International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 12 (2020) pp. 046-055
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Received: 2 January 2019; Accepted: 6 January, 2020; Published: 31 March, 2020

An Efficient Network Intrusion Detection System Using Prospective Backward Oracle Matching Algorithms: An Architectural Approach

Raviteja Gaddam[1], M. Nandhini[2]

[1] Department of Computer Science, Pondicherry University,
Puducherry, Tamil Nadu, India
raviteja.csebec@gmail.com

[2] Department of Computer Science, Pondicherry University,
Puducherry, Tamil Nadu, India
mnandhini2005@yahoo.com

***Abstract*: With the innovations in technology, sophisticated attacks are threatening the major defenses of networks. Nowadays, it is highly impossible to avoid security attacks completely. Network Intrusion Detection System (NIDS) plays a vital role in network security in detecting the attacks that happen regardless of the best defensive methods. Most NIDS search engines use pattern matching algorithms as their core component to detect the signatures of inspecting packets. The selection of pattern matching algorithms greatly affects the performance of NIDS. Challenges like handling huge traffic, high data speed, low detection rate, etc. are also degrading the performance of many existing NIDS. To overcome the stated problems, this paper proposes an efficient NIDS layer-based architecture and designed Prospective Backward Oracle Matching (PBOM) Algorithms and applied at respective layers. PBOM algorithms use reversed patterns and construct factor oracle for better pattern matching and to achieve better results. Hashtable mechanism is used to minimize the memory used to store the state transitions. PBOM algorithms are integrated into the Snort tool and deployed on Kali Linux based environment set up. Experimental evaluation indicates that the proposed design with PBOM algorithms can achieve better detection accuracy, less packet loss, and reduced false alarms.**

***Keywords*: Network Intrusion Detection System, Prospective Backward Oracle Matching, Snort, Kali Linux.**

I. Introduction

The latest technological developments are not only sophisticated common computer users but also cybercriminals. Heidi Shey, a Senior Analyst at Forester depicts "Hackers are carefully picking their victim organization, learning its businesses, understanding its partner relationships, and testing for weaknesses and vulnerabilities" [1]. Cyber attackers have shown new heights of determination with zero-day vulnerabilities and malware now used carefully and attackers are progressively trying to hide [2]. The year 2017 has seen an excessive number of cybersecurity collapses. 2017's major cyber-incidents like Shadow Brokers, WannaCry, Wikileaks CIA Vault 7 Cloudbleed, Petya/NotPetya/Nyetya/Goldeneye, Macron Campaign are examples of how chaotic things have already gotten [3]. When compared to 2017, the year 2018 is pretty much good as there were no many global ransom wares or government leaks apart from Russian Grid Hacking, rampant Data Exposures of US adults, Under Armour whose intention was to a data breach.

Network Intrusion Detection System (NIDS) has sustained as a noticeable real-field tested mechanism in security practices. Yet NIDS is not bulletproof to every attack, it increases the bar for attackers and reduces the intensity of the attack. As an effect, NIDS empirically thwarts huge illegitimate attacks that could possibly incur a loss of big amounts to companies. To understand several solutions and difficulties in the real-time situations of NIDS, this paper discusses some literature of NIDS with Snort [4] as their tool. To design an efficient NIDS, the authors propose an architecture and it's supporting Pattern Matching (PM) Algorithms called PBOM Algorithms [5]. Three variations of PBOM algorithms were designed, integrated into Snort and experimented in a lab setup. Proposed PBOM algorithms use factor oracle and state machine mechanisms for the construction of patterns.

To detect the attacks in an efficient manner, the proposed NIDS uses the proposed PBOM algorithms. The designed NIDS evaluated under several network scenarios and different types of attacks to assess its efficiency in terms of detection rate and false alarm rate.

This paper is organized as follows: Some of the works related to the research papers of existing Snort based NIDS are discussed in Section 2. Some potential directions of this paper followed by the proposed architecture and PBOM algorithms are discussed in Section 3. Section 4 discusses the experimental evaluation and the results followed by the conclusion in Section 5.

II. Related Work

We have analyzed various types of NIDS and their approaches that are operating with Snort and without Snort in the research papers [6][7] that are from various domains like data mining, Cloud Computing, Distributed Computing, Artificial Intelligence, etc. The functionality of these approaches is emphasized in various aspects like false alarm rate, detection accuracy, and scalability.

N Jongsawat and J Decharoenchitpong in [8] used Bayesian Network Learning Algorithms and proposed Behavior-Based Rules for Snort. Wireshark tool was used to capture the packets and these were used for the formation of Bayesian Network. Later framed rules for Snort that were built on network traffic and resolved that the efficiency enhanced. One disadvantage identified it was created by rules. Thus to increase detection rate it needs more rules. Using Bayesian Search Graph Algorithm to verify a number of times for the maximum scoring graph is also its drawback.

Saiyan Saiyod, Khamkone Sengaphay, and Nunnapus Benjamas in [9] discussed Private Cloud-based IDS. They proposed multi-sensors in private cloud to improve behavior detection. This paper suggested rules for Snort like behavior checking, port scanning, etc. To evaluate, this paper assumed virtual machines as attackers, sensors, databases, and monitoring. To assess the performance, Nmap and MIT-DARPA 1999 dataset were used. For the duration of evaluation, different results were recognized at diverse sensors. The main disadvantage here is getting ready rules for each sensor and synchronizing sensors to improve the detection rate.

