NETWORK INTRUSION PREVENTION SYSTEM (NIPS) BASED ON NETWORK INTRUSION DETECTION SYSTEM (NIDS) AND ID3 ALGORITHM DECISION TREE CLASSIFIER

SYURAHBIL

Thesis submitted in fulfillment of the requirements for the award of the degree of Master of Science (Computer)

Faculty of Computer Systems and Software Engineering UNIVERSITY MALAYSIA PAHANG

MAY 2011
We certify that the thesis entitled Network Intrusion Prevention System (NIPS) Based on Network Intrusion Detection System (NIDS) and ID3 Algorithm Decision Tree Classifier is written by Syurahbil. We have examined the final copy of this thesis and in our opinion; it is fully adequate in terms of scope and quality for award of degree of Master of Science (Computer). We herewith recommend that it be accepted in fulfillment of the requirements for the degree of Master of Science (Computer).

External examiner:
Prof. Dr. Kasmiran Jumari
Department of Electrical, Electronic and Systems Engineering
Faculty of Engineering
Universiti Kebangsaan Malaysia

Internal examiner:
Prof. Madya. Dr. Jasni binti Mohamad Zain
Faculty of Computer Systems and Software Engineering
Universiti Malaysia Pahang
SUPERVISOR'S DECLARATION

We hereby declare that We have checked this thesis and in our opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Master of Science (Computer).

Signature
Name of Supervisor : DR. NORAZIAH AHMAD
Position : SENIOR LECTURER FACULTY OF COMPUTER SYSTEMS AND SOFTWARE ENGINEERING, UNIVERSITI MALAYSIA PAHANG
Date : 26/5/2011

Signature
Name of Co-supervisor : MR. MOHAMAD FADLI BIN ZOLKIPLI
Position : LECTURER FACULTY OF COMPUTER SYSTEMS AND SOFTWARE ENGINEERING, UNIVERSITI MALAYSIA PAHANG
Date : 18/5/2011
I hereby declare that the work in this thesis is my own except for quotations and summaries which have been duly acknowledged. The thesis has not been accepted for any degree and is not concurrently submitted for award of other degree.

Signature

Name : SYURAHBIL
ID Number : MCC 08003
Date : 26/5/2011
Dedicated to my parents, my grand parents, my beloved wife, my children
and my big family
ACKNOWLEDGMENTS

I would like to express my most sincere gratitude to the supervisory Dr. Noraziah Ahmad for his continuing support, professional guidance and for giving me an opportunity to learn what research is all about. Special gratitude also to Prof. Abdullah Embong, Mr. Mohamad Fadli Zolkipli and Mr. Mohamad Ramiza Ramli for their contributions, guidance and time towards this research.

Sincerely thanks should be forwarded to Naib Chancellor Universiti Malaysia Pahang (UMP) YH. Prof. Dato' Dr. Daing Mohamed Nasir bin Daing Ibrahim; DIMP, for the Grant Research Scheme. Not forget to Dean Faculty Computer Systems and Software Engineering (FSKKP), Assoc. Prof. Dr. Jasni bini Mohamad Zain; staffs in HR Departement and all friends in UMP.

Special gratitude also to my family, especially to my mother, Syamsiar; in memory of my father, Abdul Hadi; my beloved wife, Neny Thirta Dewi; my child, Zahra Zahirah Muhammad and Arifah Azzura Muhammad; my brother Suhaili, Ramdhan and Muhammad Nasir; my sister Maharani; my mother in-law, Hj. Maharani Rahman, and my father in-law, H. Warnerry Robert, my brother in law, Roy Ferdinand Chandra; my sister in-law, Renny Andhora and Gusti Wulandari; for their patience and morale support. Not forget to big family of Syed Burhanuddin Zulkarnain and dr. Tengku Noraida Adlan are always helpful and supportive to me.

Finally, I thank to all my friends especially to Furqan, Sajadin, Handrizal, Rahmat, Triantoro, Fakhri, Triantoro and others who have contribute in this research.
Network security has gained significant attention in research and industrial communities. Due to the increasing threat of the network intrusion, firewalls have become important elements of the security policy. Firewall performance highly depends toward number of rules, because the large more rules the consequence makes downhill performance progressively. Firewall can be allow or deny access network packets incoming and outgoing into Local Area Network (LAN), but firewall can not detect intrusion. To distinguishing an intrusion network packet or normal is very difficult and takes a lot of time. An analyst must review all the network traffics previously. In this study, a new way to make the rules that can determine network packet is intrusion or normal automatically. These rules implemented into firewall as prevention, which if there is a network packet that match these rules then network packet will be dropped. This is called Network Intrusion Prevention System (NIPS). These rules are generated based on Network Intrusion Detection System (NIDS) and Iterative Dichotomiser 3 (ID3) Algorithm Decision Tree Classifier, which as data training is intrusion network packet and normal network packets from previous network traffics. The experiment is successful, which can generate the rules then implemented into a firewall and drop the intrusion network packet automatically. Moreover, this way can minimize number of rules in firewall.
ABSTRAK

# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td>iii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## CHAPTER 1 INTRODUCTION

1.1 Network Security and Intrusion Detection          2  
1.2 Problem Statement                                  3  
1.3 Objective of The Research                         5  
1.4 Overview of the Thesis                             5  
1.5 Organization of the Thesis                         6  
1.4 Conclusion                                        6  

## CHAPTER 2 LITERATURE REVIEW

2.1 Intruder                                          7  
2.1.1 Denial-of-Service (DoS)                         8  
2.1.2 Probe                                           8  
2.1.3 User-to Root (U2R)                              9  
2.1.3 Remote-to-Local (R2L)                           9  
2.2 Intrusion Detection System (IDS)                  10  
2.2.1 Network-based IDS (NIDS)                        11  
2.2.2 Host-based IDS (HIDS)                           11  
2.2.3 Misuse Detection                                12  
2.2.4 Anomaly Detection                              12  
2.2.5 IDS Products                                    12  
2.3 Intrusion Prevention System (IPS)                 15  
2.4 Logs Files                                       15  
2.5 Firewall
   2.5.1 Types of Firewalls
   2.5.2 Linux Firewalls
   2.5.3 Choosing a Default Packet-Filtering Policy

2.6 Data Mining
   2.6.1 Knowledge Discovery in Database (KDD) and Data Mining
   2.6.2 Data Mining Techniques

2.7 Decision Tree Classifier of Data Mining
   2.7.1 Decision Tree Algorithm
   2.7.2 Motivation of Decision Tree

2.8 Conclusion

CHAPTER 3 METHODOLOGY

3.1 Proposed Framework
3.2 Network Traffic Logs
3.3 Determine Intrusion or Normal Network Packet using Snort NIDS
3.4 Retrieving Data Set as Data Training
3.5 Extract Log Files
3.6 ID3 Algorithm
   3.6.1 Entropy
   3.6.2 Information Gain
3.7 Rule Extraction IF-THEN
3.8 Rules Implemented into Firewall Rules
3.9 Conclusion

CHAPTER 4 IMPLEMENTATION

4.1 NIPS Based on NIDS and ID3 Algorithm Decision Classifier Environment
   4.1.1 Setting Server
   4.1.2 Setting Router and Firewall
4.2 Network Traffic Logs
4.3 Construct Decision Tree using ID3 Algorithm
4.4 Rule Extraction IF-THEN
4.5 Classification Intrusion or Normal of The Network Packet 80
4.6 Implementation Rules of Intrusion Signatures into Firewall Rules 81
4.7 Decision Tree to Create Rules Minimize Firewall of Intrusion 84
4.8 Experimental and Result 88
4.9 Conclusion 93

CHAPTER 5 CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion 95
5.2 Recommendations 96

REFERENCES 97
APPENDICES 108
A Network Traffic Logs 102
B Manual and Listing Program 114
<table>
<thead>
<tr>
<th>Table No.</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Network traffics illustration</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Categories and examples of intruder</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison IDS products</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Network traffics</td>
<td>32</td>
</tr>
<tr>
<td>2.4</td>
<td>Predictive modeling</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Extract Log files into table form</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>Server components specification</td>
<td>54</td>
</tr>
<tr>
<td>4.2</td>
<td>IP address and hostname server</td>
<td>54</td>
</tr>
<tr>
<td>4.3</td>
<td>Application server</td>
<td>55</td>
</tr>
<tr>
<td>4.4</td>
<td>Router and firewall components specification</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Software applications in computer as router and firewall</td>
<td>57</td>
</tr>
<tr>
<td>4.6</td>
<td>Local DNS</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>Data set of network traffics for data training</td>
<td>64</td>
</tr>
<tr>
<td>4.8</td>
<td>Source IP</td>
<td>67</td>
</tr>
<tr>
<td>4.9</td>
<td>Destination IP</td>
<td>68</td>
</tr>
<tr>
<td>4.10</td>
<td>Protocol and Destination Port</td>
<td>69</td>
</tr>
<tr>
<td>4.11</td>
<td>All data set using Protocol = ICMP</td>
<td>71</td>
</tr>
<tr>
<td>4.12</td>
<td>All data set using Protocol = TCP and Destination Port = 22</td>
<td>72</td>
</tr>
<tr>
<td>4.13</td>
<td>Source IP using Protocol = TCP and Destination Port = 22</td>
<td>73</td>
</tr>
<tr>
<td>4.14</td>
<td>Destination IP using Destination Port = 22</td>
<td>74</td>
</tr>
<tr>
<td>4.15</td>
<td>Protocol = TCP, Source IP = 192.168.2.25 and Destination Port = 22</td>
<td>75</td>
</tr>
<tr>
<td>4.16</td>
<td>The new network traffics</td>
<td>80</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.17</td>
<td>To predict new network traffics</td>
<td>81</td>
</tr>
<tr>
<td>4.18</td>
<td>Some rules of all intrusion activities</td>
<td>86</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Firewall rules drop the intrusion network packet</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td><em>Snort</em> log files</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td><em>Syslog</em> log files</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Firewall in computer network</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>The OSI and TCP/IP models</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Packet filtering firewall</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>Policy packet filtering firewall for 10.10.1.2</td>
<td>19</td>
</tr>
<tr>
<td>2.7</td>
<td>Application level gateway</td>
<td>20</td>
</tr>
<tr>
<td>2.8</td>
<td>Circuit level gateway</td>
<td>21</td>
</tr>
<tr>
<td>2.9</td>
<td>Stateful multilayer inspection firewall</td>
<td>22</td>
</tr>
<tr>
<td>2.10</td>
<td>Linux firewall syntax</td>
<td>23</td>
</tr>
<tr>
<td>2.11</td>
<td><em>Iptables</em> firewall rules</td>
<td>23</td>
</tr>
<tr>
<td>2.12</td>
<td>The deny-everything-by-default policy</td>
<td>25</td>
</tr>
<tr>
<td>2.13</td>
<td>The accept-everything-by-default policy</td>
<td>26</td>
</tr>
<tr>
<td>2.14</td>
<td>Data mining as one part of the KDD</td>
<td>28</td>
</tr>
<tr>
<td>2.15</td>
<td>The KDD process</td>
<td>29</td>
</tr>
<tr>
<td>2.16</td>
<td>Clustering</td>
<td>31</td>
</tr>
<tr>
<td>2.17</td>
<td>Classification as the task of mapping an input attribute set $x$ into its class label $y$</td>
<td>32</td>
</tr>
<tr>
<td>2.18</td>
<td>General approach for building a classification model</td>
<td>34</td>
</tr>
<tr>
<td>2.25</td>
<td>Types of decision tree</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>Propose framework of NIPS based on NIDS and ID3 algorithm decision tree classifier</td>
<td>41</td>
</tr>
</tbody>
</table>
3.2 Network traffic logs in computer network

3.3 Fragment of *snort* rules in a file *dos.rules*

3.4 Extract log files of network traffics

3.5 Construct decision tree

3.6 Decision tree of network traffics

3.7 Pruning decision tree network traffics for NIDS

3.8 The decision tree converted to classification IF-THEN rules

3.9 Rules to implemented into firewall rules

4.1 Network topology

4.2 Setting IP address and gateway

4.3 Setting IP address and router

4.4 Configuration of DHCP server

4.5 Configure DNS server zone at /etc/named.conf

4.6 Configure file /var/named/caching-examples/db.research.com

4.7 Configure file /var/named/caching-examples/db.10.10.1

4.8 Translate IP address to DNS and on the contrary

4.9 *nmap* software to scan port host 10.10.1.2

4.10 *nikto* software to get information web server at host 10.10.1.2

4.11 Try to connect host 10.10.1.2 using password root failed through SSH

4.12 The description of the wrong rules in firewall

4.13 The set of 60 examples are 42 intrusion and 18 normal

4.14 Each Gain of attributes

4.15 Protocol and Destination Port as root node

4.16 The set of 4 examples Protocol=ICMP are 4 intrusion and 0 normal
4.17 All of instance of ICMP Protocol is Intrusion = Yes

4.18 The set of 20 examples Dest Port=22 are 17 intrusion and 3 normal

4.19 Source IP decision node under Destination Port = 22

4.20 The set of 20 examples Source Port = 22 are 17 intrusion and 3 normal

4.21 Leaf node Protocol = TCP and Destination Port = 22

4.22 Complete decision tree network traffics

4.23 Pruning decision tree network traffics

4.24 Rules extraction IF-THEN

4.25 Rules of intrusion implemented into firewall rules

4.26 Default *iptables* firewall rules before enter rules

4.27 Command line to enter rules into *iptables* firewall rules

4.28 *Iptables* firewall rules after enter rules

4.29 All of intrusion activities implemented into firewall rules

4.30 Some rules from all of intrusion activities

4.31 *R2* represent *Ra6, Ra7, Ra8, Ra9, Ra10, Ra11, Ra12, Ra13, Ra14*

4.32 Extract and compile software *nips-nid2s3*

4.33 All of data training in ‘data’ file

4.34 Execute and then to see into firewall rules

4.35 Ping using protocol ICMP from 192.168.2.21 to 10.10.1.2 is not reply

4.36 Request SSH from 192.168.0.25 to 10.10.1.2 is connection timed out

4.37 Request URL from 192.168.0.5 to 10.10.1.2 is taking too long to respond

4.38 Request SSH from 192.168.0.5 to 10.10.1.5 is connection timed out
### LIST OF SYMBOLS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E(S)$</td>
<td>Entropy of $S$</td>
</tr>
<tr>
<td>$E(S,A)$</td>
<td>Entropy $S$ of $A$</td>
</tr>
<tr>
<td>$Gain(S,A)$</td>
<td>Gain $S$ of $A$</td>
</tr>
<tr>
<td>$p^+$</td>
<td>Categorized as positive</td>
</tr>
<tr>
<td>$p^-$</td>
<td>Categorized as negative</td>
</tr>
</tbody>
</table>
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>BIND</td>
<td>Berkeley Internet Name Domain</td>
</tr>
<tr>
<td>DARPA</td>
<td>Defense Advanced Research Project Agency</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial-of-Services</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>HIDS</td>
<td>Host Intrusion Detection System</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>ID3</td>
<td>Iterative Dichotomiser 3</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPS</td>
<td>Intrusion Prevention System</td>
</tr>
<tr>
<td>KDD</td>
<td>Knowledge Discovery in Database</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NIPS</td>
<td>Network Intrusion Prevention System</td>
</tr>
<tr>
<td>NIDS</td>
<td>Network Intrusion Detection System</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>R2L</td>
<td>Remote-to-Local</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>SYN</td>
<td>Synchronize</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>U2R</td>
<td>User-to-Root</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION


Development of the internet and the availability of tools for intrusion by hackers become critical component of network administration. An intrusion can be defined as actions that threaten the integrity, confidentiality or availability of a network resources (SANS Institute, 2008), such as user account, file system, system kernels and so on (Chandrasekar A. et al., 2009 and SANS Institute, 2008.)

Network security system is described by a firewall (Gollmann D., 2006; Al-Shaer E.S. and Hamed H.H., 2004, 2004; Guan X. and Yun-jie L, 2010; Golnabi K. et al., 2006; Tibbs R.W. and Oakes E.B., 2006; Suehring S. and Ziegler R.L., 2006; Katić T. and Pale P., 2007). Firewalls can limit certain network packet, but firewall cannot recognize that is network packet is an intrusion or attack. Conversely, there is no
Intrusion Detection System (IDS) software who can be immediately implemented into the firewall policy rules. IDS software can only generate an alarm or log such as snort, honeypot and portsentry.

1.1 NETWORK SECURITY AND INTRUSION DETECTION

Network Security arise from local computer network connected to wide-area network such as internet. During local network computer not connected to wide-area network, problem of network security is not be important. Network security explains the possibilities to arise from connected local network computer to wide-area network (Joko Y. and Onno W.P., 2008). The global internet connection, network security has gained significant attention in research and industrial communities. Due to the increasing threat of network attacks, firewalls have become important elements of the security policy is generally (Al-Shaer E.S. and Hamed H.H., 2004; Benelbahri M.A. and Bouhoula A., 2007 and Suehring S. and Ziegler R.L., 2006).

Firewall is network security and first line of defense against external network attacks and threats. Firewall controls or governs network access by allowing, denying or forward the incoming and outgoing network traffics (Guan X. and Yun-jie L., 2010; Golnabi K. et al., 2006; PC Perspective, 2008; Tibbs R.W. and Oakes E.B., 2006; Suehring S. and Ziegler R.L., 2006; Mitra S. and Acharya T., 2003; Katić T. and Pale P., 2007 and Yan Y., 2010).

Firewalls can protect network systems and minimize the risk of attacks to the network. Intrusion Detection System (IDS) is good for detecting the existence of intrusion. The technology is joining ability of IDS and protection, so-called Intrusion Prevention System (IPS) (Wiliam W.S.C., 2005). IPS can detect intrusion and then drop of intrusion as prevention. Intrusion prevention is an evolution of intrusion detection.
1.2 PROBLEM STATEMENT

Behavior of intrusion conducts an attack is slowly, continuous and requires a long time. They made preparations to find all information of the system, such as machine and operating system (OS) used, ports, administrator, software application used, network topology and so on. Intrusion learn from the system and perform an attack that no longer generate an alarm (Gollmann D., 2006). All of activities can be seen on network traffic logs. Log can provide a useful information and crucial to be able to distinguish normal activities and intrusive activities (Terpstra J.H. et al., 2004).

