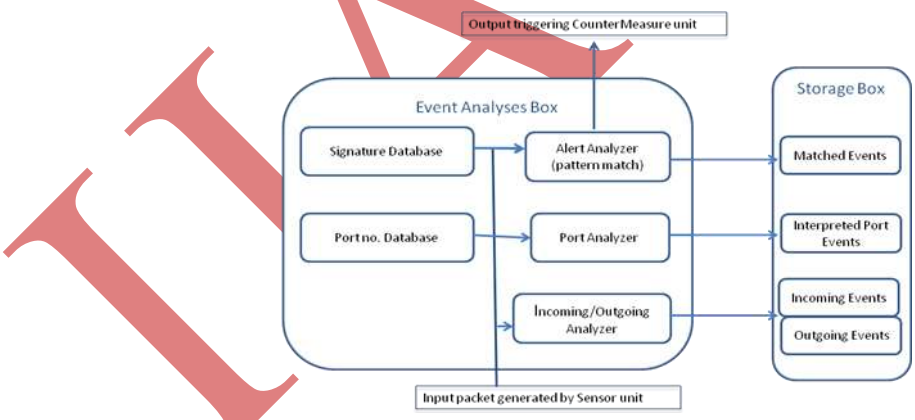

Figure 13: Event Analyses Box

The Alert Analyzer does a signature (rule) analysis by pattern matching the event with a database of known attack patterns or rules. These rules are loaded from the rule file and an alert is generated when there is a suspected problem [figure 14]. The rule (signature) file (rules.txt) is adopted from the free network based intrusion detection system, SNORT. The Alert analyzer starts by doing an extensive search on TCP, UDP or ICMP type rules based on event type and perform content pattern matching between the rules and event in text or hexadecimal mode depending on the rule. Once a pattern had been matched, the countermeasure box will trigger. When a problem had been detected by the Alert Analyzer in the Event Analysis Box, the CounterMeasure box will be triggered. The type of counter measures had been defined in the rule (signature)

file by the rule which triggers the countermeasure. Countermeasure box can also generate a HTML report (report.html) from the log file as shown in figure. The external storage file [figure13] is the log file created to log a problem detected by the alert analyzer in the analysis box. The log file can be reported almost into any database system for further analysis. To maintain smaller log file, only the header information of the packet is logged into the alert log file.

```
public void readRuleFile()
   {
      // Buffer for new rules;
       ruleTCP = new ArrayList();
       ruleUDP = new ArrayList();
       ruleICMP = new ArrayList();
       ruleIP = new ArrayList();
      ..............
 //Reading Rule file and skip all lines without starting alert
       try
       {
       ...................
      String line = bufferedreader.readLine();
             if(line== null)
             {
                break;
             }
            if(line.startsWith("alert"))
             {
                int i = line.indexOf('(') == -1 ? line.length() :line.indexOf('(');
                StringTokenizer stringtokenizer = new StringTokenizer(s1.substring(0, i));
                if(stringtokenizer.countTokens() == 7)      //   tokanizing rule Header
                {
                   String ruleOpt = "";
                   String ruleHdr[] = new String[stringtokenizer.countTokens()];
                   for(int j = 0; stringtokenizer.hasMoreTokens(); j++)
                            ruleHdr[j] = stringtokenizer.nextToken();

          if(line.endsWith(")") && line.indexOf('(') != -1)
             {
             ruleOpt = s1.substring(s1.indexOf('(') + 1, s1.length() - 1);   // Store rule Option
              }
                   If (ruleHdr [1].equals ("tcp"))
                   {
                      RuleTCP. Add (new tRuleData(ruleTCP.size()+" TCP", as, s2));
                      ;
                   } else
                   if(ruleHdr[1].equals("udp"))
                   {
                      ruleUDP.add(new RuleData("UDP " + ruleUDP.size(), as, s2));
                   } else
                   if(ruleHdr[1].equals("icmp"))
                   {
                      ruleICMP.add(new RuleData("ICMP " + ruleICMP.size(), as, s2));
                   } else
                    if(ruleHdr[1].equals("ip"))
                   {
                      ruleIP.add(new RuleData("IP " + ruleIP.size(), as, s2));
                   } else
                  .....................................;
             }
          } while (true);
```