In [10] authors discussed the usage of the Snort tool in the Cloud environment. They have discussed the operation of Snort and embedding of this tool in the Cloud as a part of principal supervision. By altering the snort.conf file, it analyzed the behavior-based and signature-based functionality of Snort. The main disadvantage is that the authors didn't suggest other diverse methods to solve the problems of Snort i.e. low detection rate and packet loss during heavy traffic flow.

Z. Chiba, N. Abghour, et al. in [11] explained the merging of the Back-Propagation Neural Network (BPN) and Snort to identify attacks in Cloud. Snort was used to discover well-known attacks and BPN for unidentified attacks. To improve the BPN detection rate, this paper discussed an optimization algorithm. But authors didn't mention any methods to prevent DoS attacks and the sharing of this data in the Cloud.

Many of the existing NIDS detection engines' core component is the Pattern Matching (PM) Algorithm. Nevertheless, of its wide deployment, throughput and scalability have reduced the first choice of NIDS due to the expensive PM actions. PM has to inspect every packet against the ruleset. Hence, PM essentially requires intensive resources for both computation and communication during heavy traffic. Numerous studies on NIDS divulge several significant features like the simultaneous search for multiple patterns, large keyword sets for searching, varied keyword lengths for searching, etc. to achieve the maximum efficiency.

Considering these features and inspired by Set Backward Oracle Matching Algorithm (SBOM) [12] and Aho-Corasick Algorithm [13], we have designed PBOM algorithms to improve the pattern matching efficiency. SBOM constructs factor oracle with a set of keywords.

A factor is a substring of a word. A factor oracle is a variety of data structures that catalog the entire factors of the given word. It is a type of deterministic automata that identifies every state as an accepting state. To identify the factors of given keywords, it maintains transitions starting from the first state. During state transition for any input, if it encounters a character that is not defined, then the given input is not a factor of the word. A sample factor oracle can be observed in Fig. 1.

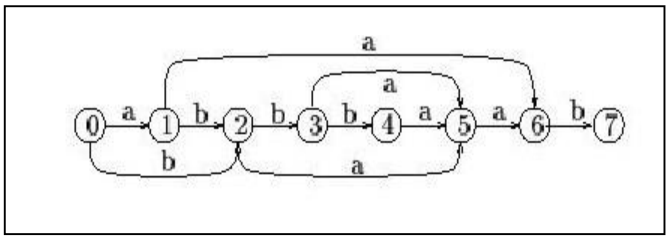

Figure 1. Sample factor oracle

Aho-Corasick uses a state machine for the transitions. In our PBOM, we have used the combination of both factor oracle and state machine. In one variation of PBOM (i.e. PBOM2), we used a hash table mechanism to reduce the memory.

III. Proposed Architecture and Algorithms

Intrusion Detection System is a safeguarding tool that every organization desires. But there are some challenges for the organizations while setting up like huge traffic, low detection rate, manual observation, false alarming, etc. [14]. These restrictions can ascertain stimulating while setting up NIDS. Efficiency can be enhanced by concentrating on several critical attacks like Port Scanning, Session Hijacking and Denial of Service [7]. Networks are susceptible to various kinds of attacks that impends the Availability, Confidentiality, and Integrity. Certain attacks capture the data, whereas others try to alter. Some may attempt to bring down the network and services. Attacks like these may cost a lot in terms of economic, status, customer slow destruction, etc. As a prospective direction, the authors propose an NIDS architecture to thwart this type of attack in an efficient manner.

A. *Our Proposal for improvement*

To overcome various NIDS challenges and to handle various attacks, the authors propose a novel architecture as shown in Fig. 2. This architecture must do the job perfectly without degrading the performance of NIDS.

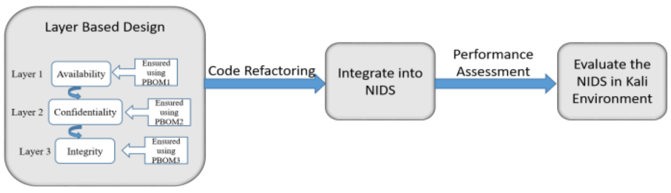

Figure 2. Proposed Architecture to improve NIDS efficiency

1) *Layer Based Design:*

To ease the complication, layers in the design are incremental to guarantee the security features of the network: Availability, Confidentiality, and Integrity. DoS type attacks aim the network services availability, Packet Capturing, and Port Scanning attacks intimidate Confidentiality and Session hijacking attacks impend integrity. These features must be ensured by the network in scenarios like huge traffic and high data speed. The above specified three features may entrust by designing a three-level based design where each level detects and prevent certain attacks and these levels are incremental i.e.; a layer can detect the attacks that are intended for the previous layer also. As a support to handle the attacks with better detection capability, the authors designed PBOM algorithms for the three layers.

2) *Integrate into NIDS:*

To test the proposed design, incorporate it into a NIDS tool like Snort by Code Refactoring. Proposed PBOM algorithms are implemented, placed in the tool with necessary configuration changes.

3) *Evaluate the NIDS in Kali Environment:*

Improved NIDS (i.e. enhanced Snort tool) can be deployed on a Kali Linux based system and assess it by offending the local network experimental setup with various attacks. Monitor and record all the results in various performance aspects. Later, assess the results and verify the efficiency of the suggested design.