Network traffics can be observed from log files (Golnabi K. et al., 2006) as human pattern recognize (Winding R. et al., 2006). For illustration, in Table 1.1 there are 15 records network traffic which consists 10 intrusion network packets and 5 normal network packet. All the network packet of intrusion should be dropped into firewall rules for network security. In Figure 1.1 shows intrusion network packet in Table 1.1 that was implemented into firewall rules.

Table 1.1 Network traffics illustration

<table>
<thead>
<tr>
<th>No</th>
<th>Source IP</th>
<th>Dest IP</th>
<th>Dest Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>122.206.13.100</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>122.306.13.100</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>122.306.13.100</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>122.306.13.100</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>122.306.13.100</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>122.306.13.100</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>203.130.14.20</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>203.130.14.20</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>203.130.14.20</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>203.130.14.20</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>206.145.206.4</td>
<td>10.10.1.2</td>
<td>21</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>206.145.206.4</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>206.145.206.4</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>14</td>
<td>206.145.206.4</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>15</td>
<td>206.145.206.4</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
</tbody>
</table>
Packet filtering firewall can limit the access to the connection base on parameters including protocol, source IP, destination IP, source port, destination port etc (Al-Shaer E.S. and Hamed H.H., 2004; Golnabi K. et al., 2006; Tibbs R.W. and Oakes E.B., 2006; Suehring S. and Ziegler R.L., 2006 and Katić T. and Pale P., 2007). Packet filtering firewall has the character of the static thus function also static and has limitation (Suehring S. and Ziegler R.L., 2006 and Wiliam W.S.C., 2005). For example, access to the web server using port 80 allowed by the firewall policy rule, so from anywhere activities pass port 80 is allowed although there is attempt penetration by intruder. Therefore, these rules are in a constant need of updating by inserting, modifying or removing, tuning and validating (Al-Shaer E.S. and Hamed H.H., 2004; Golnabi K. et al., 2006; Suehring S. and Ziegler R.L., 2006 and Katić T. and Pale P., 2007).

If every intrusion will be implemented to firewall rules which not carefully ordered and selective, this condition will makes the policy contains a large number of firewall rules. The possibility of policy anomaly will be happened relatively high, such as writing conflicting or redundant rules (Al-Shaer E.S. and Hamed H.H., 2004). This condition causes the performance of firewall decreases (Gollmann D., 2006; Dunham and Margareth H., 2002 and Benelbahri M.A. and Bouhoula A., 2007), because incoming and outgoing every network packet must be checked against the rules until the

**Figure 1.1** Firewall rules drop the intrusion network packet
rules found matching (Gollmann D., 2006; Golnabi K. et al. 2006 and Khalil R.K. et al., 2010).

The distinguishing an intrusion network packet manually through log files is difficult, because requiring a lot of time and tedious (Al-Shaer E.S. and Hamed H.H., 2004 and Golnabi K. et al., 2006). An analyst must review all of network traffic. For example in Table 1.1, that describe line 11, 12, 13, 14 and 15 with same source IP 206.145.206.4 were have two categories: intrusion activities and normal activities. It is required a method to create intrusion rules that be able to select a IP address which have two categories. Therefore, the necessary means to selectively and automatically determining a network packet is an intrusion that called NIDS and then to implemented into firewall rules as prevention. Combination both of NIDS and firewall namely NIPS.

The task of manually-manage firewall policy rules becomes very difficult and takes a lot of time, because the number of network traffics are increase and continuous. This huge task which requires a way to automatically update the firewall rules (Golnabi K. et al., 2006).

1.3 OBJECTIVES OF THE RESEARCH

The objectives of the research are as follows:

i. To propose a new framework of Network Intrusion Prevention System (NIPS) based on Network Intrusion Detection System (NIDS) and ID3 Algorithm Decision Tree Classifier.

ii. To implement and analyse the performance of the framework in the computer network environment.

1.4 OVERVIEW OF THE THESIS

The distinguish network traffics is intrusion or normal is very difficult and takes a lot of time. An analyst must review all the data of network traffics to find the intrusion then implemented into firewall rules as prevention.
This research makes rules that can determine network packet is intrusion, then rules are implemented into the firewall as NIPS. Rules resulting from the construct decision tree using ID3 algorithm and network packets as data training. Network packet obtained from the log files that record the activity of network traffics. Network packet intrusion and normal as data training extracted into five attributes: source IP address, destination IP address, source port, destination port and protocol. To determine the packet is a network intrusion or a normal use the Snort NIDS software. Snort also generate log files and parse log files that perform intrusion and normal.

1.4 ORGANIZATION OF THE THESIS

This thesis is organized as follows: Chapter 2 reviews the Intruder, Intrusion Detection System (IDS), Intrusion Prevention System (IPS), Log Files, Firewall, Data Mining and Decision Tree Classifier of Data Mining. Chapter 3, proposes NIPS based on NIDS and ID3 algorithm decision tree classifier. Chapter 4, the implementation and analysis the performance rules in computer network environment. Finally, the Conclusion and Recommendations are presented in Chapter 5.

1.5 CONCLUSION

This chapter explains network security that gained significant attention in computer network. To distinguish the intrusion activities from normal activities of the network traffics is very difficult and require a lot of time. It is needed a method to define intrusion from network traffics automatically that called NIDS. Firewall able to protect network system and can minimization risk of intrusion. NIDS and firewall have become important element of the network security. In addition NIDS can detect existence network intrusion. Combination both of NIDS and firewall namely NIPS can detect and deny existence of intrusion. Meanwhile, network security the highly performance depends of the firewall policy rules (Tibbs R.W. and Oakes E.B., 2006 and Suehring S. and Ziegler R.L., 2006). To generate and managing firewall policy rules require a lot of time. An analyst must review all the data of network traffics previously. Therefore it is needed a process of creating firewall rules that reflect the previous network traffics.
CHAPTER 2

LITERATURE REVIEW

This chapter explains the basic concepts and related work that is part of the methodology. The basic concept are Intruder, Intrusion Detection System (IDS), Intrusion Prevention System (IPS), Log Files, Firewall and ID3 Algorithm Decision Tree Classifier. All of the concept and related work are presented in this chapter.

2.1 INTRUDER

There are many alternatives to perform intrusion. The intruder will find all information about the target computer system and take advantage of the weaknesses of computer systems. Intrusion can be prevented by always update the system and issues of computer network security (Rafiuudin R., 2002).

There are four main categories of intrusion, which are denial-of-service (DoS), probe, user-to-root (U2R) and remote-to-local (R2L) (Khoi-Nguyen T. and Huidong J., 2010; Theodiridis S., 2006 and Ye Q. et al., 2010). Each of these categories represents the generalization of specific attack types. These main categories represent the classification of types of behaviors that can be grouped logically together. For each category, there are multiple attack types and unique to a particular pattern. Table 2.1 shows categories and types of intruder.
Table 2.1 Categories and examples of intruder

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoS</td>
<td>apache2, back, land, mailbomb, Neptune, pod, processtable, smurf, teardrop, upstorm</td>
</tr>
<tr>
<td>probe</td>
<td>portsweep, ipsweep, nmap, mscan, saint, satan</td>
</tr>
<tr>
<td>U2R</td>
<td>buffer_overflow, httptunnel, loadmodule, perl, ps, rootkit, sqlattack, xterm</td>
</tr>
<tr>
<td>R2L</td>
<td>spy, ftp_write, guess_passwd, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, warezclient, warezmaster, worm, xlock, xsnoop</td>
</tr>
</tbody>
</table>

2.1.1 Denial-of-Service (DoS)

Denial-of-Service (DoS) attacks is generally known by its attempts to interrupt a service provided as a part of a network (Chandrasekar A. et al., 2009; Dong S.K. et al., 2005; Firewall is it Needed, 2008; Guan X. and Yun-jie L., 2010 and Khoi-Nguyen T. and Huidong J., 2010). An example of a DoS attack is when a service is flooded with requests that it cannot respond effectively “denying” service to any request. The Transport Control Protocol (TCP) with flag SYN (synchronize/start) flood attack is type of attack, wherein SYN connection requests are made that never closed. DoS attacks have crippled websites for extended periods of time. Some have even taken down large portions of the internet, due to the traffic bottlenecks created.

2.1.2 Probe

Probe are the searches for network vulnerabilities to be used in other attacks (Chandrasekar A. et al., 2009; Dong S.K. et al., 2005 and Guan X. and Yun-jie L., 2010). Typically, a network scanned to find servers, and the servers are then scanned for open ports or known vulnerabilities. The signatures of these attacks are usually easy to identify, due to their searching nature. The danger which they impose is their ability to find vulnerabilities that can be leveraged in another attack. The difficulty in network security is the balance between usability and security. If all resources are tightly restricted, then users are limited in the features or applications they can access. If the
security is loose then it become vulnerable toward attacked. Most networks tend to be loose in their security practices. This is why probes are a successful form of intrusion and pose a threat to security.

One example of a probe attack is a “port scan”. This attack uses the approach of incrementally making request to service port on a system. It verifies whether a known service is running on that port, and it learns the attributes of the service. With this knowledge, a more threatening attack can be formed that advantage of vulnerabilities in poorly secured services.

2.1.3 User-to-Root (U2R)

This attack where an intruder exploitation began in systems with normal user account and attempts to abuse the vulnerability in the system to get super user privileged. U2R attack can be the most damaging to system integrity. A U2R attack consists of a user with normal access privileges counterfeiting root level access and full control over the system. Most of these intrusions with buffer overflows in the operating systems that allow a user to gain root access. With root access, the intruder has complete access and control over the system. Integrity of data and information can be lost or damaged. If intruders gain access to the system as root, it has a greater ability to hide from the intrusion detection.

2.1.4 Remote-to-Local (R2L)

A remote-to-local (R2L) attack allows a remote, non-authorized user to simulate local user privileges on machine (Chandrasekar A. et al., 2009; Dong S.K. et al., 2005 and Guan X. and Yun-jie L., 2010). There are different approaches, including: dictionary-based password and username guessing, attacking vulnerabilities or bugs, and attacking poorly the configured services. Some of these attacks, such as dictionary attacks, are more easily detected than others. As an authorized user on the machine, the intruder can gain the access to private information, disrupt certain services, can corrupt data, or install applications that allow him or her to gain the control of services that can
be used for other malicious behavior. This type of attack can lead to security breaches that allow complete access and control of the machine (user-to-root).

2.2. INTRUSION DETECTION SYSTEM (IDS)

Intrusion detection is detecting actions that attempt to compromise the confidentiality, integrity or availability of resources (Gollman D., 2006; Duanyang Z. et al. 2010; SANS Institute, 2008; Shingo M. et al., 2010 and Weenke L., 2001). When Intrusion detection takes a preventive measure without direct human intervention, then it becomes an Intrusion Prevention System.

Intrusion detection can be performed manually or automatically. Manual intrusion detection is examining log files and then determine network packet is intrusion or not. A system that performs automated intrusion detection is called an Intrusion Detection System (IDS) (Kenneth G.J., 2005; Rafiudin R., 2002 and Winding R., 2006).

Intrusion Detection System the operating systems that allow a user to gain root access. With root access, the intruder has complete access and control over the machine, system is very important in network security. Meanwhile, computer networks continue to expand. IDS need to be able to deal with a large computer network. Therefore, automatic procedures for detecting and responding to intrusion are becoming increasingly essential (Mehmed M.K. and Jozef Z., 2005).

Misuse detection searches for patterns user behavior that match intrusion, which are stored as signatures. These hand-coded of signature are laboriously provided by human experts based on their knowledge of intrusion techniques. If a pattern match is found, then the alarm will appear as a sign. Human security analysts evaluate the alarms to decide what action to take, whether it is shutting down part of the system, alerting the internet service provider about suspicious traffic. An intrusion detection system for a large complex network may produce thousands or millions of alarms per day. Because systems are not static, the signatures need to be updated whenever new software versions arrive or changes in network configuration (Al-Shaer E.S. and Hamed H.H.,
2004). There are two main types of IDS pursuant where data analyzed, which describe in sub chapter 2.2.1 and 2.2.2.

2.2.1 Network-based IDS

Network Intrusion Detection System (NIDS) is an attack of signatures in network traffic. Typically, a network adapter running to monitors and analyzes all network traffic in real time. NIDS is an independent platform that identifies intrusions by examining network traffic and monitors multiple hosts. NIDS gains access to network traffic by connecting to a hub or network switch. An example of a NIDS is software namely \textit{snort} (Gollman D., 2006; Flior E. et al., 2010 and SANS Institute, 2008).

2.2.2 Host-based IDS

Host Intrusion Detection System (HIDS) looks for attack signatures in log files of hosts (Duanyang Z., 2010 and Gollman D., 2006). HIDS are attempts to identify unauthorized and anomalous behavior on a specific device. HIDS generally involves an agent installed on system, monitoring and alerting on local OS and application activity. The installed agent uses a combination of signatures, rules, and heuristics to identify unauthorized activity. The role of a host IDS is passive, only gathering, identifying, logging and alerting of intrusion (SANS Institute, 2008).

There are other ways of finding intrusion attempts. One alternative method is to load software to look for signs of intrusion on the system itself. If a machine has been exploited, often certain system files will be altered. For examples, the password file may be changed, users may be added, system configuration file may be modified, or file permissions might be altered. By looking at changes in these files, there is an intrusion or other unusual activity. There are two classification IDS pursuant how data analyzed, which describe in sub chapter 2.2.3 and 2.2.4.
2.2.3 Misuse Detection

In misuse detection, the IDS analyzed the information it gathered and compare it with large databases of attack signatures. Essentially, the IDS are looks for a specific attack that has already been documented. Learning system from the existing pattern of attack and recognized. This method unable to detect the new attack is which its pattern not yet been known (Duanyang Z., 2010).

2.2.4 Anomaly Detection

In anomaly detection, the system administrator defines the baseline, or normal, state of the network’s traffic load, breakdown, protocol, and typical packet size. The anomaly detector monitors network packets to compare their state to the normal baseline and looking for anomalies.

Statistical anomaly detection uses statistical techniques to detect potential intrusions. During operation, a statistical analysis of the data monitored is performed and the deviation from the baseline is measured. If a threshold is exceeded, an alarm is issued. On the other hand, anomaly detection detects just anomalies. Suspicious behavior does not necessarily constitute an intrusion (Gollman D., 2006).

2.2.5 IDS Products

Many IDS systems exist and a lot of confusion because there is little in the way of standards with how they operate. It is difficult to provide a direct comparison between products because terminology, features and functionality. This is because there is no effective comparison can occur. Example of IDS product include Snort, Honeypot and Portsentry.

a. Snort
Snort is open source NIDS (Yan Y., 2010). Snort capable of performing real-time traffic analysis and packet logging on computer networks. It can perform protocol analysis, searching/matching and can be used to detect a variety of attacks and probes, such as buffer over-flows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting and much mores (Yan Y., 2010).

Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture. Snort has a real-time alerting capability as well, incorporating alerting mechanisms for syslog or a specific file user. Snort has three primary uses (Yan Y., 2010):

- **Sniffer mode** – to see the packet through the network.
- **Packet logger mode** – for recording all the packet through the network to the analysis at a later.
- **NIDS mode** – snort used to detect of attack carried out through computer networks. This mode required NIDS setup of rules that distinguish between normal packets or attack packets.

b. **Honeypot**

Honeypot only provide the security vulnerability of a system, thus providing space for attacked. Honeypot is to trap intruder to penetrate computer system. When there are attacks, honeypot will be recorded in the log files so the admin can do the next action. Honeypot is passive in that they are waiting for someone to attack.

c. **Portsentry**

Portsentry is a software designed to detect port scanning and response actively if any port scanning. The port scanning is a scanning process of services variety applications running on computer servers. Port scanning is the first step before an attack will be undertaken. If there is a machine scan port to servers will actively block the attacker machine.
Portsentry will react in real time by blocking the IP address of the attacker. This was done by using ipchains or ipfwadm and insert into the file /etc/host.deny automatically by the TCP Wrapper. Portsentry reports through log files in /etc/syslog. The report indicates the system name, time attack, the attacker IP machine IP, protocol, and others. Snort and portsentry have different action where prevent intrusion, whereas snort is only detects the intrusion.

IDS produce has a different way of working and has a different way of prevention. Table 2.2 shows the comparison of work and prevention between Snort, Honeypot and Portsentry.

Table 2.2 Comparison IDS products

<table>
<thead>
<tr>
<th>IDS Product</th>
<th>Work</th>
<th>Prevention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snort</td>
<td>Intrusion network packet is known based on the rules. When a network packet has a similarity with a rule, then network packet is the intrusion. New attack has a new signature so that rules should always be updated</td>
<td>✔ generate log ✔ passive</td>
</tr>
<tr>
<td>Honeypot</td>
<td>Provide the security vulnerability of a system. When there are attacks, honeypot will be recorded in the log files.</td>
<td>✔ generate log ✔ passive</td>
</tr>
<tr>
<td>Portsentry</td>
<td>Detect machine who is doing port scanning</td>
<td>✔ generate log ✔ active ✔ blocking IP address of intrusion into file /etc/host.deny</td>
</tr>
</tbody>
</table>

2.3 INTRUSION PREVENTION SYSTEM (IPS)
Intrusion Prevention System prevent from intrusion or attacks. IPS work with an IDS, and vendors have combined the two technologies to make an IPS-capable IDS. Two techniques are used to prevent an attack (Rafiudin R., 2002):

- **Sniping** – Allow the IDS to terminate a suspected attack from through the use of a TCP reset packet or ICMP unreachable message.
- **Shunning** – Allow the IDS to automatically configure router or firewall to deny traffic based on what it has detected and therefore shunning the connection. As IDS become more advanced, this shunning is evolving into a new term, blocking, where an IDS contacts a router or firewall and creates an access control list (ACL) to block the attacking IP.

The IDS can handle sniping. However, shunning requires the assistance of other device. IDS sensors should report back to a central console that, in turn also generates some responses if so configured. Following are some actions that an IDS generate in repose to an attack (Rafiudin R., 2002):

- **Reconfigure firewall/router** – An IDS with a shun enable configure the firewall to filter out the intruders IP address.
- **Send an SNMP trap** – Configure the IDS to send an SNMP trap datagram to a management console.
- **Generate log** – An IDS can log to Windows event log, Syslog server, pager or even send an e-mail.

