Chapter 5
Rule Learner and Multithreading Techniques for Online Intrusion Detection System for High Speed Network

5.1. Introduction
The performance of rule-based intrusion detection system depends on the rule sets produced by rule learner. Usually, rule formation process is a slow and time-consuming task due to the huge amount of packets in networks. Currently, most of the exiting IDS are offline and incapable to handle the packets in high speed network. To develop real time and fast intrusion detection system is a great challenge in the field of intrusion detection [79]. To develop real time IDS, the primary task of algorithm that capture and process online packets in network. Secondly, there is a need of some modules to handle packets in parallel as per their protocol. In this research work, we proposed a novel framework for intrusion detection system using multithreading technique and cache management mechanism. The proposed system is capable of logging events for cybercrime investigation. The
experimental results show that the proposed system is superior in term of model building time, packet processing time, accuracy and speed as compared with earlier proposed system and other standard classifier.

5.2. Motivation

In chapter 4, we have presented a novel method of intrusion detection system using Ripple down Rule leaner. A very less number of rules are formed by the proposed rule leaner. Though, the system generates less number of rules, but it suffers with some limitations [78]. Firstly, the proposed system is not effective in handling packets in very high speed network. It takes more time for matching online packets with rules in database, due to more access time to handle the rules from database. Secondly, the system is does not have any logging, report generation and preventive capability. So there is a need of developing a real time and fast in accessing rules intrusion detection system. Taking motivation with the above limitations of the proposed system, we have extended our earlier work by adding modules which handles packets in very high speed networks. In this chapter, we present a novel framework for IDS. Our earlier work has been extended by adding multithreading technique, cache management modules, and logging mechanism. Following subsections describe the modules of new framework for intrusion detection system. The main involvement of this chapter is summarised as, firstly we have proposed a novel algorithm for handling packets in network based on their protocol type. Secondly, to increase matching process of captured packets with rules in database, we have proposed a novel algorithm to speed up the matching process using cache management. Finally, the system is improved by adding the logging, preventive and report generation mechanism. This work has been reported in [81].
5.3. Proposed Framework of Online Intrusion Detection System

The objective of the proposed framework work is to speed up the process of packet capturing and processing for online detection of intruders. In this contribution, the multithreading technique has been implemented to capture and process packets in parallel. Then, the cache management module is added to speed up the matching rules with captured packets. The basic system architecture of the proposed rule based IDS is represented in Figure 5.1. It depicts how the packets are pre-processed and database transaction can be used to access rules in the cache from the complementary database.

Figure 5.1: Proposed framework of intrusion detection System.
The proposed system works in two phases; training phase and application phase. In chapter 4, we have discussed the training procedure of the system. After the training phase, rules are formed and stored in the database. Then, these rules are further used to test online packets in the network. Accuracy and false positive rates are calculated on NSL_KDD test dataset. For online testing of the system, we set the threat model to include our assumptions and the type of attack aiming to protect against. Some attacks are simulated in the system to test the model online. We make use of a JPCAP class of JAVA language to simulate and send anomaly packets in the system. Step by step procedure of the proposed system is described in following subsection subsequently.

5.3.1. Packet Sniffing and Packet Filtering Module
The packet sniffers are used to capture online packets over the network. In this contribution, we have implemented our own packet sniffer using a JPCAP class of Java Language. It is used to online capture packets and sent it to the packet pre-processing module. Packet pre-processor module segregates captured online packets according to protocols of the received packets [79] [80]. Traffic of received packets in the network is categorized in three main classes such as TCP, UDP and ICMP. These categorized packets are then forwarded to intrusion detector module for determining a class of the same packet. Detector module spawns separate threads to handle these packets according to the protocol of received packets. Mechanism of detector module has been described in the following sub-section in detail.

5.3.2. Application of Multithreading Technique
In Rule-base IDS, every packet must be compared with each rule in the database. Firstly, this process slows down the intrusion detection, particularly when the network traffic flow is in a blast, prominent to the likelihood of missing a potential attack. Secondly, when a novel service or protocol is
presented in the network, the network administrator is supposed to update or enhance signatures in the database. Detection module receives the packets using a sniffer over the network. This module is a single activity that determines whether the packet captured by sniffer is an intruder or not. In normal traffic, a single process works well. However, if the network is burst, there is a possibility of missing possible attack due to dropping extra packets in the network. It also can slow down detection process. To deal with this problem, we have made use of the multithreading technique. The separate threads are spawned, according to the protocol of received packets. New threads for each protocol are spawned if the numbers of packets are increased in the network. Figure 5.2 illustrates the spawning of the threads using multithreading technique based on protocol.