B. *PBOM Algorithms for better pattern matching*

For any NIDS, the risk of algorithmic complexity attacks restrains the use of PM Algorithms that are open to the input size. Thus, to improve pattern matching, the authors propose Prospective Backward Oracle Matching (PBOM) algorithms. These are an extension of Allauzen et al. [12] Multiple Backward Oracle Matching algorithm but they didn't describe the particulars in the work.

As an enhancement to Allauzen et al. [12], proposed PBOM uses State Machine (like Aho – Corasick Trie) and Multiple Keyword-based Factor Oracle. A set of all the keywords are used to build both the state machine and factor oracle. But reversed patterns are used for Factor Oracle construction. Algorithm 1 shows the construction of a Factor Oracle with Multiple Keywords

Algorithm 1: Multiple Keyword Set Factor Oracle Construction

```
1  Input:
2  $x$ : keywords array
3  $m$ : array of p integers that are of keyword lengths
4  $factororacle = TrieConstruct(x, m, p)$
5  $i$ = factor oracle root
6  $S[i] = \phi$
7  for $q \leftarrow statesfromfactororacle$ do
8      $l$ = parent(q)
9      $k = S[l]$
10     while $k \neq \phi$ and no transition from k to q do
11         Create transition and label it
12         $k = S[k]$
13     end
14     if $k$ is $\phi$ then
15         Intitialize $S$ to factor oracle root $i$
16     else
17         Index state where $k$ leads to transition
18     end
19 end
```

1) *PBOM1 Algorithm*

For a given set of patterns, it first builds a state machine by directing all the patterns to it. Later, it constructs a factor oracle by supplying the reversed patterns. Here reversed patterns are used to reduce the matching complexity. From all the patterns of the window, we calculate the *minimum* by performing a search for shortest length pattern. During the pattern matching process, it maintains a *critical position* to mark the right side of the stopping position. It starts matching process with the factor oracle. Because of using reversed patterns in factor oracle, process of matching the input with the factor oracle's pattern drives from right side to left side. This searching process stops with a mismatch or at the finishing of scanning all the input characters.

No forward move in factor oracle can be the consequence of a mismatch. This is the point where the process is shifted to state machine. Modify the *current state* to the initial state of the state machine and change the *critical position pointer* to the right side of the mismatched character. Continue the matching process in the state machine until the longest prefix match occurs. As stated in Algorithm 2, PBOM1 doesn't require input scanning as single characters. Multiple characters can be supplied at a time in a scanning window. Pre stoppage of *critical position pointer* skips all the further matching process and window length prevents adding excessive input characters

Algorithm 2: Pattern Search Using Prospective Backward Oracle Matching 1

```
Input:
$t$ : text input array of $p$ keywords
$n$ : text length
$factororacle = OracleConstruct(t, n, p)$
$statemachine = ACSMConstruct(t, n, p)$
$minimum$ = shortest pattern length
$criticalpostion = 0$
while $k$ is in factor oracle and $criticalposition$ $\langle\rangle$ $n$ do
    $l$ = $k$ + $minimum$ - 1
    $cur$ = start at oracle initial state
    while $l$ is greater than $criticalposition$ and $cur$ state exists do
        $l$ = $l$ - 1
    end
    if $l$ is greater than $criticalposition$ then
        $state$ = ACSM initial state
        $criticalposition$ = $l$ + 1
    end
    while $criticalposition$ is less than $n$ and depth($state$) exists do
        $state$ = scan a character
        $criticalposition$ = $criticalposition$ + 1
        if $state$ is terminal then
            print $criticalposition$ - length matched
        end
    end
    $k$ = $criticalposition$ - depth(state)
end
```

2) PBOM2 Algorithm

To empower the performance in the PBOM1 search preference, factor oracle nodes comprise pointers to an array of 256 to other probable nodes in the factor oracle. Every state or node in the Oracle holds 1024 bytes for these pointers. Excluding certain memory, consumptions will lead to better performance. Thus, PBOM2 represented the nodes with 16-bit integers. Hashtable was used to build the Factor Oracle in PBOM2 as shown in Algorithm 3, which can cover all transitions by reducing the memory wastage. Every key in the hash table is a pair of state number and character. Every time mapping occurs between the state numbers that are a part of the hash table key. That is memory can be saved by allocating only to the existing transitions in factor oracle.

3) PBOM3 Algorithm

The PBOM3 as shown in Algorithm 4, doesn't implement any new technique but it uses one of the above algorithms. The main purpose of this algorithm is to specify the difficulty that any algorithm suits performs well at all conditions. The difference in the grouping of patterns is tricky. In the course of pre-processing, all the patterns are added to a pattern group structure that is suitable for other algorithms. Observe minimum length patterns, and if it goes beyond two then opt PBOM2 and compile the pattern group for it. Otherwise, opt PBOM1. Hence, the PBOM3 algorithm wisely chooses PBOM1 or PBOM2 that would typically work effectively for pattern matching based on the pattern group.