### 2.4 LOG FILES

Log files are another critical facet in total security architecture. It is important to create a policy and strategy for dealing system and application. Log files are useful for three reasons (John H.T., et al., 2004):

1. Log files help with troubleshooting system problems and understanding what is happening on the system.
2. Logs serve as an early warning for both system and security events.
3. Logs can be indispensable in reconstructing events, whether determine an intrusion has occurred and are performing the follow-up forensic investigation.
Following some examples from Snort log files are shown Figure 2.1

![Snort log file](image1)

From Figure 2.1, there is effort for the scan of existence of Squid proxy server IP address 192.168.0.1 and port 3128 from workstation IP address 192.168.0.33, port 1040 and protocol is TCP.

![Syslog log files](image2)

From Figure 2.2, someone tries to log in using the 'failed password' from IP address 172.18.64.26 and port 3419 passing ssh service using TCP protocol.

2.5 FIREWALL

A firewall is placed between two or more networks, usually a private network and public networks. Typical examples are the internet which is a zone with no trust and an internal network which is a zone of higher trust must be protected. The term firewall comes from the fact that by segmenting a network into different physical subnetworks, they limited the damage that could spread from one subnet to another just like firedoors or firewalls. Figure 2.3 shows firewall in computer network.
Firewall checks every incoming and outgoing network packet to see if it meets the criteria of rules. If it does, firewall will take action to accept, forward, drop or reject the packet network depends the set of rules. Firewalls can filter packets based on source IP address, destination IP address, source port, destination port, protocol and others. Firewalls can filter specific types of network traffic.

The TCP/IP model is older than the Open System Interconnection (OSI) model (Gollman D., 2006; Tibbs R.W. and Oakes E.B., 2006). There are 5 layers in the TCP/IP model. TCP/IP and OSI model is shown in Figure 2.4.
The lowest layer in which the firewall can work is layer three. In the OSI model this is the network layer. In TCP/IP it is the Internet Protocol layer. This layer is concerned with routing packets to destination. At this layer a firewall can determine whether a packet is from a trusted source, but cannot be concerned with what it contains. Firewalls that operate at the transport layer know about a packet and able to grant or deny access depending criteria set of rules. At the application level, firewall know what is going on and can be very selective in granting access.

2.5.1 Types of Firewalls

Based on mechanism or way of firewalls work, there are four main type of firewall, which are packet-filtering, application level gateway, circuit level gateway and stateful multilayer inspection firewall.

a. Packet Filtering

Packet filtering firewalls work at the Network layer of the OSI model, or the IP layer of TCP/IP. They are usually part of a routing. A router is a device that receives packets from one network and forwards them to another network. Rules specifying which packets are allowed through the firewall and which are dropped. In a firewall, packet of filtering each network packet compared to a set of criteria before it forwarded. Depending on the packet and the criteria of rules, the firewall can drop the packet,
forward it or send a message to the source network packet. Rules can include source and destination IP address, source and destination port number and protocol (Gollman D., 2006). Packet filtering firewall is shown in Figure 2.5.

![Packet filtering firewalls](image)

**Figure 2.5** Packet filtering firewalls (Gollman D., 2006)

For example, network packets using port 80 and destination port 23 to IP address 10.10.1.2. Firewall permit and forward network packet to IP address 10.10.1.2 by using port 80 and reject or drop the network packet by using port 23. This is shown in Figure 2.6.

![Policy packet filtering firewall](image)

**Figure 2.6** Policy packet filtering firewall for 10.10.1.2

b. **Application Level Gateway**

Application level gateways also called proxies, are similar to circuit-level gateways except that they are application specific. They can filter packets at the
application layer of the OSI model. Incoming or outgoing packets cannot access services for which there is no proxy. Application level gateway that is configured to be a web proxy and can not allow FTP, telnet and others. Because they examine packets at application layer, they can filter application specific commands such as http: POST and GET, etc. This cannot be accomplished with packet filtering firewalls or circuit level which do not know anything about the application level. Application level gateways can also be used to log user activity and logins. They offer a high level of security, but have a significant impact on network performance. This is because of context switches that slow down network access dramatically (Mladenic D. et al., 2003). Application level gateway is shown in Figure 2.7.

![Figure 2.7 Application level gateway (Gollman D., 2006)](image)

c. Circuit Level Gateway

Circuit level gateways work at the Session layer of the OSI model, or the TCP layer of TCP/IP. This type of firewall is useful to hide information on a protected network packet. Connection between the user and the network is hidden from the user. Users will be confronted directly with the firewall on the connection. Firewall to connect to the network resources are accessed by the user after changing the IP address of the packet transmitted by the two sides. It is like a virtual circuit between users and network resources are accessed. users from the external network can not see the internal
network IP addresses in the packets he receives, but the IP address of the firewall. Circuit level gateway is shown in Figure 2.8.

![Figure 2.8 Circuit level gateway](image)

d. **Stateful Multilayer Inspection Firewall**

Stateful multilayer inspection firewalls combine the aspects of the other three types of firewalls. Packet filtering at the Network layer, determine whether Session packets are legitimate and evaluate contents of packets at the application layer. Stateful multilayer inspection firewalls offer a high level of security, good performance and transparency to end users. They are expensive like Cisco PIX. State multilayer inspection firewall is shown in Figure 2.9.
2.5.2 Linux Firewalls

In general distribution of the Linux Operating System, there is simple software firewall called the netfilter or iptables is available. Iptables has been part of the Linux kernel since version 2.4 (Joko Y and Onno W.P., 2008). There is difference between iptables and netfilter. Netfilter is the Linux kernel space program code to implement a firewall, either compiled directly into the kernel or included as a set of modules. Iptables is the program used for administration of the netfilter firewall. Netfilter firewall are built through the iptables administration command. The iptables command implements the firewall policies. Netfilter firewalls have three individual tables, which are filter, NAT and magle (Flior E. et al., 2010 and Suehring S. and Ziegler R.L., 2006).

The basic syntax for an iptables command begins with the iptables command itself, followed by one or more options, a chain, a set of match criteria and a target or disposition. The layout of the command largely depends on the action to performed. iptables syntax is shown in Figure 2.10.

Figure 2.9 Stateful multilayer inspection firewall (Gollman D., 2006)
There are 3 predefined tables in use by iptables to which can add rules for processing IP packets passing through those tables. These tables are (Al-Shaer E.S. and Hamed H.H., 2004; Jiawei H. and Kamber M., 2006; Joko Y and Onno W.P., 2008 and Suehring S. and Ziegler R.L., 2006):

- INPUT - All network packets to Local Area Network (LAN).
- OUTPUT - All network packets from LAN.
- FORWARD - All network packets incoming to LAN or outgoing from LAN. This machine is called the router.

Iptables check network packet traffic relevant to those tables and a decision is made about what to do with each packet: accept or drop the packets. These actions are referred to as targets. Two most common predefined targets are DROP to drop a packet or ACCEPT to accept a packet.

Firewall can check the packet filtering of pursuant to parameter of IP addresses, protocols (TCP, UDP, ICMP), ports, MAC Address and others (Golnabi K., 2006 and Suehring S. and Ziegler R.L., 2006), so that facilitate the sysadmin for the customize. Following is example of iptables firewall rules is shown in Figure 2.11.

```
-A INPUT -s 203.130.206.5 -p tcp -d 10.10.15.7 --dport 80 --j DROP
```

**Figure 2.11** Iptables firewall rules

Firewall rule is show in Figure 2.11 explaining to enhance the order by the end of chain (A) for the traffic of incoming to firewall (INPUT) by source IP address (-s)
203.130.206.5 with the type protocol (-p) tcp to destination IP address (-d) 10.10.15.7 and destination port (-dport) 80 hence done by action (- j) dropped ( DROP). Policy Firewall rules in Figure 2.11 will do the drop if there is traffic incoming to the firewall by IP address source is 203.230.206.5 type of protocol TCP to destination IP address is 10.10.15.7 and destination port is 80.

A packet was checked each rule in turn, starting at the top, and if it matches that rule, then an action is taken such as accepting (ACCEPT) or dropping (DROP) the packet. If a network packet matches a rule then a network packet is not processed by next rules in the tables. If a network packet is not match any rule, then the default action is taken. This is referred to as the default policy and may be is ACCEPT or DROP the packet (Golnabi K., 2006 and Suehring S. and Ziegler R.L., 2006).

2.5.3 Choosing a Default Packet-Filtering Policy

There are two basic approaches to a default firewall policy (Tibbs R.W. and Oakes E.B., 2006 and Suehring S. and Ziegler R.L., 2006):

- Deny everything by default and explicitly allow selected packets through.
- Accept everything by default and explicitly deny selected packets from passing through.

The deny-everything policy is the recommended approach. This approach makes it easier to set up a secure firewall, but each service and related protocol transaction that must be enabled explicitly. This means that it must understand the communication protocol each service to be enabled. The deny-everything requires more work up to enable Internet access. Some commercial firewall product support only the deny-everything policy. The deny-everything policy is show in Figure 2.12.
The accept-everything policy makes it much easier to get up and running right away, but it anticipate some network packet type to be disabled. The danger is that network packets type is too late to be disabled. Developing a secure accept-everything firewall is much more work and much more difficult. The accept-everything policy is shown in Figure 2.13.

**Figure 2.12** The deny-everything-by-default policy (Tibbs R.W. and Oakes E.B., 2006)
Firewall rules are explicitly written and managed to filter out any unwanted traffic coming into or going from the secure network. However, the management of firewall rules has been proven to be complex, error-prone, costly and inefficient for many large-networked organizations (Guan X. and Yun-jie L., 2010; Mehmed M.K. and Jozef Z., 2005; Benelbahri M.A. and Bouhoula A., 2007 and Tibbs R.W. and Oakes E.B., 2006).

The firewall rules are often custom-designed and hand-written by and for human. Firewall tailored to accommodate ever-changing business and market demands of the global internet. Therefore, these rules are in a constant need of updating, tuning

### 2.6 DATA MINING

Data Mining refers to extracting or “mining” knowledge from large amounts of data. Data mining is the process of employing one or more computer learning techniques to automatically analyze and extract knowledge from data contained within a database. The purpose of a data mining session is to identify trends and pattern data (Dunham and Margareth H., 2002; Duanyang Z. et al., 2010 and Hao-Ran D. and Yun-Hong W., 2007).

The knowledge gained from a data mining session is given a model or generalization of data. However, all data mining methods use induction-based learning. Induction-based learning is the process of forming general concept definition by observing specific examples of concepts to learned.

There are some congeniality of related to data mining (Jiawei H. and Kamber M., 2006):

1. Data mining is to match the data in a model to find information hidden in databases
2. Data mining is the application of algorithms to dig up useful information from the databases
3. Data Mining is process find the pattern in data, where the invention process conducted automatically or semi automatically and pattern found have to be useful
4. Data Mining is process of find information, which is good for depository of big data automatically.
5. Data Mining or Knowledge Discovery in Database (KDD) is information take which hidden, where the information previously the unknown to and have useful potency. This process covers a number of different technical to approach, like clustering, data summarization, learning classification rules.
2.6.1 Knowledge Discovery in Database and Data Mining

Knowledge Discovery in Database (KDD) is a term frequently used interchangeably with data mining. Technically, KDD is the application of the scientific method to data mining (Jiawei H. and Kamber M., 2006). KDD is defined as the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable pattern in data (Duanyang Z., 2010). In addition to performing data mining, a typical KDD process model includes a methodology for extracting and preparing data as well as making decision about actions to be taken once data mining has taken place. When a particular application involves the analysis of large volume of data stored in several locations, data extraction and preparation become the most time-consuming parts of the discovery process. As data mining has become popular name for the broader term like KDD (Dunham and Margareth H., 2002).

![Diagram of KDD process](Figure 2.14)

**Figure 2.14** Data mining as one part of the KDD (Jiawei H. and Kamber M., 2006)

Today, data are no longer restricted to tuples of numeric or character representation only. The advanced database management technology of today is enable to integrated different types of data, such as image, video text and other numeric as well as non-numeric data, in a probably single database for facilitate processing (Jiawei H. and Kamber M., 2006).
KDD process consists of the steps below (Dunham and Margareth H., 2002):

1. **Selection of Data** (data selection), selection of relevant data obtained from the database.
2. **Data cleaning**, the process of removing noise and inconsistent data or data not relevant.
3. **Data integration**, combining data from various databases into a new database.
4. **Data transformation**, data modified or merged into a format suitable for processing in data mining.
5. **Data Mining**, a process in which the method is applied to discover knowledge and hidden data.
6. **Pattern evaluation**, identify interesting patterns to be represented into the knowledge-based.
7. **Knowledge presentation**, visualization and presentation of knowledge about the techniques used to obtain the knowledge gained users.

### 2.6.2 Data Mining Techniques

Data Mining refers to process to dig the added of value from a data aggregate in the form of knowledge which during the time unknown in manual (Al-Shaer E.S. and Hamed H.H., 2004). Word mining it is mean the effort to get a few data worth from a large amount of basic data. In consequence of data mining in fact have the long root from science like artificial intelligent, machine learning, statistical and database. There are some techniques in literature of data mining examples association rule, clustering,

a. Association Rules

As the name implies, association rule mining techniques are used to discover interesting associations between attributes contained in a database. Examples of associative rule is the purchase of analysis in a supermarket. How is possibility a customer buys bread with milk. Knowledge is the owner of the supermarket stuff can arrange a placement or design marketing strategies by using discount coupons for the combination (Duanyang Z., 2010). For another example such as attached mailing in direct marketing, fraud detection for medical insurance and credit cards, department store floor or shelf planning (Jiawei H. and Kamber M., 2006). For this is reason a limited number of attributes are able to generate hundreds of association rules.

Important or not a association rules can be determined by two parameters. Support the combined percentage of these attributes in the database and confidence the strength of the relationship between attributes in an association rules.

b. Clustering

Clustering, defined broadly, is the grouping of similar object. Clustering is the unsupervised classification of patterns into groups based upon similarity (Berry M.W. And Browne M., 2006). Clustering to group the data without a specific data class. Clustering can be used to provide the class labels of unknown data. Therefore, clustering is called unsupervised learning methods.

The principle of clustering is to maximize the similarity among members of a class and minimize the similarity between clusters. Clustering can be performed on the data which has several attributes that are mapped as a multidimensional space.
Figure 2.16 Clustering (Jiawei H. and Kamber M., 2006)

Illustrate Clustering is shown in Figure 2.16. The area is two dimension, customer of shop can be grouped to become some cluster with the center cluster shown by positive sign (+). Many algorithms Clustering need the function of distance to measure similarity between data, needed also method for the normalization attributes that have kinds of data.

c. Classification

The input data for a classification is a collection of records. Each record, also knows as instance, characterized by a tuple $(x,y)$, where $x$ is the attribute set and $y$ is a special attribute as the class label (Berry M.W. and Browne M., 2006) or also called category or attribute of target. An attribute is a property or characteristic of an object that may vary [Dunham and Margareth H., 2002 and Berry M.W. and Browne M., 2006). For example, eye color varies from person to person, temperature of an object varies over time. Eye is a symbolic attribute with a small number of possible values {brown, black, blue, green, hazel, etc}, while temperature is a numerical attribute with a potentially unlimited number of values.
Classification is the task of learning a target function \( f \) that maps each attribute set \( x \) to one of the predefined class label \( y \), illustration is shown in Figure 2.17. Class label attribute must be discrete (Dunham and Margareth H., 2002; Mladenic D. et al., 2003; Jiawei H. and Kamber M., 2006; Berry M.W. and Browne M, 2006 and Roiger R.J. and Geatz M.W., 2003). This is a key characteristic that distinguishes classification from regression, a predictive modeling is a continuous attribute (Roiger R.J. and Geatz M.W., 2003). The target function is also known as a classification model. A classification model is useful for one predictive modeling (Roiger R.J. and Geatz M.W., 2003):

✔ **Descriptive Modeling.** Classification model can serve as a clear tool to distinguish between objects of different classes.

**Table 2.3 Network Traffics**

<table>
<thead>
<tr>
<th>No</th>
<th>Source IP</th>
<th>Dest IP</th>
<th>Source Port</th>
<th>Dest Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>122.206.13.100</td>
<td>10.10.1.2</td>
<td>1360</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>122.306.13.100</td>
<td>10.10.1.2</td>
<td>1425</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>122.306.13.100</td>
<td>10.10.1.2</td>
<td>1488</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>122.306.13.100</td>
<td>10.10.1.5</td>
<td>1559</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>122.306.13.100</td>
<td>10.10.1.5</td>
<td>1620</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>122.306.13.100</td>
<td>10.10.1.3</td>
<td>2156</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>203.130.14.20</td>
<td>10.10.1.5</td>
<td>2158</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>203.130.14.20</td>
<td>10.10.1.5</td>
<td>1624</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>203.130.14.20</td>
<td>10.10.1.3</td>
<td>4207</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>203.130.14.20</td>
<td>10.10.1.3</td>
<td>4607</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
</tbody>
</table>
For example, useful to summarize the data in Table 2.3 and explain the features of a packet is defined as an intrusion or normal.

✔ **Predictive Modeling.** A classification model can also been used to predict the class label unknown record. As shown in Table 2.3, a classification model can automatically determine the class label when presented with a new dataset where the class labels of unknown. Suppose given some new network packet shown in Table 2.4.

<table>
<thead>
<tr>
<th>No</th>
<th>Source IP</th>
<th>Dest IP</th>
<th>Source Port</th>
<th>Dest Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>122.206.13.100</td>
<td>10.10.1.5</td>
<td>1775</td>
<td>80</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>122.206.13.100</td>
<td>10.10.1.3</td>
<td>1775</td>
<td>22</td>
<td>TCP</td>
<td>?</td>
</tr>
</tbody>
</table>

A classification technique is a systematic approach to building classification models from an input data set such decision tree classifier, rule-based classifier, neural networks, support vector machines and naïve Bayes classifiers. Each technique employs a learning algorithm to identify a model that best fits between relationship and the attribute set and also class label of the input data.