![Figure 5.2: Spawning of threads according to protocols.](image)

According to Figure 5.2, main process thread captures online packets and segregates them into three groups, namely TCP, UDP and ICMP packets. Child threads are spawned to handle packets belonging to each protocol. Each child thread dynamically creates new child threads. The number of threads
spawned for each protocol depends on the count of packets received by each protocol. As the number of receiving packets of each protocol increase, the number of child threads also increases. We developed an algorithm that takes the input of received packets and groups them in classes according to protocols. We make use of multithreading technique to develop an algorithm which executes in parallel on the basis of protocol. Algorithm 5.1 describes the process of grouping and spawning threads of a single protocol. This multithreading algorithm is used to spawn multiple threads depending on a number of packets received at each protocol. PacketCount keep track of captured packets and ThreadNo variable is used to count the number of threads created by Detection process. A single thread can handle N number of packets. Initially, ThreadNo variable is reset to 1. Principal thread will be produced and it will wait for the first packet. Afterwards, it handles up to N packets. On receiving N packets, a novel thread is created to lever or handle next incoming packets. Thus, second thread handles N to 2N packets, third thread handles 2N to 3N packets and so on.

**ALGORITHM 5.1: Spawning of threads.**

**INPUT:** Number of Packets Received from Sniffer

**VARIABLES:** PacketCount=0, ThreadNo=1, Size=N (user can assign value of N)

**PROCESS:**

Start: For each captured packet
PacketCount++;
If (PacketCount! =Size)
Jump to Start
Else
ThreadNo++;
Create newThread ();

**OUTPUT:** Number of Threads for each Protocol
Algorithm 5.2 had developed for handling packets of a different protocol. Algorithm 5.2 frequently calls Algorithm 5.1 according to the protocol of the received packet. The process of multithreading techniques is described in Algorithm 5.2 as given below. The main thread of Algorithm 5.2 is used to capture packets and add packets in three vectors. The numbers of receiving packets are checked by the main thread of each group. If the number of packets is exceeding N, then a new thread is spawned dynamically. Algorithm 5.2 is introduced to explain the spawning of threads in the system according to the protocol of the packet. Algorithm 5.2 calls Algorithm 5.1 frequently according to the type of protocol of received packets over the network.

**ALGORITHM 5.2: For Creating Threads According to Protocols of Received Packets**

**INPUT:** Number of Packet from Sniffer  
**VARIABLES:** Packets_per_Thread, No_of_Threads, Max_No_Threads  
**PROCESS:**  
1. Initialize Vectors: TCPVect, UDPVect, ICMPVect, and TEMPvect  
2. Max_No_Thread==TEMPVect.size()/Packets_per_Thread  
3. Main thread capture packet and add in vectors according to protocol TCP, UDP and ICMP.  
//Threads is spawned dynamically as given below:  
4. If(capturedThread==TCPacket&&No_Of_Threads<=Max_No_Thread)  
If (TCPVect.size()>0) then  
TEMPVect<-TCPVect.  
Apply Algorithm 1// Create(NewThread)  
No_of_Threads++;  
5. Repeat 4 step for UCPPacket and ICMPPacket  
**OUTPUT:** Number of Threads in system.

### 5.3.3. Detection and Cache Updating Modules

Detection module of the proposed framework of IDS consists of three sub-modules such as frequent attack rule database management, detecting and cache updating modules. In the rule-based intrusion detection system, rules
are frequently accessed from the database in the cache. Cache memory contains the rules, which are frequently called when intruders are found. The main task of detection module is to process the packet, and determine the class of the packet. The detecting module accepts the captured packet as input and abstracts its features. These extracted features are then associated with all the rules in the cache memory first. If any match happens, then the packet is determined as intruder packet. Still, if no match happens, then the extracted features are compared with all the rules in the complementary database. In this case, the rules in the complementary database are accessed in the cache memory. The packets those do not match with any rule in the complementary database are determined as a normal packet. If the database is large, it is a time-consuming process. To deal with this problem, we have implemented cache updating mechanism.