Algorithm 3: Pattern Search Using Prospective Backward Oracle Matching 2

```
Input:
$t$ : text input array of $p$ keywords
$n$ : text length
$factororacle$ = OracleConstruct(t,n,p)
$hashtable$ = HashTableConstruct($factororacle$)
$statemachine$ = ACSMConstruct(t,n,p)
$minimum$ = shortest pattern length
$criticalpostion = 0$
while $k$ is in factor oracle and $criticalposition$ $\langle\rangle$ $n$ do
    $l$ = $k$ + $minimum$ - 1
    $cur$ = start at oracle initial state
    $state$ = scan a character
    $t$ = getNode($cur$,$state$)
    while $l$ is greater than $criticalposition$ and $t$ state exists do
        $l$ = $l$ - 1
        $state$ = scan a character
        $t$ = getNode($cur$,$state$)
    end
    if $t$ is NULL then
        $state$ = ACSM initial state
        $criticalposition$ = $l$ + 1
    end
    while $criticalposition$ is less than $n$ and $hashtable[state]$ exists do
        $state$ = $hashtable[state]$.NextState
        $criticalposition$ = $criticalposition$ + 1
        if $state$ is terminal then
            for all pat matches at this state do
                $m$ = $criticalposition$ - length matched
                if not case sensitive then
                    if pattern match found then
                        return match state
                    end
                else
                    if pattern match occurs with memory comparison then
                        return match state
                    end
                end
            end
        end
    end
    $k$ = $criticalposition$ - $hashtable[state]$
end
```

Algorithm 4: Pattern Search Using Prospective Backward Oracle Matching 3

```
1  Input:
2  $t$ : text input array of $p$ keywords
3  $n$ : text length
4  $factororacle$ = OracleConstruct(t,n,p)
5  $statemachine$ = ACSMConstruct(t,n,p)
6  $minimum$ = shortest pattern length
7  $criticalpostion = 0$
8  if $minimum$ is greater than 2 then
9      Set the Search Method as PBOM2
10     for all the patterns of $statemachine$ do
11         AddPatterns(PBOM2,pattern of $statemachine$)
12         Initiate PBOM2
13     end
14 else
15     Set the Search Method as PBOM1
16     for all the patterns of $statemachine$ do
17         AddPatterns(PBOM1,pattern of $statemachine$)
18         Initiate PBOM1
19     end
20 end
```

C. *PBOM Algorithms to NIDS layers*

For the three layer-based design as mentioned earlier, PBOM algorithms can be mapped to preserve the Confidentiality, Integrity, and Availability of the network.

1) *Ensuring Availability*

Availability states the ability of a user to access the resources or information in a truthful manner. The system must be regularly functioning to ensure that it is available at any time in a secure way. Non-functioning of it highly impacts the users. Most of today's cyber-attacks like Denial of Service (DoS) attacks, target to bring down the servers and to effect the companies and persons both in terms of financial and reputation.

To detect DoS like attacks and to preserve the availability feature, PBOM1 algorithm applied at this layer. PBOM1 algorithm search for the flooded packets and alert the administrator by sending the messages and logging the activity. This is to be done in a faster manner so that the administrators can take necessary actions and prevent the crashing of servers.

2) *Ensuring Confidentiality*

Confidentiality allows only authorized persons to access sensitive data. Sensitive information must be disclosed to only those users who have authorization and measures must be taken to ensure it. Failure of confidentiality may lead to a breach where someone who should not have access has managed to get it and can do whatever they want.

PBOM2 algorithm ensures the Confidentiality feature at this level, detects abnormal connection establishment activities and port scanning attacks. This detected information can be sent as alerts to the administrator and logged, which help for restricting or closing the unused ports.

3) *Ensuring Integrity*

Integrity refers to ensuring that the information is real, correct and protected from unauthorized user modification. Data must remain unchanged within a system and during transmission. Hackers try to infiltrate the systems with malware and session hijacking like techniques.

To preserve Integrity, the PBOM3 algorithm monitors and detects any man in the middle attacks like session hijacking is happening in the network. If such activity occurs, it immediately alerts the administrator by sending a message and logging the activity.

All three PBOM variations ensure the better detection capability of various attacks at respective layers. To illustrate the performance of these algorithms, this paper evaluates the proposed architecture by using the Snort NIDS tool.

IV. Experimental Evaluation & Discussion

Since the tenacity of the work is to evaluate PBOM algorithms with Snort to supervise various attacks and its efficiency, the assessment has been carried out in a sophisticated lab environment. Simulation has been done for several attacks like SYN Flooding [15], Port Scanning [16] and Session Hijacking [17] to depict the network scenarios of huge traffic and high data speed. Monitored the performance of Snort per unit time of 15 minutes for these network scenarios for the designed algorithms and one in-built algorithm.

A. *Environment Setup*

The computer lab network is built around D-Link gigabit switches. All machines are hp Compaq, Intel i3, 2.40GHz, 4 GB RAM, 1Gbps inbuilt Network Card. For the evaluation purpose, we have configured 26 machines as shown in Fig. 3 Configuration details of various machines follow.

1) *Victim Server – Host Configuration:*

Victim machine runs on Microsoft Windows 8.1 platform and Microsoft IIS Server feature configured to serve as a web server and one HTML page hosted for evaluation purposes.

2) *Attacker Machine – Host Configuration:*

Kali Linux flavor has been installed on a virtual machine using VMWare workstation player over Windows 8.1 platform. This machine was configured to perform various attacks on the webserver.

3) *Snort Machine – Virtual Configuration:*

On the laptop, the virtual platform has been built using VMWare workstation player over Windows 10 OS and Kali Linux was created as a virtual machine. In this Kali Linux, modified Snort NIDS was installed and configured to monitor the Private Network.

4) *Traffic Generation and Reception Hosts:*

All the remaining machines were used to generate and receive regular traffic on both public and private networks.