The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records that has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability and models that accurately predict the class labels that previously unknown record [29].
Figure 2.18 General approach for building a classification model (Roiger R.J. and Geatz M.W., 2003)

Figure 2.18 shows a general approach for solving classification problems with predict class that has possibility of two categories that are 'Yes' or 'No'. First, a training data set consisting of records whose class labels are known must be provided. The training set is used to build a classification model, which is subsequently applied to the test set that consists of records with unknown class labels.

2.7 DECISION TREE CLASSIFIER OF DATA MINING

Decision tree classification is one of the fundamental techniques used in data mining (Berry M.W. and Browne M., 2006). A decision tree classifier is one of the most widely used supervised learning methods used for data exploration. It is easy to interpret and can be re-represented as if-then-else rules (Mladenic D., 2003; Fayyad U., 1996; Jiawei H. and Kamber M., 2006; Roiger R.J. and Geatz M.W., 2003 and Tan P.N. et al., 2006). If conditions then class, with a conjunction of features (attribute values) in the rule condition and a class label in the rule consequent (Mladenic D., 2003). In
approximates a function of constant does not require any prior knowledge of data distribution (Berry M.W. and Browne M., 2006).

A decision tree is a tree-like structure used for classification, decision theory, clustering and prediction functions. It describes rule for dividing training data into groups based on the regularities in the data. A decision tree can be used for categorical and continuous response variables. When the response variables are continuous, the decision tree is often referred to as a regression tree. If the response variables are categorical, it is called a classification tree. However, the same concepts apply to both types of trees.

Decision trees are widely used in computer science for data structures, in medical sciences for diagnosis, in botany for classification, in psychology for decision theory, in economy analysis for evaluating investment alternative and anomaly detection. The trees may differ in how they are created. For example, in some cases the trees are created top to bottom, while in other cases they are created from left to right. Decision tree have been described as universal approximates, because they are map linear and nonlinear relationships. However, they do not require as much training data as other universal approximates, such as neural networks.

2.25 (a) Binary Tree 2.25 (b) Nonbinary Tree

Figure 2.25 Types of decision tree (Jiawei H. and Kamber M., 2006)

The tree has three type of nodes as shown in Figure 2.25:

- A **root node** that has no incoming edges and zero or more outgoing edges.
• **Internal nodes**, each of which has exactly one incoming edge and two or more outgoing edges.

• **Leaf or terminal nodes**, each of which has exactly one incoming edge and no outgoing edges.

A decision tree consists of a root and internal nodes. The root and the internal nodes labeled with questions in order to find a solution to the problem under consideration. The root node is the first state of a decision tree. This node is assigned to all of examples from the training data. A decision tree is binary if each node is split into two parts. Nonbinary (multi-branch) if each node is split into three or more parts (Jiawei H. and Kamber M., 2006) as shown in Figure 2.25(b).

In decision tree, each leaf node is assigned a class label. The non-terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics (Tan P.N., 2006).

If an internal node can not be split further, it becomes a terminal node. The paths to each internal or terminal node are mutually exclusive, that is no more than one group can possibly be chosen. The process is repeated for each of internal nodes until a completely discriminating tree is obtain or the terminal node is reached.

### 2.7.1 Decision Tree Algorithm

A decision tree model consists of two parts: creating the tree and applying the tree to database. Decision trees use several different algorithms. The most widely-used algorithms by computer scientists are ID3, C4.5 and C5.0 (Berry M.W. and Browne M., 2006 and Tan P.N., 2006). This algorithm helps decision trees gain credibility and acceptance in the statistics community. Each algorithm employs different mathematical process to determine how to group and rank variables.
The original idea of constructing a decision tree of branch by Hoveland and Hunt. The skeleton of Hunt’s methods for constructing a decision tree from a set $T$ of training cases is as follows (Berry M.W. and Browne M., 2006):

Let the classes $\{C_1, C_2, \ldots, C_n\}$. There are three possibilities:

(i) $T$ contains one or more cases, but all belonging to a single class $C_j$. The decision tree for $T$ is a leaf identifying class $C_j$.

(ii) $T$ contains no cases. The decision tree is a leaf in this case, but class to be associated with the leaf must be determined from sources other than $T$.

(iii) $T$ contains cases that belong to a mixture of classes. $T$ is partitioned into subsets $T_1$, $T_2$, $\ldots$, $T_k$, where $T_i$ contains all cases in $T$ that have outcome $O_i$ of the chosen test. The decision tree for $T$ consists of a decision node identifying the test, and one branch for each possible outcome. This process is applied recursively to each subset of training cases, that the $i$-th branch leads to the decision tree constructed from the subset $T_i$ of the training cases.

Generally, a decision tree algorithm is most appropriate for the third case. In this case, the decision tree algorithm can be stated as follows (Berry M.W. and Browne M., 2006 and Tan P.N. et al., 2006):

- From the training data set, identify a target variable and a set of input variables.

- Examine each input variable one at a time:
  - To create two or more groupings of the values of the input variables, and measure how similar items are within each group and how different items are between groups.
  - Select the grouping that maximizes similarity within groupings and differences between groupings.

- Once the groupings have been calculated for each input variable, select the single input variable that maximizes similarity within groupings and differences between groupings.
This process is repeated in each group that contains a percentage of information in the original data. The process is not terminated until all divisible groups have been divided (Berry M.W. and Browne M., 2006).

### 2.7.2 Motivation of Decision Tree

Decision tree are one of fundamental techniques used in data mining (Berry M.W. and Browne M., 2006). Decision tree is tree-like structures used for classification. Decision trees are easily interpretable and intuitive for humans. They well suited for high-dimensional applications. Decision trees are fast and usually produce high-quality solutions. Decision tree objectives are consistent with the goals of data mining and knowledge discovery.

Decision trees are popular for partitioning data and identifying local structures in small and large databases, which are mostly concern with the discovery of classificatory properties of data tables. Decision tree models have two objectives: producing an accurate classifier, and understanding the predictive structure of the problem. The first goal deal with the accuracy of decision tree classification and the second goal aims at developing understandable patterns that can be interpreted as interesting knowledge (Berry M.W. and Browne M., 2006).

The representation of acquired knowledge in tree form is intuitive and generally easy to assimilated by humans. The learning and classification step of decision tree induction are simple and fast. In general, decision tree classifier to have good accuracy (Jiawei H. and Kamber M., 2006).

### 2.8 CONCLUSION

This chapter reviews the concept of NIPS based on NIDS and ID3 algorithm decision tree classifier. Intrusion detection is the act of detecting actions that attempt to compromise the confidentiality, integrity or availability of a resource. There are 4 types of intruder include DoS, probe, U2R and R2L. Each of intruder has specific work.
Intruders observed by looking at the log files. A system that performs automated intrusion detection called IDS. Intrusion should be denied or not allowed to entry in the LAN with reason for security. The intrusion prevention in computer networks is described by a firewall. Firewall can be allow or deny access network packets based on characteristics, such as IP address, port, protocol and others. Firewall can not define a network packet as intrusion. Therefore it is required a method for define whether a network packet as intrusion or not automatically. Decision Tree conduct a learning process from previous network traffics. Decision tree is a tree-like structure used for classification, where each internal node indicates a test on the attribute, each branch represents the test results, and the leaf nodes represent classes or class distributions. Decision tree can to convert to classification rules. These rules can define whether a network packet as a intrusion. These rules are implemented into firewall as intrusion prevention.
CHAPTER 3

METHODOLOGY

3.1 PROPOSED FRAMEWORK

Decision tree classifier to generate rules of intrusion signature patterns where these rules as NIPS that can determine a network packet is intrusion or a normal. Algorithm is used Quinlan’s (Mladenic D., 2003; Jiawei H. and Kamber M., 2006; Benelbahri M.A. and Bouhoula, 2007; Berry M.W. and Browne M., 2006 and Tan P.N. et al. 2006) ID3 to construct decision tree. Network traffic logs describe the human behavior which is the intrusion activities or the normal activities. Every network packet in network traffic logs extracted into five attributes as data set for data training of decision tree. The five attributes are source IP address, destination IP address, source port, destination port and protocol. These attributes are most important element in network packet, because these attributes can represent a network packet to denied or dropped that be implemented into firewall policy rules (Predrag Pale T.K., 2007). The results of decision tree training will get rules. These rules can define a network traffic as intrusion, then implemented into firewall rules as prevention. That is a simplified description of a complex entity or process which in this research called framework. The framework in this research using the NIDS and ID3 algorithm of decision tree classifier as shown in Figure 3.1.

There are several reasons for this research using the ID3 algorithm for construct decision tree classifier of data mining:

- Every packet network have object and clearly to identify, such as source IP address, destination IP address, source port, destination port and protocol. Object also called attribute or characteristics.
• Training data for class label attribute is discrete and not numeric. There are only two categories of occurrence in network packet which is either intrusion or not. This can be categorized as a class for decision tree classifier.

• Decision tree classifier using ID3 algorithm generate rules. The rules can determine a intrusion network packet automatically, that is called NIDS. The rules can be implemented into firewall, because they have the same attributes.

**Figure 3.1** Propose framework of NIPS based on NIDS and ID3 algorithm decision tree classifier

3.2 Network Traffic Logs

Log files record all the events on the machine. This files help administrator to monitor various problems such as bugs, network traffics and damage of the machine.
This research, there is a machine as router to record the network traffics. Router is a device that serves to forward packets from one network to another network, so that hosts on a network can communicate with hosts on other networks. Every network packet incoming and outgoing recorded this machine in log files. Illustration network traffic logs is shown in Figure 3.2.

![Network Traffic Logs](image)

**Figure 3.2** Network traffic logs in computer network

### 3.3 Determine Intrusion or Normal Network Packet using *Snort* NIDS

Determining occurrence of intrusion or normal at network traffic logs can be conducted with two ways:

- See manually by perceiving activities network traffic through log. Follow the example software application of logs files like *syslog*, *syslog_ng*, *tcpdump* and others. Pattern found to see intrusion through log seen modestly, for example there are some times trying to conduct and login of password failed, trying port scan, abundant ping, delivery of abundant package by repeat.

- Using software to determine intrusion activities or normal activities, for example *snort* software.
**Snort** is an open source network intrusion detection system and can be downloaded for free on the official website http://www.snort.org. There are millions of users downloading and more than 300,000 registered users. Snort has become the de facto standard for NIDS (Roozbahani A.R. and Rikhtechi L. , 2010).

**Snort** application is a combination of network sniffers and log files are applied to analyze the network traffics (Beheshti, M. et al., 2008). **Snort** is the most commonly used signature-based intrusion detection system, capable of performing real time traffic analysis and packet logging on IP networks. **Snort** depends on a pattern matching in **snort** rules and **snort** can only detect attacks known beforehand (Kang H. and Zhang J, 2009).

Snort rules by default placed in the folder /etc/snort/rules/. Figure 3.3 is a fragment of the contents **dos.rules** file, one of the **snort** file rules.

```plaintext

```

**Figure 3.3 Fragment of snort rules in a file dos.rules**

To edit the rules required a deep knowledge of the protocol, payload intrusions etc. **Snort** application has provided a lot of rules by default and can be added the new rules by downloading periodically at the **snort** official site http://www.snort.org/snort-rules/#rules.

This research using snort software as NIDS in Linux OS. **Snort** can determine a network packet is intrusion or normal. Snort also produces log files that can be analyzed
by admin. All of network traffic is saved by the log files in a folder `/var/log/snort/` and save the log files that intrusion in a folder `/var/log/snort/alert`.

### 3.4 Retrieving Data Set as Data Training

Network packet consists of the source IP, destination IP, source port, destination port, protocol and others. Intruder performs intrusion with a variety of ways such as seeing the open ports, OS used, the type and version of software application and others. Intruders performs intrusion in step by step process to get information. It causes intruder left many traces in log files. Meanwhile, one IP public represents many other computers to use by translating IP private. This is called Network Address Translation (NAT) (Suehring S. and Ziegler R.L., 2006). NAT was called masquerading. It enables many user do normal activities using source IP address same. Therefore to take normal activities as data training based on source IP address.

This research the way to take intrusion activities and normal activities for data training are:

1. For intrusion activities, capture all the network packet did an intrusion.
2. For normal activities, compare every source IP address of network packet normal one by one with network packet intrusion. If has same source IP address then take a network packet normal as data training.

If all network traffic as training data is the intrusion, decision tree is not useful as a classification because all the events are pure as the intrusion. Otherwise if all the network traffic as data training is normal. The function of decision tree is to generate rules intrusion and normal character that can classify the new network traffic is intrusion or normal despite having the same source IP address.

### 3.5 Extract Log Files

For data training, extract every log files intrusion and normal become five parameters as attributes and categorize 'Yes' or 'No' from intrusion as shown in Table
3.1. The attributes are IP address source, IP address destination, port source, port destination and protocol. For example, extract log files is shown in Figure 3.4

![Figure 3.4 Extract log files of network traffics](image)

For easier, dataset results from the extract to data training is formed into a table. Put the value attribute for each column plus one column as a class worth 'Yes' or 'No' of intrusion. Table 3.1 shows the results of the extract formed into a table.

**Table 3.1 Extract Log files into table form**

<table>
<thead>
<tr>
<th>No</th>
<th>Source IP</th>
<th>Dest IP</th>
<th>Source Port</th>
<th>Dest Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.1.13</td>
<td>10.10.1.3</td>
<td>1180</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>....</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
3.6 ID3 Algorithm

ID3 builds a decision tree from a fix set of examples. The resulting tree is used to classify future samples. This research has five attributes and belongs to a class ‘Yes’ or ‘No’ of intrusive. The decision node is an attribute test each branch (to another decision tree) being a possible value of the attribute. ID3 uses information going to help it decide which attribute goes into a decision node.

The calculation for information gain is the most difficult part of this algorithm. ID3 performs a search whereby the search states are decision trees and the operator involves adding a node to an existing tree. It uses information gain to measure the attribute to put in each node, and performs a greedy search using this measure of worth. The ID3 algorithm that implemented in this study goes as follows:

Given a set of examples, $S$, categorized in categories $c_i$, then:

1. Choose the root node to be the attribute, $A$, which scores the highest for information gain relative to $S$.
2. For each value $v$ that $A$ can possibly take, draw a branch from the node.
3. For each branch from $A$ corresponding to value $v$, calculate $S_v$. Then:
   • If $S_v$ is empty, choose the category $c_{\text{default}}$ which contains the most examples from $S$, and put this as the leaf node category which ends that branch.
   • If $S_v$ contains only examples from a category $c$, then put $c$ as the leaf node category which ends that branch.
   • Otherwise, remove $A$ from the set of attributes which can be put into nodes. Then put a new node in the decision tree, where the new attribute being tested in the node is the one which scores highest for information gain to $S_v$. This new node starts the cycle again (from 2), with $S$ replaced by $S_v$ in the calculations and the tree gets built iteratively like this.

The algorithm terminates either when all the attributes have been exhausted, or the decision tree perfectly classifies the examples. The diagram to explain the ID3 algorithm is shown in Figure 3.5.
3.6.1 Entropy

Given a binary categorization, $C$, and a set of examples, $S$, for which the proportion of examples categorized as positive by $C$ is $p_+$ and the proportion of examples categorized as negative by $C$ is $p_-$, then the entropy of $S$ is:

$$E(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-) \quad \text{.................. (3.1)}$$

Given a categorization, $C$ into categories $c_1, \ldots, c_n$, and a set of examples, $S$, for which the proportion of examples in $c_i$ is $p_i$, then the entropy of $S$ is:

$$E(S) = \sum_{i=1}^{n} -p_i \log_2(p_i) \quad \text{............................................... (3.2)}$$

If $p$ gets close to zero, $\log(p)$ becomes a big negative number. Entropy calculates the disorder in the data; if entropy is low score it means good, as it reflects desire to reward categories. Similarly, if $p$ gets close to 1, then the $\log_2(p)$ part gets very close to zero, so the overall value gets close to zero (Dunham and Margareth H., 2002; Mladenic...
3.6.2 Information Gain

The following measure calculates entropy value for a given attribute, \( A \), with respect to a set of examples, \( S \). Note that the values of attribute \( A \) will range over a set of possibilities which call \( \text{Values}(A) \), and that, for a particular value from that set, \( v \), write \( S_v \) for the set of examples which have value \( v \) for attribute \( A \), then entropy \( S \) of \( A \) is:

\[
E(S, A) = \sum_{v \in \text{Value}(A)} \frac{|S_v|}{|S|} E(S_v) \tag{3.3}
\]

The information gain of attribute \( A \), relative to a collection of examples, \( S \), is calculated as:

\[
\text{Gain}(S, A) = E(S) - E(S, A) = E(S) - \sum_{v \in \text{Value}(A)} \frac{|S_v|}{|S|} E(S_v) \tag{3.4}
\]

3.7 Rule Extraction IF-THEN

Calculation having taken steps by using ID3 algorithm hence formed by decision tree example as shown in Figure 3.6.
Figure 3.6 Decision tree of network traffics

Figure 3.7 Pruning decision tree network traffic for NIDS
Decision tree classifier made moderate with only taking pattern which occurrence intrusive 'Yes' at final node and this describe rules of intrusion signature pattern. Decision tree can simplified by pruning all connections are assumed to be normal and not classified as intrusion. Figure 3.7 shows pruning decision tree network traffics.

The Decision trees can become interpretation rule-based classifier by extracting IF-THEN rules (Mladenic D., 2003; Jiawei H. and Kamber M., 2006; Roiger R.J. and Geatz M.W., 2003 and Tan P.N. et al., 2006). In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand (Jiawei H. and Kamber M., 2006).

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically AND form the rule antecedent (“IF” part). The leaf node holds the class prediction, forming the rule consequent (“THEN” part) (Jiawei H. and Kamber M., 2006).