Updating module is answerable for keeping frequent attack rule database up to date in the cache. We have developed an algorithm that takes care of updating rules (Attack rules) in the cache from the complementary database. Algorithm 5.3 describes the cache updating mechanism in detail. Variable attackCount in Algorithm 5.3 is used to stores every record in the frequent database as well as in the complementary database. When intruder detects with a record in the database the variable attackCount is incremented. Updating module works according to the increment of attackCount value. This algorithm runs at regular intervals. Rules for new or unknown attack can be written by studying pattern or behavior of new attacks. These new rules should be added to the database without halting running intrusion detection process. Thus, the system is capable of adding new rules to the database.
**ALGORITHM 5.3: Cache Updating**

**INPUT:** Pattern of received packet

**VARIABLES:** m - Maximum capacity of Cache, n - the number of existing entries in Cache.

Initially Cache==Empty  // frequently attacking cache is empty.
AttackCount==0          // All Field in complementary database is reset to zero.
MaxOccurrance=={Assign value of Threshold}

**PROCESS:**
For every rule in complementary database
If (AttackCount>=MaxOccurance)
Transfer this Rule in Cache
If (n<m)
n++;
Insert Rule in n location in Cache
Assign AttackCount==0
Else
Search an entry in Cache database with lowest AttackCount.
Switch this entry with the selected Rule in complimentary database
Make the AttackCount of both Rules==0

**OUTPUT:** Frequent Rules in Cache.

---

### 5.3.4. Logging, Preventive Capability and Report Generation of the Proposed IDS

Logging mechanism is a vital component of intrusion detection system. It is used to log unclassified packets for reprocessing and further investigation. This logging module is also required for recording of data related to unknown noticed events. The logged data or packets can be spent to authorize the validity of alarm, to correlate events and to investigate incidents. To deal with this, we have implemented logging mechanism which is capable of logging the unclassified packets and events for future packet analysis and investigation. This logging module logs all the information of the packet such
as the source and destination address which can be used for a cyber-forensic purpose. Following important data fields of packets are logged in log files.

- Host
- Timestamp (usually time, date)
- Protocol
- IP addresses of Source and destination
- Number of bytes
- Hop
- Priority
- Sequence Number, Flags and offset

The network administrator can study and analyze the received packet from logged file. Accessing and reading the logged data from the database is inconvenient for an administrator or any other expert. To deal with this issue, we have implemented a converter which can convert log file into PDF file. The system can generate PDF file so that it can distribute among all experts for the study and analysis of the events. The intrusion detection system which prevents the attack is called as an active system. The presentation mechanism is the beauty of this intrusion detection system (IDS). It consists of one cooperative agent who performs in-between the host node and central server. Whenever an intruder is detected, the host reports it to an agent. In the next step, the agent sends this attack information to the central server. Centre server can block all the packets from an intruder’s source IP address or disconnect the TCP connection. The Server also can be configured the setting of the firewall for taking preventive action against intruders. The packets which are matched with any one or more rules in the database are indicated as a vulnerable packet. The details of vulnerable or unrecognised packets can be displayed on the screen by double clicking on these packets. The details of these packets can be used for future investigation. This feature of the system is helpful in forensic/cyber crime investigation. When vulnerable packet is
observed in the system, then an alert signal is given through alarm to network administrator for further action.

5.4. Experimental Results and Discussions

In chapter 4, we have seen that the proposed system is slow because of more time taken for matching rules with online packets. To overcome these problems the system is extended by adding multithreading and cache management modules. The system proposed in this chapter is fast in packet pre-processing and detection due to multithreading technique. Initially, the rules have been generated using RIpple down Rule learner. After this, these rules have been stored in the complementary database which can be used to detect abnormal packets in the network. The performance of the proposed IDS is evaluated in both offline and online mode. The performance of the proposed intrusion detection system is evaluated along with existing standard classifiers and earlier proposed system on test dataset and using 10-fold Cross Validation. The experimental results are arranged in Table 5.1.

<table>
<thead>
<tr>
<th>Name of Classifier</th>
<th>FP (CV)</th>
<th>Accuracy In % (CV)</th>
<th>FP (Test Dataset)</th>
<th>Accuracy In % (Test Dataset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bays</td>
<td>0.102</td>
<td>90.22</td>
<td>0.194</td>
<td>76.68</td>
</tr>
<tr>
<td>Random Tress</td>
<td>0.003</td>
<td>99.66</td>
<td>0.193</td>
<td>77.17</td>
</tr>
<tr>
<td>C4.5/J48</td>
<td>0.004</td>
<td>99.64</td>
<td>0.189</td>
<td>76.22</td>
</tr>
<tr>
<td>AdaBoost (Base Classifier Decision Stump)</td>
<td>0.065</td>
<td>94.00</td>
<td>0.199</td>
<td>74.58</td>
</tr>
<tr>
<td>Proposed Bagging (PART) Ref. [64]</td>
<td>0.003</td>
<td>99.71</td>
<td>0.172</td>
<td>78.37</td>
</tr>
<tr>
<td>Proposed RLMIDS</td>
<td>0.003</td>
<td>99.71</td>
<td>0.153</td>
<td>80.77</td>
</tr>
</tbody>
</table>

According to Table 5.1 and Figure 5.3, the classification accuracy of proposed system on test data set is better as compared with earlier proposed ensemble method and other existing standard classifiers. It is also observed that the
accuracy of both proposed approaches are same on 10-fold cross validation, but better than other standard classifiers. Experimental results show that the proposed system is better in classification accuracy and false positive rate as compared with earlier proposed ensemble method. It is also observed that the proposed system is faster in rules matching process as compared with earlier proposed ensemble [Bagging of PART].