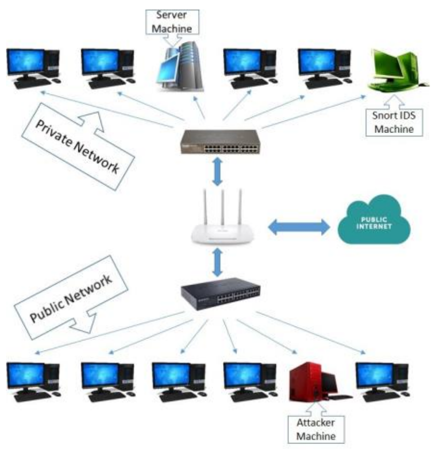

Figure 3. Evaluation Environment Setup

After integrating the PBOM algorithms into Snort using code refactoring, we have installed the modified Snort for intrusion detection and Zabbix [18] to monitor the performance of the Snort NIDS machine. Zabbix [18] is an open-source network monitoring software. It can be associated with Snort to represent the Snort performance details in a graphical form.

B. *Experimentation & Results*

The designed NIDS has to be tested in several scenarios like high data speed, huge traffic and a combination of both. For this experiment, we consider

1) High Data Speed:
Generating 5,000 network packets each 128 bytes size and pushing them into the network at one-second intervals.

2) Huge Traffic:
Generating 10,000 network packets each 128 bytes size and pushing them into the network at fifteen seconds intervals.

3) High Data Speed & Heavy Traffic:
The mixture of the above two cases i.e.; Generating 10,000 network packets each 128 bytes size and pushing them into the network at one-second intervals.

We have used Kali Linux Penetration Testing Tools – hping3 and hamster to generate heavy traffic by performing various security attacks as described below.

4) SYN Flooding Attack:

From the attacker machine, we launched this attack using the hping3 tool as various instances in 5 terminals. Each instance of hping3 floods the victim server with UDP, TCP and ICMP packets, each with 128 bytes packet size and randomizing the source.

5) Port Scanning Attack:
By mentioning a range of reserved port numbers of victim servers as the option to hping3 tool, we launched this attack from the attacker machine. More than 10 instances of it were launched in multiple terminals.

6) SYN Flooding & Session Hijacking Attack [19]:
In this scenario, we tried to flood the victim server as heavy as possible by running hping3 instances in 20 terminals. At the same time, it launched the Session hijacking attack by using hamster tool.

All these attacking scenarios were carried out for a unit time of 15 minutes and every activity was logged and many alerts were generated. To measure the performance of the proposed design, we considered a metric called Traffic Analysis Efficiency (TAE) along with the number of received packets, the number of analyzed packets and the number of dropped packets. We can define TAE as the ratio of a number of analyzed packets to the number of received packets. TAE must be more for assessing the packet processing speed of the pattern matching algorithms. The higher value of TAE represents more traffic packets are processed. Dropping Rate of Packets (DRP) must be reduced to improve the efficiency of the deployed NIDS. We define DRP as the ratio of a number of dropped packets to the number of received packets.

$$\text{Traffic Analysis Efficiency } TAE = \frac{NP_A}{NP_R} \qquad (1)$$

$$\text{Dropping Rate of Packets } DRP = \frac{NP_D}{NP_R} \qquad (2)$$

Since $NP_R = NP_A + NP_D$, our effort is to maximize TAE and minimize DRP .i.e. $\min\left(1 - \frac{NP_A}{NP_R}\right)$

Table 1 shows various scenarios of experimental results that are carried by different algorithms.

From Table 1, we can infer that all the tested algorithms were handling more than 85% of incoming huge traffic with high speed except in PBOM1 and PBOM2, where the handling is nearly 75%. We can also observe these details in Fig. 4 and Fig. 5. Because of the high data speed of incoming traffic, the packet drop rate percentage was more than 20%. But in that critical situation also, PBOM3 performed well in all the three test cases. This algorithm competes with the in-built AC-Std algorithm and tried to minimize the packet drop rate especially in the crucial test case of combined High Data Speed & Heavy Traffic.

Algorithm	Network Scenario	Received Packets	Analyzed Packets	Dropped Packets
PBOM1	High Data Speed	87.74%	54.33%	33.41%
	Heavy Traffic	89.33%	62.67%	26.66%
	High Data Speed & Heavy Traffic	74.33%	34.67%	39.66%
PBOM2	High Data Speed	88.64%	59.76%	28.88%
	Heavy Traffic	91.47%	68.54%	22.93%
	High Data Speed & Heavy Traffic	76.34%	47.32%	29.02%
PBOM3	High Data Speed	91.43%	72.47%	18.96%
	Heavy Traffic	97.12%	85.33%	11.79%
	High Data Speed & Heavy Traffic	89.75%	63.11%	26.64%
AC - Std	High Data Speed	92.46%	75.92%	16.54%
	Heavy Traffic	93.77%	81.02%	12.75%
	High Data Speed & Heavy Traffic	88.53%	58.77%	29.76%