This is example extracting classification rules from a decision tree. The decision tree is shown in Figure 3.7 can converted to classification IF-THEN rules by tracing the root node to each leaf node in the tree is shown in Figure 3.8.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>IF (Source IP = 122.306.13.100) THEN Intrusion = Yes</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>IF (Source IP = 203.130.14.20) AND (Protocol = TCP) AND (Dest Port = 80) THEN Intrusion = Yes</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>IF (Source IP = 206.145.206.4) AND (Dest IP = 10.10.1.2) THEN Intrusion = Yes</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>IF (Source IP = 206.145.206.4) AND (Dest IP = 10.10.1.5) AND (Protocol = TCP) AND (Source Port = 1330) THEN Intrusion = Yes</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>IF (Source IP = 206.145.206.4) AND (Dest IP = 10.10.1.5) AND (Protocol = TCP) AND (Source Port = 1630) THEN Intrusion = Yes</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.8 The decision tree converted to classification IF-THEN rules
3.8 Rules Implemented into Firewall Rules

Decision tree has to generate rules. If there is network packet match with one of rules, the network packet is intrusion. For security, intrusion network packet will be dropped and not forward into LAN. These rules in Figure 3.8 can be implemented in firewall rules as shown in Figure 3.9.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>-A FORWARD -s 122.306.13.100 -j DROP</td>
</tr>
<tr>
<td>R2</td>
<td>-A FORWARD -s 203.130.14.20 -p tcp --dport 80 -j DROP</td>
</tr>
<tr>
<td>R3</td>
<td>-A FORWARD -s 206.145.206.4 -d 10.10.1.2 -j DROP</td>
</tr>
<tr>
<td>R4</td>
<td>-A FORWARD -s 206.145.206.4 -d 10.10.1.5 –p tcp --sport 1330 -j DROP</td>
</tr>
<tr>
<td>R5</td>
<td>-A FORWARD -s 206.145.206.4 -d 10.10.1.5 –p tcp --sport 1630 -j DROP</td>
</tr>
</tbody>
</table>

**Figure 3.9** Rules to be implemented into firewall rules

Firewall policy rules as shown in Figure 3.9 describe that every network packet match rules will DROP. This action prevention of intrusion network packet.

3.9 CONCLUSION

This chapter explains propose NIPS based on NIDS and ID3 algorithm decision tree classifier. The network traffic logs as data training have two categories that are intrusion or normal. Network traffics extracts five attributes include source IP, destination IP, source port, destination port and protocol. Decision tree is construct by using ID3 algorithm. Decision tree generate rules of intrusion, where rules can determine automatically network packet either intrusion or normal. Rules of intrusion was implemented into firewall as prevention action.
CHAPTER 4

IMPLEMENTATION

4.1 NIPS BASED ON NIDS AND ID3 ALGORITHM DECISION TREE CLASSIFIER ENVIRONMENT

Internet is made up of several computer network connections. This connection then expanded and spread all over the world. This research takes only a few clients and servers that can represent a large computer network like the Internet. Client request to the server and some tried intrusion to the server. The most important in this research obtain data training that is intrusion network packets in the network traffics and does not depend on the size of a computer network. Machines as routers and firewalls that important to capture network traffic in the log files and take action for network packet. This machine can be implemented into a large computer network where the log files as data training generated reflects the network topology used.

The implementation is built 3 Local Area Network (LAN) as client is connected to the LAN as the server computer. There is 3 computer for servers in a LAN. Each of LAN connected through a router. Each computer server has application server same that is SSH server, Apache web server, MySQL database server, Sendmail Mail Server and Squirellmail Webmail. Squirellmail is Webmail to provide email service through a URL or web browser. Machine computer for the router and firewall applications have a Dynamic Host Configuration Protocol (DHCP) Server, Berkeley Internet Name Domain (BIND) Name Server and Snort software for network traffic logs and NIDS. All network traffic logs will be recorded in this machine as router and firewall using Snort software. Topology network and all information shown in Figure 4.1.
Figure 4.1 Network topology
4.1.1 Setting Server

A computer server provide many services to the client. In addition, server is to detect intrusive from the client too. Each server uses Slackware Linux 12.1 Operating System kernel 2.6.27.1. The same hardware specification for all servers in LAN is shown in Table 4.1.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>AMD Duron 1200+</td>
</tr>
<tr>
<td>Mainboard</td>
<td>ECS K7SEM</td>
</tr>
<tr>
<td>Memory</td>
<td>2x@256 MB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>80 GB</td>
</tr>
<tr>
<td>VGA Card</td>
<td>AGP NVDIA GeForce2 MX 400</td>
</tr>
<tr>
<td>Network Card</td>
<td>Realtek RTL8139</td>
</tr>
</tbody>
</table>

Computer as the server is in LAN 10.10.1.0/29. There are 3 servers. IP address and hostname of each computer is shown in Table 4.2.

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.10.1.2</td>
<td>server1</td>
</tr>
<tr>
<td>10.10.1.3</td>
<td>server2</td>
</tr>
<tr>
<td>10.10.1.5</td>
<td>server3</td>
</tr>
</tbody>
</table>

Setting IP address and gateway have done by using the command `ifconfig` and `route` on the Linux command line interface. Gateway is the nearest route to the LAN. Setting IP address and gateway is shown in Figure 4.2.
There are several ways to setting IP address:

- Use command `ifconfig`. For easily command setting IP address is written in the `rc.local` file. This file in the directory `/etc/rc.d/rc.local`. This is a local startup command, where every time booting all command in the `rc.local` file is executed.

- Setting static IP address by tools, use command `netconfig` then there is dialog box.

- Setting static IP address use script in `rc.inet1.conf` file. In Slackware that is file in folder `/etc/rc.d/rc.inet1.conf`.

Each computer has same application server. Application server is shown in Table 4.3.

<table>
<thead>
<tr>
<th>Application Server</th>
<th>Software</th>
<th>Port Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH Server</td>
<td>ssh</td>
<td>22</td>
</tr>
<tr>
<td>Web Server</td>
<td>apache</td>
<td>80</td>
</tr>
<tr>
<td>Database</td>
<td>mysql</td>
<td>3306</td>
</tr>
<tr>
<td>Mail Server</td>
<td>sendmail</td>
<td>143, 110, 25</td>
</tr>
<tr>
<td>Webmail</td>
<td>squirrelmail</td>
<td>80, 143, 110, 25</td>
</tr>
</tbody>
</table>
Application server will run through the port as a service to the client. This research, these ports are destination ports, one of the attribute data training. These ports has some value as shown in Table 4.3. It is a target for the intruder to access system.

4.1.2 Setting Router and Firewall

Router is a computer or machine that forward packets from one or more than one LAN to another LAN. Router has a broader function than the gateway. Router connect between gateway on computer network. Figure 4.3 shows the setting IP address computer and router.

![Figure 4.3 Setting IP address and router](image)

Enter command line is shown in Figure 4.3 above in file `/etc/rc.d/rc.local` as the local startup command.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Core 2 Dou 2.0 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>1GB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>250 GB</td>
</tr>
<tr>
<td>Network Card</td>
<td>Realtek RTL 8111/8168B</td>
</tr>
</tbody>
</table>
Computer specification for the router and firewall is shown Table 4.4. In the implementation computer as router and firewall also as a DHCP Server, Name Server, Firewall and network traffic logs. DHCP server functions to provide automatically IP address, netmask, gateway and Domain Name System (DNS) for client. Name Server will be a service that translates IP address to a system for name as well as the name of the system to the IP address. The name system in this hierarchy called the DNS. Software application at computer router and firewall is shown in Table 4.5.

**Table 4.5 Software applications in computer as router and firewall**

<table>
<thead>
<tr>
<th>Software Aplication</th>
<th>Version</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHCP Server</td>
<td>dhcp-3.0.3</td>
<td>67</td>
</tr>
<tr>
<td>Name Server</td>
<td>bind-9.2.3</td>
<td>53</td>
</tr>
<tr>
<td>Firewall</td>
<td>iptables-1.2.10</td>
<td>-</td>
</tr>
<tr>
<td>Log Files</td>
<td>snort-2.8.3.1 (/var/log/snort)</td>
<td>-</td>
</tr>
</tbody>
</table>

DHCP server used for one LAN, because computer has only one network card that is 'eth0'. The configuration of DHCP is shown Figure 4.4.
Local DNS makes the computer host easily known in the LAN. In this implementation the domain name is 'research.com'. The configuration of DNS are shown in Figure 4.5, Figure 4.6 and Figure 4.7.

```
options {
    directory "/var/named";
};

zone "." IN {
    type hint;
    file "caching-example/named.root";
};

zone "localhost" IN {
    type master;
    file "caching-example/localhost.zone";
    allow-update { none; }
};

zone "0.0.127.in-addr.arpa" IN {
    type master;
    file "caching-example/named.local";
    allow-update { none; }
};

zone "research.com" IN {
    type master;
    file "caching-example/db.research.com";
    allow-update { none; }
};

zone "1.10.10.in-addr.arpa" IN {
    type master;
    file "caching-example/db.10.10.1";
    allow-update { none; }
};
```

Figure 4.5 Configure DNS server zone at /etc/named.conf
In Figure 4.7, IP address 10.10.1.2 mapped to a domain name server1.research.com and for name of another server computer is shown in Table 4.6.
Table 4.6 Local DNS

<table>
<thead>
<tr>
<th>IP Address</th>
<th>DNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.10.1.6</td>
<td>research.com</td>
</tr>
<tr>
<td>10.10.1.2</td>
<td>server1.research.com</td>
</tr>
<tr>
<td>10.10.1.3</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>10.10.1.5</td>
<td>server3.research.com</td>
</tr>
</tbody>
</table>

Client computer must to enter which computer is DNS server. That's means DNS server will translate hostname to IP address and contrary IP address to hostname request from client. Settings DNS server in computer client at file /etc/resolv.conf. DNS server translate IP address to hostname and name to IP address are shown in Figure 4.8.

```
rabill@client2:~$ nslookup 10.10.1.2
Server: 10.10.1.6
Address: 10.10.1.6#53

2.1.10.10.in-addr.arpa name = server1.research.com.

rabill@client2:~$ nslookup server1.research.com
Server: 10.10.1.6
Address: 10.10.1.6#53

Name: server1.research.com
Address: 10.10.1.2

rabill@client2:~$ nslookup 10.10.1.6
Server: 10.10.1.6
Address: 10.10.1.6#53

6.1.10.10.in-addr.arpa name = research.com.

rabill@client2:~$ nslookup research.com
Server: 10.10.1.6
Address: 10.10.1.6#53

Name: research.com
Address: 10.10.1.6
```

**Figure 4.8** Translate IP address to DNS and on the contrary

In implementation, *iptables* has been used to packet filter firewall. It works at the Network Level of the OSI model. In a firewall packet filtering phase, each network
packet compared to a set of criteria. Depending on the packet and the criteria, the firewall can drop the packet, forward it or send a message to the client.

Without case of generality, this implementation uses concept that accept everything by default and drop network packets match with rules in firewall. The reason to find which one is intrusive network packet, that will then be dropped at the firewall. Therefore it accepts all the default network packet and drop the network packet intrusive.

4.2 NETWORK TRAFFIC LOGS

All event in the Linux OS recorded in log files. In general log files on Linux OS in the directory /var/log/. For example are /var/log/syslog, /var/log/messages, /var/log/snort/alert and others.

Snort software has been used in this implementation to see network traffics logs. Snort is the open source to run real-time analysis and packet logging on IP network. This software can detect various attacks and infiltration, such as buffer overflow, stealth port scans, CGI attacks, efforts of fingerprinting and others.

Snort has 3 main purposes, which are:

- **Sniffer mode**, to see the package through.
- **Logger mode packet**, to record all the packets through the network for later analysis.
- **Intrusion and Detection mode**, to detect attacks made through a computer network. To use this mode in IDS requires to setup a variety of rules that will distinguish normal packet with intrusion packet.

In this implementation, client computers to attack or intrusion to server computer by using some software for hackers such as nmap, nikto, tcpdump and others. For example, the attempt for intrusion is shown in Figure 4.9, 4.10 and 4.11.
Figure 4.9 *nmap* software to scan port host 10.10.1.2

```
root@aminzm:/home/rabill# nmap -sS 10.10.1.2

Starting Nmap 4.76 ( http://nmap.org ) at 2009-11-10 08:29 WIT
Interesting ports on server1.research.com (10.10.1.2):
Not shown: 991 closed ports

PORT     STATE SERVICE
21/tcp    open     ftp
22/tcp    open     ssh
25/tcp    open     smtp
37/tcp    open     time
80/tcp    open     http
110/tcp   open     pop3
113/tcp   open     auth
143/tcp   open     imap
587/tcp   open     submission

Nmap done: 1 IP address (1 host up) scanned in 0.39 seconds
```

Figure 4.10 *nikto* software to get information web server at host 10.10.1.2

```
root@aminzm:/home/rabill# nikto -h 10.10.1.2
+ Target IP: 10.10.1.2
+ Target Hostname: server1.research.com
+ Target Port: 80
+ Server: Apache/2.2.8 (Unix) DAV/2 PHP/5.2.5
  + Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
  + OSVDB-877: HTTP method ( "Allow" header ): "TRACE" is typically only used for debugging and should be disabled. This message does not mean it is vulnerable to XST.
  + Apache/2.2.8 appears to be outdated ( current is at least Apache/2.2.9). Apache 1.3.9 and 2.0.61 are also current.
  + PHP/5.2.5 appears to be outdated ( current is at least 5.2.6RC4).
  + OSVDB-3092: GET /data/ : This might be interesting...
  + OSVDB-3092: GET /misc/ : This might be interesting...
  + OSVDB-3093: GET /mail/src/rend_body.php : This might be interesting... has been seen in web logs from an unknown scanner.
  + OSVDB-3268: GET /images/ : Directory indexing is enabled: /images
  + OSVDB-3268: GET /style/ : Directory indexing is enabled: /style
  + 3577 items checked: 10 item(s) reported on remote host
  + 1 host(s) tested
```

root@aminzm:/>
**Figure 4.11** Try to connect host 10.10.1.2 using password root failed through SSH

In *snort* software an intrusive attempt was logged on directory files in `/var/log/snort/alert` is shown on Section A of Appendix A and all of network traffics that intrusion and normal in the directory `/var/log/snort/snort.log.*` shown section B of Appendix A.

Log files intrusion activities and normal activities is shown in appendix A, retrieving data set as data training refer to Section 3.4:

1. For the data training is intrusion, take all the data intrusion on the log files in the folder `/var/log/snort/alert`.
2. For the data training is normal, each network packet on log files in the folder `/var/log/snort/snort.log.*` will be compared with network packet is intrusion at no.1 above. If have the same source IP address, then take the network packet network is normal as training data.

Therefore there is 60 data set as data training then extracted in tabular form shown in the Table 4.7.
Table 4.7 Data set of network traffics for data training

<table>
<thead>
<tr>
<th>No.</th>
<th>Source IP</th>
<th>Source Port</th>
<th>Destination IP</th>
<th>Destination Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.2.25</td>
<td>1142</td>
<td>10.10.1.2</td>
<td>161</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>192.168.2.25</td>
<td>1143</td>
<td>10.10.1.2</td>
<td>162</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>192.168.2.25</td>
<td>1179</td>
<td>10.10.1.2</td>
<td>110</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>192.168.2.25</td>
<td>1175</td>
<td>10.10.1.2</td>
<td>25</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>192.168.2.25</td>
<td>1130</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>192.168.2.25</td>
<td>1131</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>192.168.2.25</td>
<td>1345</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.25</td>
<td>1347</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.25</td>
<td>1348</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>192.168.2.25</td>
<td>1351</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>192.168.2.25</td>
<td>1352</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>192.168.2.25</td>
<td>1356</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>192.168.2.25</td>
<td>1354</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>192.168.2.25</td>
<td>1392</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>15</td>
<td>192.168.2.25</td>
<td>1394</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>192.168.2.25</td>
<td>1396</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>17</td>
<td>192.168.2.25</td>
<td>1400</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>18</td>
<td>192.168.2.25</td>
<td>1401</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>19</td>
<td>192.168.2.25</td>
<td>1405</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>20</td>
<td>192.168.2.25</td>
<td>1458</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>21</td>
<td>192.168.2.25</td>
<td>1465</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>22</td>
<td>192.168.2.25</td>
<td>1463</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>23</td>
<td>192.168.2.25</td>
<td>1466</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>24</td>
<td>192.168.2.25</td>
<td>1398</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>25</td>
<td>192.168.2.25</td>
<td>1392</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>26</td>
<td>192.168.2.25</td>
<td>1396</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>27</td>
<td>192.168.2.25</td>
<td>1394</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>28</td>
<td>192.168.2.25</td>
<td>1402</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>29</td>
<td>192.168.2.25</td>
<td>1409</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>30</td>
<td>192.168.2.25</td>
<td>1411</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>31</td>
<td>192.168.2.25</td>
<td>1413</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>32</td>
<td>192.168.2.25</td>
<td>1414</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>33</td>
<td>192.168.2.25</td>
<td>1184</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>34</td>
<td>192.168.2.25</td>
<td>1185</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>35</td>
<td>192.168.2.25</td>
<td>1384</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>36</td>
<td>192.168.2.25</td>
<td>1386</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>37</td>
<td>192.168.2.25</td>
<td>1391</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>38</td>
<td>192.168.2.25</td>
<td>1393</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>39</td>
<td>192.168.2.25</td>
<td>1409</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>40</td>
<td>192.168.2.25</td>
<td>1411</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>41</td>
<td>192.168.2.25</td>
<td>1413</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>42</td>
<td>192.168.2.13</td>
<td>1173</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>43</td>
<td>192.168.2.13</td>
<td>1175</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>44</td>
<td>192.168.2.13</td>
<td>1179</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>45</td>
<td>192.168.2.13</td>
<td>1180</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>46</td>
<td>192.168.2.13</td>
<td>1158</td>
<td>10.10.1.2</td>
<td>25</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>47</td>
<td>192.168.2.13</td>
<td>1203</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>48</td>
<td>192.168.2.13</td>
<td>1204</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>49</td>
<td>192.168.2.13</td>
<td>1220</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>50</td>
<td>192.168.2.13</td>
<td>1419</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>51</td>
<td>192.168.2.13</td>
<td>1423</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>52</td>
<td>192.168.2.16</td>
<td>10.10.1.2</td>
<td>161</td>
<td>162</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>53</td>
<td>192.168.2.25</td>
<td>34592</td>
<td>10.10.1.2</td>
<td>3306</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>54</td>
<td>192.168.2.16</td>
<td>34592</td>
<td>10.10.1.2</td>
<td>3306</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>55</td>
<td>192.168.2.16</td>
<td>53814</td>
<td>10.10.1.2</td>
<td>53815</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>56</td>
<td>192.168.2.16</td>
<td>53815</td>
<td>10.10.1.2</td>
<td>53814</td>
<td>TCP</td>
<td>Yes</td>
</tr>
</tbody>
</table>
4.3 CONSTRUCT DECISION TREE USING ID3 ALGORITHM

Based on the network traffics shown in Table 4.7, there are two conditions that must be followed in building a decision tree:

1. The value of the source port attribute so many, there 60 value in source port attribute where the value is different each other. This causes the value of entropy of the source port is 0 based on Equation (3.3) in section 3.4.2:

\[
E(S, A) = \sum_{v \in \text{Value}(A)} \frac{|S_v|}{|S|} E(S_v)
\]

\[
E(\text{Intrusion, Source Port}) = \frac{1}{60} (-1/1 \log_2 1/1 - 0/1 \log_2 0/1) + \frac{1}{60} (-1/1 \log_2 1/1 - 0/1 \log_2 0/1) + \ldots + \frac{1}{60} (-1/1 \log_2 1/1 - 0/1 \log_2 0/1)
\]

\[
= 0 + 0 + 0 + \ldots + 0
\]

\[
= 0
\]

and value of gain is always high based on Equation (3.4) in section 3.4.2:

\[
\text{Gain}(S, A) = E(S) - E(S, A)
\]

\[
G(\text{Intrusion, Source IP}) = E(\text{Intrusion}) - E(\text{Intrusion, Source IP}) = E(\text{Intrusion}) - 0
\]

Consequently source port attribute always as root node. If the source port attribute as root node decision tree classification is not useful anymore, because under source port of the node is the all value of source port attribute. Therefore, the source port is ignored calculations to construct decision tree. Source port is used as the last node if all attribute has become nodes in the tree, then the last options is the source port attribute as a node.