Figure 14: Source Code for Alert Generation

V. EXPERIMENTAL RESULTS

The outputs are shown in screen snapshots from figure 15 to figure 18 and the alert output caught on single run is framed in HTML report form [figure19] which shows source IP addresses of faulty website. Traffic monitoring graph is shown in figure20. The results show that the approach followed in this paper is quite effective and efficient for detecting the network based attacks.

VI. CONCLUSION

In this work, the proposed intrusion detection system is implemented in Java. This system has been tested on a closed network by simulating different types of attack. The proposed system detects all these attacks correctly. The proposed network intrusion detection system is extensible and portable and much other functionality can be implemented. Nevertheless, it presents some drawbacks. First the proposed system takes into account only the scenario approach. The behavioral approach will be examined in the future. Evaluating an intrusion detection system is a difficult task. Indeed, it can be difficult even impossible to identify the set of all possible intrusions that might occur at the site where a particular intrusion detection system is employed. A smart intruder who realizes that an IDS has been deployed on a network will likely attack the IDS first, disabling it or forcing it to provide false information (distracting from the actual attack in progress, or framing someone else for the attack). In order for a software component to resist attack, it must be designed and implemented with an understanding of the specific means by which it can be attacked. Unfortunately, very little information is publicly available to IDS designers to document the traps and pitfalls of implementing such a system. Further studies can be done to improve the security aspect of proposed IDS so that attacks which compromise the availability of the system can be detected or prevented. Although the presented framework is efficient, robust, less network load, scalable and efficient utilization of memory but the security and performance will be the important area for researches for network based intrusion detection system