<table>
<thead>
<tr>
<th>Classifier Accuracy (CV)</th>
<th>Classifier Accuracy Test Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bays</td>
<td>90.22</td>
</tr>
<tr>
<td>Random Tress</td>
<td>76.68</td>
</tr>
<tr>
<td>C4.5/J48</td>
<td>99.66</td>
</tr>
<tr>
<td>AdaBoost (Base Classifier Decision Stump)</td>
<td>99.64</td>
</tr>
<tr>
<td>Proposed Bagging(PART) Ref. [64]</td>
<td>94</td>
</tr>
<tr>
<td>Proposed RLMIDS</td>
<td>99.71</td>
</tr>
<tr>
<td>Proposed RLMIDS</td>
<td>80.77</td>
</tr>
</tbody>
</table>

Figure 5.3: Accuracy on cross validation and test dataset.

Figure 5.4 displays false positive value graph of the proposed approach and earlier existing approaches, it is clearly observed that the proposed approach is superior to earlier approaches on test dataset as well as on cross validation.

<table>
<thead>
<tr>
<th>False Positives (CV)</th>
<th>False Positives (Test Dataset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bays</td>
<td>0.102</td>
</tr>
<tr>
<td>Random Tress</td>
<td>0.003</td>
</tr>
<tr>
<td>C4.5/J48</td>
<td>0.004</td>
</tr>
<tr>
<td>AdaBoost (Base Classifier Decision Stump)</td>
<td>0.065</td>
</tr>
<tr>
<td>Proposed Bagging(PART) Ref.[64]</td>
<td>0.003</td>
</tr>
<tr>
<td>Proposed RLMIDS</td>
<td>0.003</td>
</tr>
<tr>
<td>Proposed RLMIDS</td>
<td>0.153</td>
</tr>
</tbody>
</table>

Figure 5.4: False Positives on cross-validation and test dataset.
Confusion matrix is used as a measurement criterion of proposed rule based classifier. Confusion matrix of the proposed ensemble classifier on cross-validation is shown in Table 5.2. According to Table 5.2, the values of Type-I and Type-II errors are very low. It indicates that the proposed approach of intrusion detection system reduces both the false values.

<table>
<thead>
<tr>
<th>Classified As</th>
<th>Normal</th>
<th>Anomaly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>67254</td>
<td>89</td>
</tr>
<tr>
<td>Anomaly</td>
<td>265</td>
<td>58365</td>
</tr>
</tbody>
</table>

Table 5.2
Confusion Matrix of the Proposed RLMIDS on Cross-validation

5.5. Chapter Summary and Conclusions
In this chapter, our earlier research work reported in chapter 4, has been extended by adding multithreading technique and cache management modules. The proposed intrusion detection system is a simple rule-based. The multithreading technique is used for processing the online packets in parallel. It also deals with high-speed network. Initially, the proposed intrusion detection system divides the flow of high-speed network into three groups matching to the protocol such as Transport Control Protocol (TCP), UDP and ICMP. The Multithreading Technique is consumed to handle the traffic of packets of the individual protocol. The first thread of each protocol can handle an assured number of packets in the network. If the number of receiving a packet goes beyond the certain number of same protocols, then new threads are spawned to handle next certain number of packets in the network. The same multithreading process is used for all protocols.

This rule-based intrusion detection system has a facility to add and delete the rules from the database. The rules of frequent attack are accessed in the cache memory. If the event doesn't match with rules in the frequent database, then the rules are accessed from the complementary database. This approach of
Rule Learner and Multithreading Techniques for Online Intrusion Detection System for High Speed Network

rule accessing increases the speed of detection module. The experiments are performed on the proposed intrusion detection system. The experimental results displays that the proposed system outperforms other standard classifiers and earlier proposed system. The average classification accuracy of the proposed intrusion detection system is 80.77 %, and the False Positive rate is 0.153 on testing dataset. On Cross Validation, the accuracy of the proposed approach is 99.71 % and false positive rate is 0.003. In modern era of IDS, we need to developed hybrid technique of intrusion detection system to cope up with new attacks. Taking motivation of developing a hybrid IDS, in the next chapter, we present a second approach of intrusion detection system using ensemble of neural networks.