2. Source port and destination port cannot implemented into firewall rules without protocol. Figure 4.12 show an example of rules is wrong.
Figure 4.12. The description of the wrong rules in firewall

Source Port and Destination Port need Protocol to create rules into firewall. Therefore, Protocol and Destination Port become one attribute for calculate Entropy and Gain. To construct decision tree Protocol attribute as node for first follow is Destination Port.

The first step to find attributes at the top of tree as root node from data set is shown in Table 4.7, there is 60 data sets where 42 intrusion and 18 normal.

![Figure 4.13 The set of 60 examples where 42 intrusion and 18 normal](image)

To Calculate entropy of Intrusion is based on Equation (3.1) in Section 3.4.1:

\[
E (S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)
\]

\[
E \text{ (Intrusion)} = -\frac{42}{60} \log_2\left(\frac{42}{60}\right) - \frac{18}{60} \log_2\left(\frac{18}{60}\right)
\]

\[
= 0.7 \log_2 0.7 - 0.3 \log_2 0.3
\]

\[
= 0.8813
\]

The next step to calculate for each of attributes.

In Table 4.7 there are four values from the Source IP. Every value of the Source IP is counted in order to know how many intrusion and not intrusion. To more easily

<table>
<thead>
<tr>
<th>R1</th>
<th>-A FORWARD --dport 22 -j DROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>-A FORWARD --sport 1890 -j DROP</td>
</tr>
<tr>
<td>R3</td>
<td>-A FORWARD --sport 1890 --dport 22 -j DROP</td>
</tr>
</tbody>
</table>
make in the form of tables. An example in Table 4.8 is described the value of the Source IP attribute.

**Table 4.8 Source IP**

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Intrusion</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>192.168.2.25</td>
<td>14</td>
<td>0</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>192.168.0.5</td>
<td>18</td>
<td>9</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>192.168.1.13</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>192.168.2.16</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td></td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

To Calculate entropy Intrusion of Source IP based on Equation (3.3) in Section 3.4.2.

\[
E(S, A) = \sum_{v \in \text{Value}(A)} \frac{|S_v|}{|S|} E(S_v)
\]

\[
E(\text{Intrusion, Source IP}) = \frac{14}{60} (-14/14 \log_2 14/60 - 0/14 \log_2 0/14) \\
+ \frac{27}{60} (-18/27 \log_2 18/27 - 9/27 \log_2 9/27) \\
+ \frac{15}{60} (-5/10 \log_2 5/10 - 5/10 \log_2 5/10) \\
+ \frac{9}{60} (-5/9 \log_2 5/9 - 4/9 \log_2 4/9)
\]

\[
= 0 + 0.4132 + 0.1666 + 0.1487 \\
= 0.7285
\]

To calculate gain Intrusion of Source IP based on Equation (3.4) in Section 3.4.2.

\[
\text{Gain}(S, A) = E(S) - E(S, A)
\]

\[
G(\text{Intrusion, Source IP}) = E(\text{Intrusion}) - E(\text{Intrusion, Source IP})
\]

\[
= 0.8813 - 0.7285 \\
= 0.1528
\]

To calculate entropy and gain Intrusion of Destination IP:

In Table 4.7 there are four values from the Destination IP. Every value of the Destination IP is counted how many intrusion and not intrusion. This Procedure was
conducted to define the value of every attribute. In Table 4.8 is described the value of the Source IP attribute.

Table 4.9 Destination IP

<table>
<thead>
<tr>
<th>Destination IP</th>
<th>Intrusion</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>10.10.1.2</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>10.10.1.3</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>10.10.1.5</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>60</strong></td>
<td></td>
</tr>
</tbody>
</table>

To Calculate entropy Intrusion of Destination IP based on Equation (3.3) in Section 3.4.2.

\[
E(S,A) = \sum_{v \in Value(A)} \frac{|S_v|}{|S|} E(S_v)
\]

\[
E(\text{Intrusion}, \text{Destination IP}) = \frac{22}{60} \left(-\frac{18}{22} \log_2 \frac{18}{22} - \frac{4}{22} \log_2 \frac{4}{22}\right) + \frac{18}{60} \left(-\frac{11}{18} \log_2 \frac{11}{18} - \frac{7}{18} \log_2 \frac{7}{18}\right) + \frac{20}{60} \left(-\frac{13}{20} \log_2 \frac{13}{20} - \frac{7}{20} \log_2 \frac{7}{20}\right)
\]

\[
= 0.25 + 0.2892 + 0.3114
\]

\[
= 0.8506
\]

To calculate gain Intrusion of Destination IP based on Equation (3.4) in Section 3.4.2.

\[
\text{Gain}(S,A) = E(S) - E(S,A)
\]

\[
\text{G}(\text{Intrusion}, \text{Destination IP}) = E(\text{Intrusion}) - E(\text{Intrusion}, \text{Destination IP})
\]

\[
\text{G}(\text{Intrusion}, \text{Destination IP}) = 0.8813 - 0.8506
\]

\[
= 0.0307
\]

To calculate entropy and gain Intrusion of Protocol and Destination Port attribute:


### Table 4.10 Protocol and Destination Port

<table>
<thead>
<tr>
<th>Protocol &amp; Destination Port</th>
<th>Intrusion</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Sum</td>
</tr>
<tr>
<td>ICMP</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>TCP</td>
<td>22</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>TCP</td>
<td>25</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>TCP</td>
<td>80</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>TCP</td>
<td>110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TCP</td>
<td>161</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>TCP</td>
<td>162</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>TCP</td>
<td>3360</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

| Sum | 60 |

To Calculate entropy Intrusion of Protocol and Destination Port based on Equation (3.3) in Section 3.4.2.

\[
E(S,A) = \sum_{v \in Value(A)} \frac{|S_v|}{|S|} E(S_v)
\]

\[
E(\text{Intrusion, Protocol & Dest Port}) = \frac{4}{60} (-4/4 \log_2 4/4 + 0/4 \log_2 0/4) + \frac{20}{60} (-17/20 \log_2 17/20 + 3/20 \log_2 3/20) + \frac{2}{60} (-2/2 \log_2 2/2 + 0/2 \log_2 0/2) + \frac{27}{60} (-12/27 \log_2 12/27 + 15/27 \log_2 15/27) + \frac{1}{60} (-1/1 \log_2 1/1 + 0/1 \log_2 0/1) + \frac{2}{60} (-2/2 \log_2 2/2 + 0/2 \log_2 0/2) + \frac{2}{60} (-2/2 \log_2 2/2 + 0/2 \log_2 0/2) + \frac{2}{60} (-2/2 \log_2 2/2 + 0/2 \log_2 0/2)
\]

\[
= 0 + 0.2033 + 0 + 0.4460 + 0 + 0 + 0 + 0 = 0.6493
\]

To calculate gain Intrusion of Protocol and Destination Port based on Equation (3.4) in Section 3.4.2.

\[
G(S,A) = E(S) - E(S,A)
\]

\[
G(\text{Intrusion, Protocol & Dest Port}) = E(\text{Intrusion}) - E(\text{Intrusion, Protocol & Dest Port})
\]

\[
G(\text{Intrusion, Protocol & Dest Port}) = 0.8813 - 0.6493 = 0.2320
\]
The highest Gain in Figure 4.14 is Protocol and Destination Port attribute which value is 0.2320. Therefore, Protocol and Destination Port attribute is used as root node which shown in Figure 4.15.

Separate the attribute protocol and destination port become two parts. The first attribute is Protocol that has the value is ICMP and TCP. Then follow the Destination Port attribute is shown in Figure 4.15. Now define the following attributes under the ICMP protocol.

To calculate Gain for each attribute use Protocol = ICMP:

These instances that use Protocol = ICMP as shown in Table 4.11.
Table 4.11 All data set using Protocol = ICMP

<table>
<thead>
<tr>
<th>No.</th>
<th>Source IP</th>
<th>Source Port</th>
<th>Destination IP</th>
<th>Destination Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.2.25</td>
<td>10.10.1.2</td>
<td>ICMP</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>192.168.0.5</td>
<td>10.10.1.2</td>
<td>ICMP</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>192.168.0.5</td>
<td>10.10.1.3</td>
<td>ICMP</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>192.168.2.16</td>
<td>10.10.1.2</td>
<td>ICMP</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.16 The set of 4 examples Protocol = ICMP are 4 intrusion and 0 normal

To calculate entropy of Protocol = ICMP based on Equation (3.1) in Section 3.4.1:

\[
E(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)
\]

\[
E(\text{ICMP}) = -\frac{4}{4} \log_2 \left( \frac{4}{4} \right) - \frac{0}{4} \log_2 \left( \frac{0}{4} \right)
\]

\[= 0\]

From the above calculation, \(E(\text{ICMP})\) is equals to 0 This shows that the incident was pure impurity is 'Yes' and can be described in decision tree as shown in Figure 4.17.

Figure 4.17 All of instance of ICMP Protocol is Intrusion = Yes
Next, to find and define the next attribute under the Protocol = TCP and Destination Port = 22.

To calculate Gain for each attributes using Protocol = TCP and Destination = 22 attribute. All of instances (see in Table 4.7) which using attribute Protocol = TCP and Destination Port = 22 is shown in Table 2.12.

Table 4.12 All data set using Protocol = TCP and Destination Port = 22

<table>
<thead>
<tr>
<th>No.</th>
<th>Source IP</th>
<th>Source Port</th>
<th>Destination IP</th>
<th>Destination Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.2.25</td>
<td>1130</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>192.168.2.25</td>
<td>1131</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>192.168.2.25</td>
<td>1345</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>192.168.2.25</td>
<td>1347</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>192.168.2.25</td>
<td>1348</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>192.168.2.25</td>
<td>1351</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>192.168.2.25</td>
<td>1352</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.25</td>
<td>1356</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.25</td>
<td>1354</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>192.168.0.5</td>
<td>1401</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>192.168.0.5</td>
<td>1405</td>
<td>10.10.1.2</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>192.168.0.5</td>
<td>1394</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>192.168.0.5</td>
<td>1398</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>192.168.0.5</td>
<td>1400</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>15</td>
<td>192.168.0.5</td>
<td>1409</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>192.168.0.5</td>
<td>1411</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>17</td>
<td>192.168.0.5</td>
<td>1413</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>18</td>
<td>192.168.1.13</td>
<td>1203</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>19</td>
<td>192.168.1.13</td>
<td>1204</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>20</td>
<td>192.168.1.13</td>
<td>1220</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
</tbody>
</table>

22
[17+, 3–]

Yes
17/20 = 0.85

No
3/20 = 0.15

Figure 4.18 The set of 20 examples Dest Port=22 are 17 intrusion and 3 normal
To calculate entropy of Destination Port = 22 based on Equation (3.1) in Section 3.4.1:

\[ E(S) = -\sum p_i \log_2(p_i) \]

\[ E(22) = -17/20 \log_2(17/20) - 3/20 \log_2(3/20) \]

\[ = 0.6098 \]

To calculate each entropy and Gain attributes are Source IP and Destination IP:

✓ Entropy and Gain Destination Port = 22 attribute of Source IP

**Table 4.13** Source IP using Protocol = TCP and Destination Port = 22

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Intrusion</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Sum</td>
</tr>
<tr>
<td>192.168.2.25</td>
<td>9</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>192.168.0.5</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>192.168.1.13</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Sum</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To calculate entropy Destination Port = 22 of source IP based on Equation (3.3) in Section 3.4.2.

\[ E(S_A) = \sum_{v \in \text{Value}(A)} \frac{|S_v|}{|S|} E(S_v) \]

\[ E(S_{22}, \text{Source IP}) = 9/20 (-9/9 \log_2(9/9) - 0/9 \log_2(0/9)) \]

\[ + 8/20 (-8/8 \log_2(8/8) - 0/8 \log_2(0/8)) \]

\[ + 3/20 (-0/3 \log_2(0/3) - 3/3 \log_2(3/0)) \]

\[ = 0 \]

To calculate gain of Source IP based on Equation (3.4) in Section 3.4.2.

\[ \text{Gain}(S_A) = E(S) - E(S_A) \]

\[ G(S_{22}, \text{Source IP}) = E(22) - E(S_{22}, \text{Source IP}) \]

\[ G(S_{22}, \text{Source IP}) = 0.6098 - 0 \]

\[ = 0.6098 \]
Entropy and Gain Destination Port = 22 of Destination IP

**Table 4.14** Destination IP using Destination Port = 22

<table>
<thead>
<tr>
<th>Destination IP</th>
<th>Intrusion</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Sum</td>
</tr>
<tr>
<td>10.10.1.2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>10.10.1.3</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>10.10.1.5</td>
<td>13</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Sum</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To calculate entropy Destination Port = 22 of Destination IP based on Equation (3.3) in Section 3.4.2.

\[
E(S, A) = \sum_{v \in Value(A)} \frac{|S_v|}{|S|} E(S_v)
\]

\[
E(S_{22}, \text{Destination IP}) = \frac{2}{20} (-\frac{2}{2} \log_2 \frac{2}{2} - 0/2 \log_2 0/2)
\]

\+
\[
\frac{4}{20} (-\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4})
\]

\+
\[
\frac{14}{20} (-\frac{13}{20} \log_2 \frac{13}{20} - \frac{1}{14} \log_2 \frac{1}{14})
\]

\[
= 0 + 0.2 + 0.2599
\]

\[
= 0.4599
\]

Calculate gain Destination Port = 22 of Destination IP based on Equation (3.4) in Section 3.4.2

\[
\text{Gain}(S, A) = E(S) - E(S, A)
\]

\[
G(S_{22}, \text{Destination IP}) = E(22) - E(S_{22}, \text{Destination IP})
\]

\[
G(S_{22}, \text{Destination IP}) = 0.6098 - 0.4599
\]

\[
= 0.1499
\]

Source IP has the highest Gain. Therefore, Source IP used as the decision node under Protocol = TCP and Destination Port = 22 which shown in Figure 4.19.
The next step to find node under Protocol=TCP, Destination Port=22 and Source IP = 192.168.2.25. To calculate Gain for each attributes use Protocol=TCP, Destination Port=22 and Source IP = 192.168.2.25

**Table 4.15** Protocol = TCP, Source IP = 192.168.2.25 and Destination Port = 22

<table>
<thead>
<tr>
<th>No.</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.2.25</td>
<td>10.10.1.3</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>192.168.2.25</td>
<td>10.10.1.3</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.25</td>
<td>10.10.1.5</td>
<td>TCP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Figure 4.20** The set of 20 examples Source Port =22 are 17 intrusion and 3 normal
\[
E(S) = -p \log_2(p) - p \log_2(1 - p)
\]
\[
E(192.168.0.25) = -\frac{9}{9} \log_2\left(\frac{9}{9}\right) - \frac{0}{9} \log_2\left(\frac{0}{9}\right) = 0
\]

From the calculation above, \(E(192.168.0.25)\) is equal 0. This is pure impurity. That is all occurrences using Protocol=TCP, Destination Port = 22 and Source IP =192.168.2.25 is Intrusion =’Yes’. For another is same, \(E(192.168.0.5)\) is equal to 0 and \(E(192.168.0.13)\) is equal to 0. Therefore, decision tree is shown in Figure 4.21.

![Figure 4.21 Leaf node Protocol = TCP and Destination Port = 22](image)

This process goes on until all data classified perfectly or run out of attributes. The complete of tree is shown in Figure 4.22. Decision tree can simplified by pruning all connections to normal and not classified is shown in Figure 4.23.
Figure 4.22 Complete decision tree network traffics
Figure 4.23 Pruning decision tree network traffics
4.4 RULE EXTRACTION IF-THEN

The knowledge represented in decision trees can be extracted and represented in the form of IF-THEN rules. One rule created for each path from the root to a leaf node. Each attribute-value pair along a given path forms a conjunction in the rule antecedent (“IF” part). The leaf node holds the class prediction, forming the rule consequent (“THEN” part). The IF-THEN rules may be easier than decision trees for humans to understand. Decision tree in shown Figure 4.23 have been extracted in the form IF-THEN as shown Figure 4.24.