Table 1. Processing of packets by various Algorithms

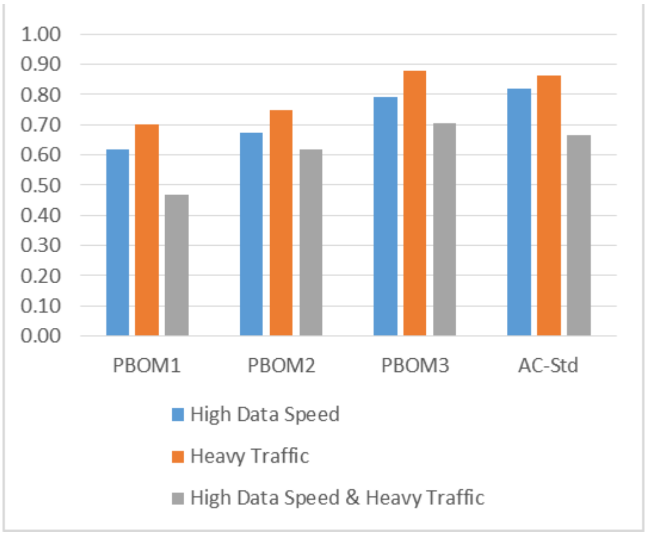

Figure 4. Comparison of Traffic Analysis Efficiency of various Algorithms

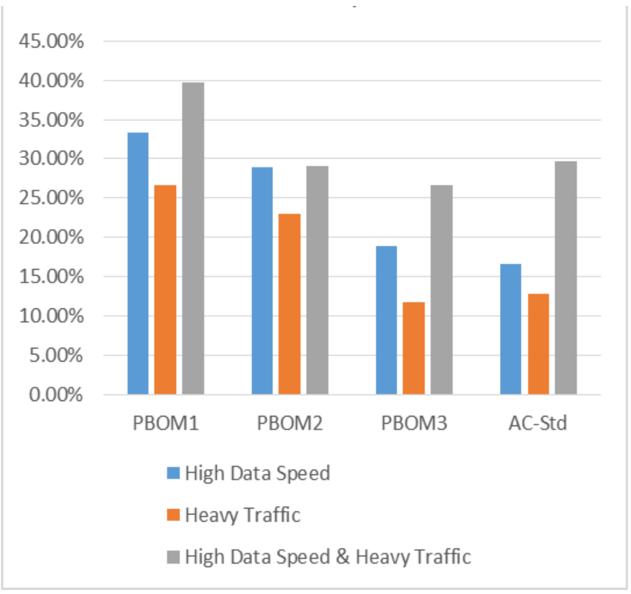

Figure 5. Comparison of Packets Drop Rate of various Algorithms

After evaluating the performance of the algorithms in terms of packets analyzed and packets dropped in high data speed and heavy traffic conditions, now we try to analyze the accuracy. To assess the accuracy of the designed algorithms, we calculate Detection Rate (DR) and False Alarm Rate (FAR). For this, we evaluate the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN).

- True Positives (TP) – identifies Illegitimate Elements as Illegitimate
- True Negatives (TN) – identifies Legitimate Elements as Legitimate
- False Positives (FP) – identifies Legitimate Elements as Illegitimate
- False Negatives (FN) – identifies Illegitimate Elements as Legitimate

$$\text{Detection Rate } DR = \frac{TP}{TP + FN} \qquad (3)$$

$$\text{False Alarm Rate } FAR = \frac{FP}{FP + TN} \qquad (4)$$

To consider a NIDS as efficient, we expect a very high DR and a very low FAR. Along with the above parameters, we can also calculate the Memory Usage (MU) and Average Pattern Matching Percentage (APMP) values as

$$MU = (\, NumberofStates * StateSize) + NumberofTransitions +$$

$$NumberofGroups + NumberofPatterns$$

Where

$$StateSize = 2\text{ bytes}|1048\text{ bytes}|2096\text{ bytes} \qquad (5)$$

$$APMP = \frac{TotalPatternMatchedPercentage}{NumberofTimeStamps} \qquad (6)$$

Where considering only $TotalPatternMatchedPercentage > 9$ for effective calculation because the single-digit percentage doesn't have any impact while calculating APMP.

Table 2 shows the above stated four parameters i.e.; Detection Rate (DR), False Alarm Rate (FAR), Average Pattern Matching Percentage (APMP) and Memory Usage (MU) of the designed Algorithms by using the formulas (3), (4), (5) and (6).

Algorithm	DR	FAR	APMP	MU
PBOM1	0.86 4	0.052	24.93%	176092 KB
PBOM2	0.95 9	0.021	32.69%	9532 KB
PBOM3	0.98 8	0.013	36.10%	1320 KB
AC-Std	0.97 4	0.019	33.21%	161640 KB

Table 2. Performance Parameters of various Algorithms

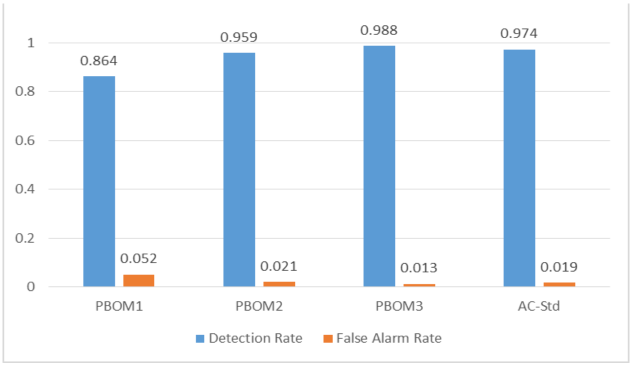

Figure 6. Comparison of Detection Rate and False Alarm Rate of various Algorithms

From Table 2 and Fig. 6, we can infer that the Detection Rate of designed algorithms is promising and especially PBOM3 is better than the in-built AC-Std algorithm. With less False Alarm Rate and more Average Pattern Matching Percentage, PBOM3 wisely handled the heavy traffic with high speed. It also uses less memory when compared to the other algorithms. Moreover, it makes a choice of selecting PBOM1 and PBOM2 and from the analyzed statistics it's clear that PBOM2 is also performing well and competing with AC-Std.