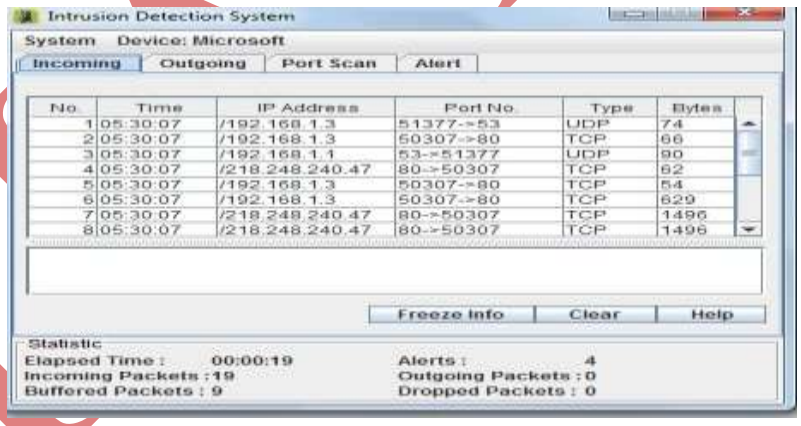

Figure 15: Snapshot for Incoming packets

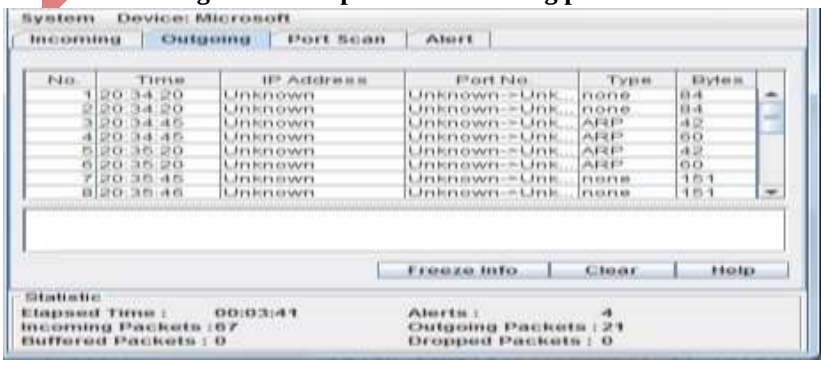

Figure 16: Snapshot for Outgoing packets

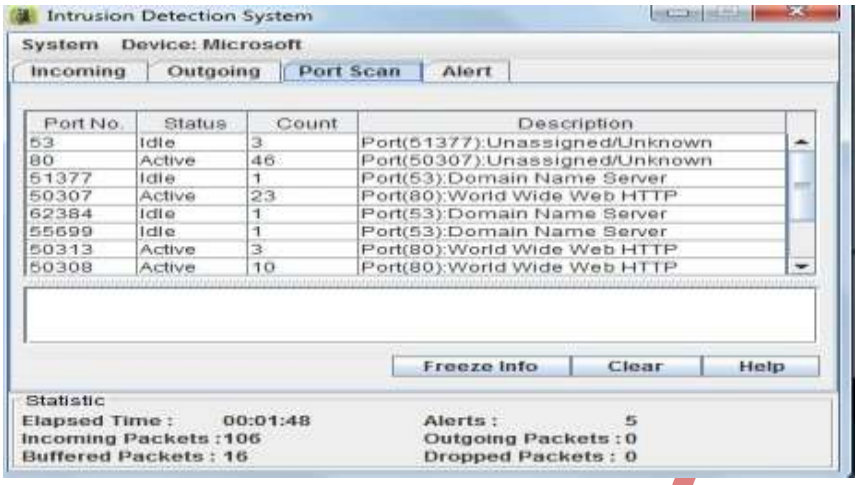

Figure 17: Snapshot for Capture Port

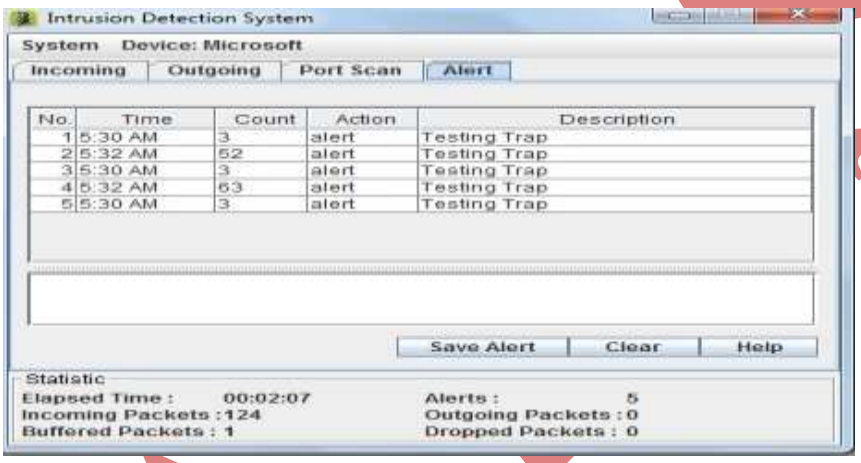

Figure 18: Snapshot for Generating Alerts

Figure19: HTML Report for Captured Alerts

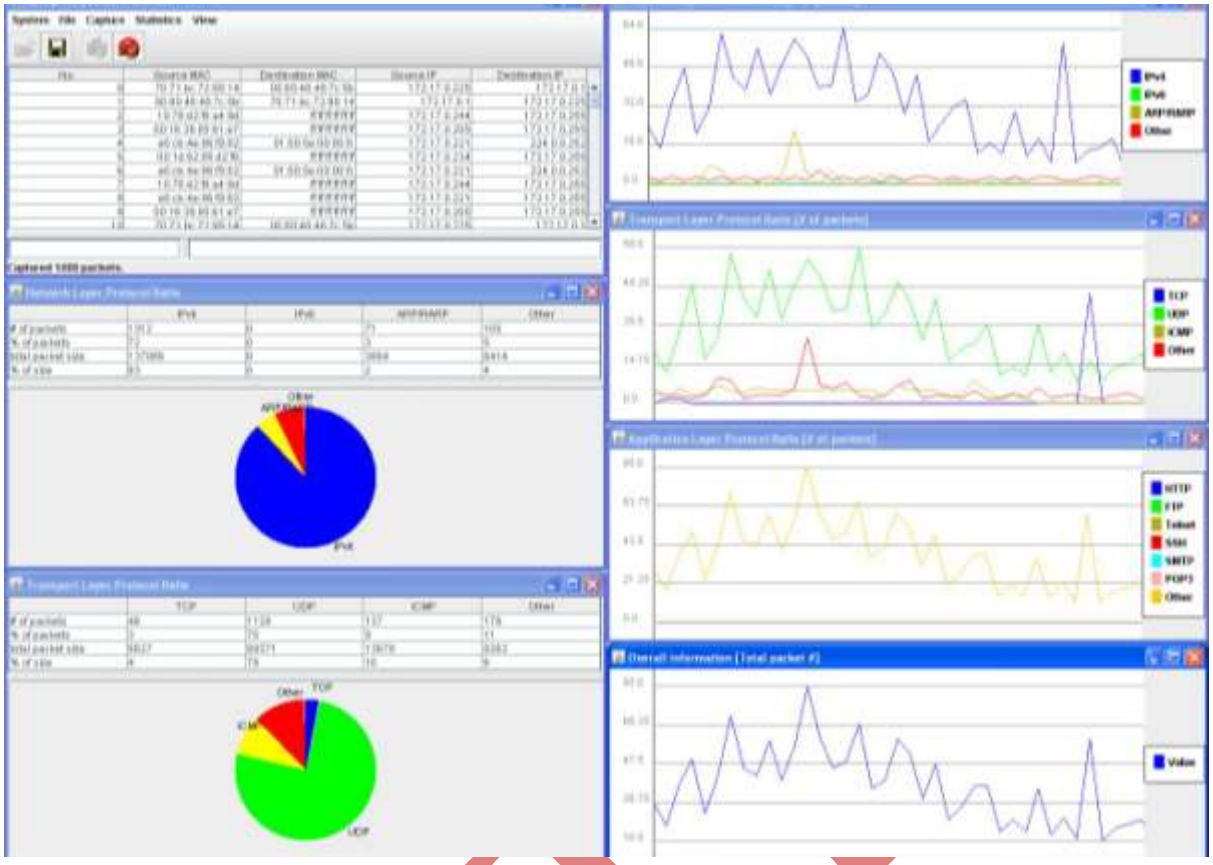

Figure 20: Screens for Network Traffic Monitoring

REFERENCES

[1] J.P Anderson. "Computer Security Threat Monitoring and Surveillance" Technical report, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.

[2] Denning D. E. "An intrusion detection model ", 1987 IEEE transaction on Software engineering, SE-13:222-232 .

[3] Martin Roesch (2009), ―Snort User Manual 2.8.5, available: http://www.snort.org/assets/125/snort_manual-2_8_5_1.pdf.

[4] Keita Fujii, Jpcap (java package for libpcap), Tech. report, School of Science and Engineering, Waseda University, Last Modified on Jan 2007.

[5] Martin Roesch, Snort - lightweight intrusion detection for networks, Tech. report, Lawrence Berkeley National Laboratory (LBNL) in Berkeley, California, 1999, http://www.snort.org/lisapaper.txt.

[6] Kemmerer R. A. and Giovanni Vigna, "Intrusion Detection: A brief History and Overview", Security &Privacy 2002.

[7] Stafen Axelsson, S. "Intrusion detection systems: A survey and taxonomy". Technical Report No 99-15, Dept.of Computer Engineering, Chalmers University of Technology, Sweden, March 2000.

[8] www.winpcap.org

[9] "Intrusion Detection Systems: Definition, Need and Challenges", SANS Institute (2001).