R1 : IF (Protocol = ICMP) THEN Intrusion = Yes
R2 : IF (Protocol = TCP) AND (Dest Port = 22) AND (Source IP=192.168.2.25) THEN Intrusion = Yes
R3 : IF (Protocol = TCP) AND (Dest Port = 22) AND (Source IP = 192.168.0.5) THEN Intrusion = Yes
R4 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.2) AND (Source IP = 192.168.0.5) THEN Intrusion = Yes
R5 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.2) AND (Source IP = 192.168.2.16) THEN Intrusion = Yes
R6 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.0.5) AND (Source Port = 1458) THEN Intrusion = Yes
R7 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.0.5) AND (Source Port = 1463) THEN Intrusion = Yes
R8 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.0.5) AND (Source Port = 1465) THEN Intrusion = Yes
R9 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.0.5) AND (Source Port = 1466) THEN Intrusion = Yes
R10 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.1.13) AND (Source Port = 1173) THEN Intrusion = Yes
R11 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.1.13) AND (Source Port = 1175) THEN Intrusion = Yes
R12 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.1.13) AND (Source Port = 1179) THEN Intrusion = Yes
R13 : IF (Protocol = TCP) AND (Dest Port = 80) AND (Dest IP = 10.10.1.3) AND (Source IP = 192.168.1.13) AND (Source Port = 1180) THEN Intrusion = Yes
R14 : IF (Protocol = TCP) AND (Dest Port = 25) THEN Intrusion = Yes
R15 : IF (Protocol = TCP) AND (Dest Port = 110) THEN Intrusion = Yes
R16 : IF (Protocol = TCP) AND (Dest Port = 161) THEN Intrusion = Yes
R17 : IF (Protocol = TCP) AND (Dest Port = 162) THEN Intrusion = Yes
R18 : IF (Protocol = TCP) AND (Dest Port = 3306) THEN Intrusion = Yes

Figure 4.24 Rules extraction to IF-THEN
4.5 CLASSIFICATION INTRUSION OR NORMAL OF THE NEW NETWORK PACKET

The example of extract rules of tree decision (see Figure 4.24) is rules of intrusion. Every network packet will be checked with these rules one by one. If there is one of the rules contained in a network packet, the network packet is intrusion. This called the IDS, the network can determine the packet is intrusion or not. For illustration, given new network traffics as show in Table 4.16.

Table 4.16 The new network traffics

<table>
<thead>
<tr>
<th>No.</th>
<th>Source IP</th>
<th>Source Port</th>
<th>Destination IP</th>
<th>Destination Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.2.25</td>
<td>1430</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>192.168.0.5</td>
<td>1557</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>192.168.0.5</td>
<td>1622</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>192.168.1.4</td>
<td>1647</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>192.168.1.13</td>
<td>1185</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>192.168.0.5</td>
<td>1500</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>7</td>
<td>192.168.0.2</td>
<td>1352</td>
<td>10.10.1.5</td>
<td>3306</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.25</td>
<td>1356</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>?</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.20</td>
<td>1890</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>?</td>
</tr>
</tbody>
</table>

The rules based on the results of the training set of network traffics can to predictive the new network traffics is intrusion activities or normal activities. Table 4.17 shows to predictive the new network traffics is intrusion activities or normal activities.
Table 4.17 To predict new network traffics

<table>
<thead>
<tr>
<th>No.</th>
<th>Source IP</th>
<th>Source Port</th>
<th>Destination IP</th>
<th>Destination Port</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.2.25</td>
<td>1430</td>
<td>10.10.1.3</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>192.168.0.5</td>
<td>1557</td>
<td>10.10.1.3</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>192.168.0.5</td>
<td>1622</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>192.168.1.4</td>
<td>1647</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>192.168.1.13</td>
<td>1185</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>192.168.0.5</td>
<td>1500</td>
<td>10.10.1.2</td>
<td>80</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>192.168.0.2</td>
<td>1352</td>
<td>10.10.1.5</td>
<td>3306</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.25</td>
<td>1356</td>
<td>10.10.1.5</td>
<td>22</td>
<td>TCP</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.20</td>
<td>1890</td>
<td>10.10.1.5</td>
<td>80</td>
<td>TCP</td>
<td>No</td>
</tr>
</tbody>
</table>

Consider a case for network traffic in line 1. This network packet is intrusion, because network traffic in line 1 have subset characteristics of R2. (see Figure 4.24). Because on R2 everything network packet has Protocol = TCP, Destination Port = 22 and source IP = 192.168.2.25 is intrusion. These characters belong line 1 in Table 4.17.

For another example, consider network traffics in line 6. This network packet is intrusion, because network traffic in line 6 have subset characteristics of R4. Because in R4 everything network packet has Protocol = TCP, Destination Port = 80, Destination IP = 10.10.1.2 and source IP = 192.168.0.5 is intrusion. These characters belong line 6 in Table 4.17.

4.6 Implementation rules of intrusion signature into firewall rules

Rules of intrusion implemented in firewall rules for prevention, where every network packet have rules of intrusion signature will drop. The network packet cannot allow to go through a firewall machine. The rules of intrusion signature implemented firewall rules is shown in Figure 4.25.
R1 : -A FORWARD -p icmp -j DROP
R2 : -A FORWARD -p tcp -s 192.168.2.25 --dport 22 -j DROP
R3 : -A FORWARD -p tcp -s 192.168.0.5 --dport 22 -j DROP
R4 : -A FORWARD -p tcp -s 192.168.0.5 -d 10.10.1.2 --dport 80 -j DROP
R5 : -A FORWARD -p tcp -s 192.168.0.5 --sport 1458 -d 10.10.1.3 --dport 80 -j DROP
R6 : -A FORWARD -p tcp -s 192.168.0.5 --sport 1463 -d 10.10.1.3 --dport 80 -j DROP
R7 : -A FORWARD -p tcp -s 192.168.0.5 --sport 1465 -d 10.10.1.3 --dport 80 -j DROP
R8 : -A FORWARD -p tcp -s 192.168.0.5 --sport 1466 -d 10.10.1.3 --dport 80 -j DROP
R9 : -A FORWARD -p tcp -s 192.168.1.13 --sport 1173 -d 10.10.1.3 --dport 80 -j DROP
R10 : -A FORWARD -p tcp -s 192.168.1.13 --sport 1175 -d 10.10.1.3 --dport 80 -j DROP
R11 : -A FORWARD -p tcp -s 192.168.1.13 --sport 1179 -d 10.10.1.3 --dport 80 -j DROP
R12 : -A FORWARD -p tcp -s 192.168.1.13 --sport 1180 -d 10.10.1.3 --dport 80 -j DROP
R13 : -A FORWARD -p tcp --dport 25 -j DROP
R14 : -A FORWARD -p tcp --dport 110 -j DROP
R15 : -A FORWARD -p tcp --dport 161 -j DROP
R16 : -A FORWARD -p tcp --dport 162 -j DROP
R17 : -A FORWARD -p tcp --dport 3306 -j DROP

**Figure 4.25.** Rules of intrusion implemented into firewall rules

Researcher uses the command 'FORWARD' because firewall rules above will be implemented in machine as the router will forward packets from outgoing to incoming LAN. Each packet network must be checked against the rules established by firewall, if the rules match the firewall will drop the network packet.

Rules of intrusion to be implemented into *iptables* firewall Linux are shown in Figure 4.26, Figure 4.27 and Figure 4.28.
Figure 4.26 Default `iptables` firewall rules before enter rules

Figure 4.27 Command line to enter rules into `iptables` firewall rules
### Figure 4.28 Iptables firewall rules after enter rules

<table>
<thead>
<tr>
<th>Chain</th>
<th>target</th>
<th>prot opt source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain INPUT (policy ACCEPT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.2.25</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.0.5</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.0.5</td>
<td>server1.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.2.16</td>
<td>server1.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.0.5</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.0.5</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.0.5</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.0.5</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.1.13</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.1.13</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.1.13</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>192.168.1.13</td>
<td>server2.research.com</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>tcp --</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>icmp --</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
</tbody>
</table>

| Chain FORWARD (policy ACCEPT) | | | |
| DROP | tcp -- | 192.168.0.5 | anywhere | tcp dpt:ssh |
| DROP | tcp -- | 192.168.0.5 | anywhere | tcp dpt:ssh |
| DROP | tcp -- | 192.168.0.5 | server1.research.com | tcp dpt:http |
| DROP | tcp -- | 192.168.2.16 | server1.research.com | tcp dpt:http |
| DROP | tcp -- | 192.168.0.5 | server2.research.com | tcp spt:ncat-dm dpt:http |
| DROP | tcp -- | 192.168.0.5 | server2.research.com | tcp spt:nucleus dpt:http |
| DROP | tcp -- | 192.168.0.5 | server2.research.com | tcp spt:pipes dpt:http |
| DROP | tcp -- | 192.168.0.5 | server2.research.com | tcp spt:osnsoft-lm dpt:http |
| DROP | tcp -- | 192.168.1.13 | server2.research.com | tcp spt:1273 dpt:http |
| DROP | tcp -- | 192.168.1.13 | server2.research.com | tcp spt:1375 dpt:http |
| DROP | tcp -- | 192.168.1.13 | server2.research.com | tcp spt:1199 dpt:http |
| DROP | tcp -- | 192.168.1.13 | server2.research.com | tcp spt:1180 dpt:http |
| DROP | tcp -- | anywhere | anywhere | tcp dpt:smtp |
| DROP | tcp -- | anywhere | anywhere | tcp dpt:pop3 |
| DROP | tcp -- | anywhere | anywhere | tcp dpt:smp |
| DROP | tcp -- | anywhere | anywhere | tcp dpt:snmptrap |
| DROP | tcp -- | anywhere | anywhere | tcp dpt:3306 |
| DROP | icmp -- | anywhere | anywhere | |

### 4.7 DECISION TREE TO CREATE RULES MINIMIZE FIREWALL OF INTRUSION

For security all of intrusion activities is show in Tables 4.7 must be drop is shown in Figure 4.29.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ra1</td>
<td>-A FORWARD -p icmp -s 192.168.1.25 -d 10.10.1.2 -j DROP</td>
</tr>
<tr>
<td>Ra2</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1142 -d 10.10.1.2 --dport 161 -j DROP</td>
</tr>
<tr>
<td>Ra3</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1143 -d 10.10.1.2 --dport 162 -j DROP</td>
</tr>
<tr>
<td>Ra4</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1179 -d 10.10.1.2 --dport 110 -j DROP</td>
</tr>
<tr>
<td>Ra5</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1175 -d 10.10.1.2 --dport 25 -j DROP</td>
</tr>
<tr>
<td>Ra6</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1130 -d 10.10.1.3 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra7</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1131 -d 10.10.1.3 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra8</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1345 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra9</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1347 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra10</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1348 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra11</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1351 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra12</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1352 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra13</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1356 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra14</td>
<td>-A FORWARD -p tcp -s 192.168.2.25 -sport 1354 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra15</td>
<td>-A FORWARD -p icmp -s 192.168.0.5 -d 10.10.1.3 -j DROP</td>
</tr>
<tr>
<td>Ra16</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1392 -d 10.10.1.2 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra17</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1394 -d 10.10.1.2 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra18</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1396 -d 10.10.1.2 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra19</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1400 -d 10.10.1.2 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra20</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1401 -d 10.10.1.2 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra21</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1405 -d 10.10.1.2 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra22</td>
<td>-A FORWARD -p icmp -s 192.168.0.5 -d 10.10.1.3 -j DROP</td>
</tr>
<tr>
<td>Ra23</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1458 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra24</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1465 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra25</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1463 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra26</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1466 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra27</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1394 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra28</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1398 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra29</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1402 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra30</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1409 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra31</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1411 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra32</td>
<td>-A FORWARD -p tcp -s 192.168.0.5 -sport 1413 -d 10.10.1.5 --dport 22 -j DROP</td>
</tr>
<tr>
<td>Ra33</td>
<td>-A FORWARD -p tcp -s 192.168.1.13 -sport 1173 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra34</td>
<td>-A FORWARD -p tcp -s 192.168.1.13 -sport 1175 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra35</td>
<td>-A FORWARD -p tcp -s 192.168.1.13 -sport 1179 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra36</td>
<td>-A FORWARD -p tcp -s 192.168.1.13 -sport 1180 -d 10.10.1.3 --dport 80 -j DROP</td>
</tr>
<tr>
<td>Ra37</td>
<td>-A FORWARD -p tcp -s 192.168.1.13 -sport 1158 -d 10.10.1.2 --dport 25 -j DROP</td>
</tr>
<tr>
<td>Ra38</td>
<td>-A FORWARD -p icmp -s 192.168.2.16 -d 10.10.1.2 -j DROP</td>
</tr>
<tr>
<td>Ra39</td>
<td>-A FORWARD -p tcp -s 192.168.2.16 -sport 34592 -d 10.10.1.2 --dport 162 -j DROP</td>
</tr>
<tr>
<td>Ra40</td>
<td>-A FORWARD -p tcp -s 192.168.2.16 -sport 34595 -d 10.10.1.2 --dport 161 -j DROP</td>
</tr>
<tr>
<td>Ra41</td>
<td>-A FORWARD -p tcp -s 192.168.2.16 -sport 53814 -d 10.10.1.2 --dport 3306 -j DROP</td>
</tr>
<tr>
<td>Ra42</td>
<td>-A FORWARD -p tcp -s 192.168.2.16 -sport 53815 -d 10.10.1.2 --dport 3306 -j DROP</td>
</tr>
</tbody>
</table>

Figure 4.29 All of intrusion activities implemented into firewall rules.
Before to develop rules filtering by using packet filter, anything have to be considered beforehand how far demarcation which will be applied. Because more and more demarcation applied hence increased the search time and space requirements of the packet filtering process and consequences to make downhill performance progressively (Al-Shaer E.S. and Hamed H.H., 2006; Suehring S. and Ziegler R.L., 2006 and Predrag Pale T.K., 2007). This matter because every incoming network packet and go out the network checked beforehand by rules alternately until matching rule found in firewall.

To rewrite some of all network traffics (see in Table 4.7) the number row 6, 7, 8, 9, 10, 11, 12, 13 and 14 is shown in Tables 4.18.

### Tables 4.18 Some rules of all intrusion activities

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Intrusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP</td>
<td>Port</td>
<td>IP</td>
<td>Port</td>
</tr>
<tr>
<td>6</td>
<td>192.168.2.25</td>
<td>1130</td>
<td>10.10.1.3</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>192.168.2.25</td>
<td>1131</td>
<td>10.10.1.3</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.25</td>
<td>1345</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.25</td>
<td>1347</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>192.168.2.25</td>
<td>1348</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>192.168.2.25</td>
<td>1351</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>192.168.2.25</td>
<td>1352</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td>13</td>
<td>192.168.2.25</td>
<td>1356</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>192.168.2.25</td>
<td>1354</td>
<td>10.10.1.5</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

All of intrusion characteristic is shown in Tables 4.18 implemented into firewall rules is shown in Figure 4.30.
By using the decision tree all of rule above all represented by one rule is shown in Figure 4.31.

**Figure 4.30** Some rules from all of intrusion activities

By using the decision tree all of rule above all represented by one rule is shown in Figure 4.31.

**Figure 4.31.** R2 represent Ra6, Ra7, Ra8, Ra9, Ra10, Ra11, Ra12, Ra13, Ra14

If the observed of all network traffic that has Protocol = TCP, source IP = 192.168.2.25 and Destination Port = 22 and any value of Destination IP is intrusion. So, do not need Destination IP for rules, because anything values of Destination IP that has Protocol = TCP, source IP = 192.168.2.25 and Destination Port = 22 is intrusion. Based on all of network traffics R2 represent and replace Ra6, Ra7, Ra8, Ra9, Ra10, Ra11, Ra12, Ra13, Ra14. Same case for other rules.

By using decision tree and all data training is shown Table 4.7, 42 rules of intrusion activities network traffics to represent 17 rules will implemented into firewall rules. Compare between Figure 4.25 and Figure 4.29, by using decision tree, one rule to represent two or more rules. This is become simple rules to be implemented firewall rules.
4.8 EXPERIMENTAL AND RESULT

For the calculation of ID3 algorithm and then implemented directly in the *iptables* firewall rules automatically be made software. The name is software *nips-nid2s3*. This software is running in Linux Operating System. This research, computers as routers and firewalls using Linux Slackware 12.1. Figure 4.32 shows the extract and compile *nips-nid2s3* software.

All of data training in Table 4.7 contained in the file 'data' is shown in Figure 4.33
Figure 4.33 All of data training in ‘data’ file

Execute program using command `./syh -a sdpd` and then to see `iptables` firewall rules is shown in Figure 4.34.
Figure 4.34 Execute and then to see into firewall rules

In Figure 4.34 shows how to is to create 17 rules into firewall. The rule in line 1, that is mean is all of network packets use protocol ICMP is drop.
Figure 4.35 Ping using protocol ICMP from 192.168.2.21 to 10.10.1.2 is not reply

Researcher tries computer client connect to server using protocol ICMP as shown in Figure 4.35. This packet network to drop and not allow forwarded by firewall. The network traffics have characteristics same match into firewall rules at line 1, that is mean network packet has Protocol = ICMP is drop. This network packet is intrusion. Firewall cannot allow forwarded a network packet has Protocol=ICMP.

For another, researcher tries to request to server, which request from client has characteristics of intrusion.
Figure 4.36 Request SSH from 192.168.2.25 to 10.10.1.2 is connection timed out

In Figure 4.36 is someone has IP address 192.168.2.25 connect to 10.10.1.2 by SSH service (port 22) is connection timed out. This packet network to drop and not allow forwarded by firewall. The network traffics have characteristics same match into firewall rules at line 6, that is mean network packet has Protocol = TCP and source IP = 192.168.2.25 and Destination Port = 20 any value of Destination IP action is drop. This network packet is intrusion. Firewall cannot allow forwarded.

Figure 4.37 Request URL from 192.168.0.5 to 10.10.1.2 is taking too long to respond
In Figure 4.37 is someone has IP address 192.168.0.5 connect request Web browser (port 80) to 10.10.1.2 port 80 s taking too long to respond. This network to drop and not allow forwarded by firewall. The network traffics have characteristics same match into firewall rules at line 8, that is mean network packet has Protocol = TCP and source IP = 192.168.0.5 and Destination IP = 10.10.1.2 and Destination Port = 80 action is drop. This network packet is intrusion. Firewall cannot allow forwarded.