To assess the performance of the proposed algorithms on standard datasets, we considered the NSL-KDD dataset [20] and performed the experiment. Several versions of files are present in the NSL-KDD dataset for training and testing purposes. For the evaluation purpose of the proposed NIDS, this work considers the NSL-KDD Test+ file, which contains a full set of attack records. This data set contains various types of traffic scenarios like DoS, U2R, R2L, Probe and normal. This Test+ file contains a total of 22544 attack records. In them, 7458 are DoS records, 200 are U2R records, 2754 are R2L records, 2421 are Probe records and 9711 are Normal traffic records of this data set were taken for the assessment. All three algorithms were tested by supplying this data set. Table 3, Table 4 and Table 5 show the performance parameters of these algorithms against the NSL-KDD dataset.

Traffic	DR	FAR
DoS	0.998	0.002
U2R	0.862	0.011
R2L	0.878	0.008
Probe	0.891	0.006
Normal	0.972	0.004

Table 3. Performance Parameters of PBOM1 Algorithm

Traffic	DR	FAR
DoS	0.853	0.014
U2R	0.971	0.005
R2L	0.894	0.011
Probe	0.821	0.013
Normal	0.983	0.003

Table 4. Performance Parameters of PBOM2 Algorithm

Traffic	DR	FAR
DoS	0.946	0.008
U2R	0.952	0.010
R2L	0.983	0.003
Probe	0.935	0.013
Normal	0.986	0.003

Table 5. Performance Parameters of PBOM3 Algorithm

From Table 3, we can infer that the Detection Rate of the PBOM1 algorithm is better in detecting DoS attack traffic. This ensures the Availability feature of the network. The average DR is 0.92 and the average FAR is 0.006 of this PBOM1 algorithm on the NSL-KDD dataset.

From Table 4, we can infer that the Detection Rate of the PBOM2 algorithm is better in detecting U2R traffic. This ensures the Confidentiality feature of the network. The average DR is 0.9 and the average FAR is 0.009 of this PBOM2 algorithm on the NSL-KDD dataset.

From Table 5, we can infer that the Detection Rate of the PBOM3 algorithm is better in detecting R2L traffic. The detection of other attacks is also good because of the intelligent selection of the two algorithms. This not only ensures the Integrity, but also the Availability and the Confidentiality features of the network. The average DR is 0.96 and the average FAR is 0.007 of this PBOM1 algorithm on the NSL-KDD dataset.

During the evaluation against this standard dataset, the running time of the algorithms was computed and mentioned the same below.

PBOM1 Algorithm – 14.27356484 sec

PBOM2 Algorithm – 26.49501537 sec

PBOM3 Algorithm – 11.28763853 sec

Table 6 compares the DR and FAR of various existing approaches with the proposed algorithms. The proposed algorithms outperform the other compared algorithms/techniques with better DR and FAR.

Author(s) Reference	Algorithm/ Technique	DR	FAR
[21]	I3DS	0.65	0.15
[22]	Active IDS	0.75	--
[23]	NN	0.914	0.57
[24]	Semi-Supervised Learning	0.95	--
PBOM1 (Live Traffic)		0.864	0.052
PBOM2 (Live Traffic)		0.959	0.021
PBOM3 (Live Traffic)		0.988	0.013
PBOM1 (NSL-KDD DS)		0.92	0.006
PBOM2 (NSL-KDD DS)		0.90	0.009
PBOM3 (NSL-KDD DS)		0.96	0.007

Table 6. Comparison of DR and FAR with various algorithms

From the above assessment, we can observe that the designed three algorithms are performing well not only on live traffic but also on standard datasets like the NSL-KDD dataset. The Detection Rate and False Alarm Rate of these algorithms were promising against various attacks like DoS, Port Scanning, U2R, R2L, and Probe. But the shortcoming is, we didn't yet evaluate for the zero-day attacks [25]. Also, the packet dropping rate during live traffic needs to bring down to consider the proposed design for organizations

V. Conclusion

This paper proposes an efficient architecture for a network intrusion detection system using a layer-based design. For better pattern matching, PBOM algorithms were designed and integrated into the proposed NIDS design. The main reason for incorporating these algorithms is to improve the detection engine efficiency in handling huge traffic with high data speed. To assess the performance of this model, the Snort tool was chosen in the Kali Linux environment. Experimental evaluation depicts that the functionality of the designed algorithms is promising in both live traffic and NSL-KDD dataset scenarios. By analyzing various performance parameters, the competence of the PBOM algorithms can be observed. For future enhancements, we will evaluate the proposed design for zero-day attacks and develop a generalized framework that can be operable in both conventional networks and IoT networks in an efficient manner.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable feedback. We would like to thank our Computer Science Department for providing the necessary resources for our work. This paper reflects the views only of the authors, and others cannot be held responsible for any use which may be made of the information contained therein.

References

[1] H. Taylor, "The key industry that's way behind on data security," *CNBC*. 2018.

[2] K. Chandrasekar *et al.*, "Internet Security Threat Report - April 2017," *Istr*, no. April, 2017.

[3] L. Newman *et al.*, "The Biggest Cybersecurity Disasters of 2017 So Far," *WIRED*. 2018.

[4] "Snort - Network Intrusion Detection & Prevention System," 2018. [Online]. Available: https://www.snort.org/.

[5] R. Gaddam and M. Nandhini, "Prospective backward Oracle matching algorithm for network intrusion detection system," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2017, pp. 1143–1148.

[6] R. Gaddam and M. Nandhini, "An analysis of various snort based techniques to detect and prevent intrusions in networks proposal with code refactoring snort tool in Kali Linux environment," *2017 Int. Conf. Inven. Commun. Comput. Technol.*, no. Icicct, pp. 10–15, 2017.

[7] R. Gaddam and M. Nandhini, "Analysis of Various Intrusion Detection Systems with a Model for Improving Snort Performance," *Indian J. Sci. Technol.*, vol. 10, no. 20, pp. 1–12, 2017.

[8] N. Jongsawat and J. Decharoenchitpong, "Creating behavior-based rules for snort based on Bayesian network learning algorithms," *Proc. 2015 Int. Conf. Sci. Technol. TICST 2015*, pp. 267–270, 2015.

[9] N. Khamphakdee, N. Benjamas, and S. Saiyod, "Improving intrusion detection system based on Snort rules for network probe attack detection," *2014 2nd Int. Conf. Inf. Commun. Technol. ICoICT 2014*, pp. 69–74, 2014.

[10] V. Mishra, V. K. Vijay, and S. Tazi, "Intrusion detection system with snort in cloud computing: Advanced IDS," *Adv. Intell. Syst. Comput.*, vol. 408, pp. 457–465, 2016.

[11] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, "A Cooperative and Hybrid Network Intrusion Detection Framework in Cloud Computing Based on Snort and Optimized Back Propagation Neural Network," *Procedia Comput. Sci.*, vol. 83, pp. 1200–1206, 2016.

[12] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor oracle: A new structure for pattern matching," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1725, pp. 295–310, 1999.

[13] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[14] "The Pros & Cons of Intrusion Detection Systems." [Online]. Available: https://komunity.komand.com/learn/featured/the-pros-cons-of-intrusion-detection-systems.

[15] "Hping3 Examples - Firewall testing |

0DAYsecurity.com," *0daysecurity.com*, 2018. [Online]. Available: http://0daysecurity.com/articles/hping3_examples.html.

[16] "Network Scanning with HPING3," *Devilzlinux.blogspot.com*, 2018. [Online]. Available: https://devilzlinux.blogspot.com/2017/04/network-scanning-with-hping3.html

[17] "IT Security – Session Hijacking." [Online]. Available: http://itsecurity.telelink.com/session-hijacking/.

[18] "zabbix." [Online]. Available: https://www.zabbix.com/.

[19] "hamster-sidejack," *Tools.kali.org*, 2018. [Online]. Available: https://tools.kali.org/sniffingspoofing/hamster-sidejack.

[20] "NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB." [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html.

[21] M. Jha and R. Acharya, "An immune inspired unsupervised intrusion detection system for detection of novel attacks," *IEEE Int. Conf. Intell. Secur. Informatics Cybersecurity Big Data, ISI 2016*, pp. 292–297, 2016.

[22] A. Bhandari, M. Agarwal, S. Biswas, and S. Nandi, "Intrusion detection system for identification of throughput degradation attack on TCP," *2016 22nd Natl. Conf. Commun. NCC 2016*, 2016.

[23] A. a. Alfantookh, "DoS Attacks Intelligent Detection using Neural Networks," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 18, no. 2006, pp. 31–51, 2006.

[24] W. Li, W. Meng, X. Luo, and L. F. Kwok, "MVPSys: Toward practical multi-view based false alarm reduction system in network intrusion detection," *Comput. Secur.*, vol. 60, pp. 177–192, 2016.

[25] FireEye, "Zero-Day Danger: A Survey of Zero-Day Attacks and What They Say About the Traditional Security Model," p. 16, 2015

[26] "Hping3 Examples - Firewall testing |

0DAYsecurity.com," *0daysecurity.com*, 2018. [Online]. Available: http://0daysecurity.com/articles/hping3_examples.html.

[27] "Network Scanning with HPING3," *Devilzlinux.blogspot.com*, 2018. [Online]. Available: https://devilzlinux.blogspot.com/2017/04/network-scanning-with-hping3.html

Author Biographies

Raviteja Gaddam received B.Tech degree in CSE from Bapatla Engineering College and M.Tech degree in CSE from NIMRA College of Engineering & Technology. He is currently pursuing Ph.D. (CSE) at Pondicherry University. He received TCS Gold Medal for standing "Best Student of CSE&IT" during his B.Tech course. He qualified both SET & NET. He worked as a lecturer for three years at Bapatla Engineering College and as an Assistant Professor for six years in St. Mary's Women's Engineering College. His research interests include Network Security, Networking, Cryptanalysis, and Information Security. Currently, he is doing his research work on providing efficient intrusion detection in conventional and IoT networks.

.

Dr. M. Nandhini received B.Sc. and MCA degrees from Bharathidasan University, M.Phil degree from Alagappa University and pursued Ph.D. from Bharathiar University, Tamilnadu and qualified NET with lectureship. Currently, she is working as an Associate Professor in Department of Computer Science, Pondicherry University. She published more than 75 papers in various national and international conferences and journals. Her area of interests includes Evolutionary Algorithms – Soft Computing, Combinatorial Problem Optimization, Artificial Intelligence, and Software Engineering.