![Figure 4.37](image1)

**Figure 4.38** Request SSH from 192.168.0.5 to 10.10.1.5 is connection timed out

In Figure 4.38 is someone has IP address 192.168.0.5 connect to 10.10.1.5 by SSH service (port 22) is connection timed out. This packet network to drop and not allow forwarded by firewall. The network traffics have characteristics same match into firewall rules at line 7, that is mean network packet has Protocol = TCP and source IP = 192.168.0.5 and Destination Port = 22 and any value of Destination IP action is drop. This network packet is intrusion. Firewall cannot allow forwarded.

4.9 CONCLUSION

In this chapter, the implementation has generate rules of intrusion as NIDS and to implemented into firewall as prevention. Three LAN have been develop to support the client request to server. A computer become router and firewall which have been connected in LAN. It is a function to record log files. *Snort* software has been used to determine either the intrusion or normal activities and from the log files. Decision tree
has been used to generate rules. Next, these rules implemented into firewall as prevention of intrusion network packet. These the filter rules reflect the current network traffics. Overall, this way can minimize the number of that implemented in firewall.
5.1 CONCLUSION

Network security has become a critical issue with the development of business and other transactions through the internet system. Firewall is one important element in network security systems because it can drop the a network packet incoming to LAN. Firewall can not define a network packet is intrusion or normal. An intrusion can be defined as any set of actions that threaten the integrity, confidentiality or availability of a network resource, such as user account, file system, system kernels and so on. Specifies the network packet is intrusion to be implemented on the firewall rules are very difficult. An analyst should reviews large data from network traffics previously. Meanwhile, to update and manage the firewall rules are very difficult and takes a lot of time.

By using the ID3 algorithm decision tree classifier to generate rules where network traffics as data training. A network packet from the network traffics can be seen from the log files. To determine the network packet is intrusion or normal using snort application.

Rules generated can determine a new network packet is intrusion or normal. These rules are implemented into firewall automatically where the firewall will drop the network packet that match those rules as intrusion.
This research contributes: first, to create rules that function to determine the network traffic is the intrusion and then implemented rules into the firewall as a prevention automatically. This combination is called NIPS, because it can determine network packet is intrusion automatically and can be prevented by using a firewall.

This method can minimize the number of rules in the firewall, where one rule can replace two or more rules and make a better firewall performance. This is very helpful and easier to update and manage the firewall rules.

This research made software that is named \textit{nips-nids2s3}, this software helps construct decision tree to generate the rules and implemented into firewall \textit{iptables} automatically as prevention.

5.2 RECOMMENDATIONS

Data collection for intrusion do if the network packets do intrusion several times from the same source IP address. If intrusion is conducted just one time, it can be an unintentionally and ignored. Intruder do many times intrusion many times, with goal to find and get as much as information from a machine target. Intruder takes a lot of time and in many ways to get information such as port scans, ping, send packages, etc. It is impossible intruder to do intrusion just once.

In large computer networks, this condition will produces large data set and also generate so many rules. It is recommended reduces some rules become one rule using the port range or multi port and IP range or IP network by masking the same protocol and action.

For future work, the whole way this is done with real time system and performed on large computer networks such as Wide Area Network (WAN) and internet.
REFERENCES


Firewall is it Needed (online). http://www.sendmeemail.co.uk/advice/html/what_is_a_firewall.htm (15 July 2008)


Iptables Tutorial 1.2.2 (online). http://security.maruhn.com/iptables-tutorial/x12114.html (27 June 2008)


APPENDIX A

NETWORK TRAFFIC LOGS

i) Intrusion Activities Logs using snort software in folder /var/log/snort/alert

```plaintext
[**] [1:1418:13] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2]
******* Seq: 0x9C20F9B4 Ack: 0x0 Win: 0xFFFF TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
[Xref => http://www.securityfocus.com/bid/4088]

[**] [1:1420:13] SNMP trap tcp [**]
[Classification: Attempted Information Leak] [Priority: 2]
******* Seq: 0x604278A Ack: 0x0 Win: 0xFFFF TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
[Xref => http://www.securityfocus.com/bid/4088]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/09-22:20:30.855749 0:0:B4:98:21:B0 -> 0:0:B4:98:21:B0 type:0x800 len:0x57
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0x00 ID:40045 IpLen:20 DgmLen:73
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:110 -> 192.168.2.25:1179 TCP TTL:64 TOS:0x0 ID:49893 IpLen:20 DgmLen:45 DF
Seq: 0x7EDBB8CC
(17 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0x00 ID:40045 IpLen:20 DgmLen:73
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:110 -> 192.168.2.25:1179 TCP TTL:64 TOS:0x0 ID:49893 IpLen:20 DgmLen:45 DF
Seq: 0x7EDBB8CC
(17 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-02:59:45.547181 0:0:B4:98:21:B0 -> 0:0:B4:98:21:B0 type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0x00 ID:18410 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
```
** ORIGINAL DATAGRAM DUMP:
10.10.2.80 -> 192.168.0.5:1392 TCP TTL:64 TOS:0x0 IP:20 DgmLen:48 DF
Seq: 0xABA54F77
(20 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-02:59:46.686349 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0x0 ID:18412 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:80 -> 192.168.0.5:1394 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0xAC888CBE
(20 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:895:8] WEB-CGI redirect access [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/10-03:00:00.599880 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x302
192.168.0.5:1400 -> 10.10.2.80 TCP TTL:128 TOS:0x0 ID:11236 IpLen:20 DgmLen:757 DF
** AP** Seq: 0x3D88FFCD Ack: 0x3D99F925 Win: 0xFFFF TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2000-0382][Xref =>
http://www.securityfocus.com/bid/1179]

[**] [1:895:8] WEB-CGI redirect access [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/10-03:01:56.097112 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.0.5:1400 -> 10.10.1.2:80 TCP TTL:128 TOS:0x0 ID:11236 IpLen:20 DgmLen:757 DF
** AP** Seq: 0xC303018D Ack: 0xC7F84882 Win: 0xFFFF TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2000-0382][Xref =>
http://www.securityfocus.com/bid/1179]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-03:01:46.285346 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x7E
192.168.0.5:1401 -> 10.10.1.2:80 TCP TTL:128 TOS:0x0 ID:11236 IpLen:20 DgmLen:757 DF
Seq: 0x24DCE69E
(20 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-03:01:46.285346 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x7E
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:47275 IpLen:20 DgmLen:112
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3 -> 192.168.0.5 ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type: 8 Code: 0 Csum: 26953 Id: 3850 SeqNo: 1
(56 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-03:25:42.348971 0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:47284 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1458 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0x16D4FD96
(20 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-03:25:44.042042 0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x24E
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:47287 IpLen:20 DgmLen:576
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1465 TCP TTL:64 TOS:0x0 ID:47643 IpLen:20 DgmLen:1500 DF
Seq: 0x17899C51
(520 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-03:25:47.832002 0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x24E
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:47288 IpLen:20 DgmLen:576
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1466 TCP TTL:64 TOS:0x0 ID:45805 IpLen:20 DgmLen:1500 DF
Seq: 0x17DAFC51
(520 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-04:33:08.098943 0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x208
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:47290 IpLen:20 DgmLen:506
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1466 TCP TTL:64 TOS:0x0 ID:45805 IpLen:20 DgmLen:478 DF
Seq: 0x170C8D9A
(450 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-04:33:08.098943 0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x208
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:26239 IpLen:20 DgmLen:112
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:22 -> 192.168.2.25:1130 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-04:33:58.593157 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:62464 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:22 -> 192.168.2.25:1131 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0xB131F2B
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-07:02:06.347845 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0xC0 ID:62052 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:22 -> 192.168.2.25:1345 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0x587FA76C
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-07:02:32.361511 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0xC0 ID:62058 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:22 -> 192.168.2.25:1347 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0x70C690D6
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-07:05:23.271162 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0xC0 ID:62067 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:22 -> 192.168.2.25:1348 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0x84827577
(20 more bytes of original packet)
** END OF DUMP
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.2.25:1351 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF Seq: 0xFC7832B
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
Classification: Potentially Bad Traffic [Priority: 2]
12/10-07:10:5:24.541795 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x52
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0x0 ID:62069 IpLen:20 DgmLen:68
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.2.25:1352 TCP TTL:64 TOS:0x0 ID:9056 IpLen:20 DgmLen:40 DF Seq: 0x10080A7
(12 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
Classification: Potentially Bad Traffic [Priority: 2]
12/10-07:10:5:35.367546 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x24E
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0x0 ID:62075 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.25
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.2.25:1356 TCP TTL:64 TOS:0x0 ID:46541 IpLen:20 DgmLen:1500 DF Seq: 0x13F93EAB
(520 more bytes of original packet)
** END OF DUMP

[**] [1:114:11] WEB-MISC handler access [**]
Classification: access to a potentially vulnerable web application [Priority: 2]
12/10-07:10:5:36.752156 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x1B9
192.168.2.25:1354 -> 10.10.1.5:80 TCP TTL:128 TOS:0x0 ID:4067 IpLen:20 DgmLen:427 DF
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1394 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF Seq: 0x13F93EAB
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
Classification: Potentially Bad Traffic [Priority: 2]
12/10-07:10:5:40.752156 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0x0 ID:62080 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1398 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF Seq: 0x219C3954
(20 more bytes of original packet)
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
Classification: Potentially Bad Traffic [Priority: 2]
12/10-07:10:5:45.752156 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x24E
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0x0 ID:62083 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1398 TCP TTL:64 TOS:0x0 ID:56881 IpLen:20 DgmLen:1500 DF Seq: 0x285A0AF4
(520 more bytes of original packet)
** END OF DUMP
107

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-07:10:35.049931 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0x00 ID:62084 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
12/10-07:10:35.049931 -> 10.10.1.5 TX:
0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0x00 ID:62084 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** END OF DUMP

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-07:11:30.758547 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0xC0 ID:62085 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
12/10-07:11:30.758547 -> 10.10.1.5 TX:
0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.5 ICMP TTL:64 TOS:0xC0 ID:62085 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** END OF DUMP

[**] [1:895:8] WEB-CGI redirect access [**]
[Classification: Attempted Information Leak] [Priority: 2]
192.168.0.5:1411 -> 10.10.1.5:80 TCP TTL:128 TOS:0x00 ID:5933 IpLen:20 DgmLen:728 DF
***AP*** Seq: 0xA9BE804D  Ack: 0x6FE9C253 Win: 0xFFFF TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2000-0382]

[**] [1:895:8] WEB-CGI redirect access [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/10-07:11:52.767158 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x306
192.168.0.5:1413 -> 10.10.1.5:80 TCP TTL:128 TOS:0x00 ID:5981 IpLen:20 DgmLen:760 DF
***AP*** Seq: 0x57D9768C  Ack: 0x7C03B6DE Win: 0xFFFF TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2000-0382]

[**] [1:469:4] ICMP PING NMAP [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/11-00:38:16.056092 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x3C
192.168.2.16 -> 10.10.1.2 ICMP TTL:48 TOS:0x00 ID:42521 IpLen:20 DgmLen:28
Type:8 Code:0 ID:62073 Seq:45584 ECHO

[**] [1:1418:13] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/11-00:38:16.603968 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x3C
192.168.2.16 -> 10.10.1.2 TCP TTL:53 TOS:0x00 ID:38721 IpLen:20 DgmLen:40

[**] [1:1420:13] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/11-00:38:16.040032 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x3C
192.168.2.16:34592 -> 10.10.1.2:161 TCP TTL:53 TOS:0x00 ID:38721 IpLen:20 DgmLen:40

[**] [1:1418:13] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/11-00:38:16.603968 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x3C
192.168.2.16 -> 10.10.1.2 TCP TTL:53 TOS:0x00 ID:38721 IpLen:20 DgmLen:40

[**] [1:1420:13] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/11-00:38:16.040032 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x3C
192.168.2.16:34592 -> 10.10.1.2:161 TCP TTL:53 TOS:0x00 ID:38721 IpLen:20 DgmLen:40
**ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/11-00:38:50.424780 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x52
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0xC0 ID:17264 IpLen:20 DgmLen:68
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.2.16
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:3306 -> 192.168.2.16:53814 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
Seq: 0x0
(12 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

**ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/11-00:39:04.831516 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x52
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0xC0 ID:17265 IpLen:20 DgmLen:68
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.2.16
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:3306 -> 192.168.2.16:53815 TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
Seq: 0x0
(12 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

**ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-04:59:17.202551 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:16461 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
Seq: 0x9689FA3A
(20 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

**ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/10-04:59:17.324732 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0x5A
192.168.2.30 -> 10.10.1.3 ICMP TTL:64 TOS:0xC0 ID:16462 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
Seq: 0x99ACD224
(20 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]
(20 more bytes of original packet)

** END OF DUMP

[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]

[**] [1:1141:11] WEB-MISC handler access [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
12/10-04:59:30.635958 0:8:2:E1:27:AA -> 0:0:B4:98:21:B0 type:0x800 len:0x1B9
***AP*** Seq: 0xC8B30EC2 Ack: 0x9968151B Win: 0xFC41 TcpLen: 20
[Xref => http://cgi.nessus.org/plugins/dump.php3?id=10100][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0148][Xref =>

[**] [1:472:5] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/11-15:06:44.508449 0:0:B4:98:21:B0 -> 0:55:D0:F1:DC:1B type:0x800 len:0xAA
192.168.2.30 -> 10.10.1.2 ICMP TTL:64 TOS:0xC0 ID:21848 IpLen:20 DgmLen:156
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
Seq: 0xB58FB249
(100 more bytes of original packet)
** END OF DUMP
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0265][Xref =>
http://www.whitehats.com/info/IDS135]
ii) Normal Activities Logs using snort software in folder /var/log/snort/alert

The other hand, there is source IP address of intrusion activities doing normal activities. They all in folder /var/log/snort/snort.log.*

```
12/10-07:10:07.947573 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62077 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1384
TCP TTL:64 TOS:0x0 ID:10034 IpLen:20 DgmLen:1500 DF
Seq: 0x195C194B
(520 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:10:13.038867 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62079 IpLen:20 DgmLen:68
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1391
TCP TTL:64 TOS:0x0 ID:50455 IpLen:20 DgmLen:40 DF
Seq: 0x19A3336C
(12 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:10:15.038869 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62086 IpLen:20 DgmLen:68
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1409
TCP TTL:64 TOS:0x0 ID:47038 IpLen:20 DgmLen:40 DF
Seq: 0x681F9B1D
(12 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:10:15.038869 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62086 IpLen:20 DgmLen:68
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1409
TCP TTL:64 TOS:0x0 ID:47038 IpLen:20 DgmLen:40 DF
Seq: 0x681F9B1D
(12 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:10:15.038869 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62086 IpLen:20 DgmLen:68
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1409
TCP TTL:64 TOS:0x0 ID:47038 IpLen:20 DgmLen:40 DF
Seq: 0x681F9B1D
(12 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:11:30.844676 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62086 IpLen:20 DgmLen:68
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1409
TCP TTL:64 TOS:0x0 ID:47038 IpLen:20 DgmLen:40 DF
Seq: 0x681F9B1D
(12 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:11:43.212440 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62092 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1411
TCP TTL:64 TOS:0x0 ID:62918 IpLen:20 DgmLen:1500 DF
Seq: 0x6FE9C253
(520 more bytes of original packet)
** END OF DUMP
```

```
12/10-07:11:43.212440 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:62092 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.0.5:1411
TCP TTL:64 TOS:0x0 ID:62918 IpLen:20 DgmLen:1500 DF
Seq: 0x6FE9C253
(520 more bytes of original packet)
** END OF DUMP
```
12/10-07:12:03.396917 192.168.0.5:1413 -> 10.10.1.5:80
TCP TTL:128 TOS:0x0 ID:6010 IpLen:20 DgmLen:761 DF
***AP*** Seq: 0x57D97B93  Ack: 0x7C03CAC7  Win: 0xFFF2 TcpLen: 20

12/11-00:43:13.223665 192.168.2.30 -> 10.10.1.2
ICMP TTL:64 TOS:0xC0 ID:17267 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.16
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:80 -> 192.168.2.16:49869
TCP TTL:64 TOS:0x0 ID:58872 IpLen:20 DgmLen:1500 DF
Seq: 0x63587A97
(520 more bytes of original packet)
** END OF DUMP

12/11-00:43:57.140425 192.168.2.30 -> 10.10.1.2
ICMP TTL:64 TOS:0xC0 ID:17271 IpLen:20 DgmLen:80
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.16
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:80 -> 192.168.2.16:49870
TCP TTL:64 TOS:0x0 ID:32847 IpLen:20 DgmLen:52 DF
Seq: 0x871A8285
(24 more bytes of original packet)
** END OF DUMP

12/11-00:44:00.634244 192.168.2.30 -> 10.10.1.2
ICMP TTL:64 TOS:0xC0 ID:17272 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.16
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:80 -> 192.168.2.16:49871
TCP TTL:64 TOS:0x0 ID:12694 IpLen:20 DgmLen:1500 DF
Seq: 0x8CA8C85C
(520 more bytes of original packet)
** END OF DUMP

12/11-00:44:12.642863 192.168.2.30 -> 10.10.1.2
ICMP TTL:64 TOS:0xC0 ID:17274 IpLen:20 DgmLen:576
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.2.16
** ORIGINAL DATAGRAM DUMP:
10.10.1.2:80 -> 192.168.2.16:49872
TCP TTL:64 TOS:0x0 ID:65005 IpLen:20 DgmLen:1500 DF
Seq: 0x93BD0881
(520 more bytes of original packet)
** END OF DUMP

12/12-07:01:58.825394 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:43929 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1181
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0xDEE28FFC
(20 more bytes of original packet)
** END OF DUMP

12/12-07:02:23.383762 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:43932 IpLen:20 DgmLen:76
Type:5  Code:1  REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1183
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0xF5EEEBB2
(20 more bytes of original packet)
** END OF DUMP

12/12-07:02:28.429310 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:43933 IpLen:20 DgmLen:68
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.0.5
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.0.5:1185
TCP TTL:64 TOS:0x0 ID:10639 IpLen:20 DgmLen:40 DF
Seq: 0xF5EF1F10
(12 more bytes of original packet)
** END OF DUMP

12/12-07:13:11.164337 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:43934 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:22 -> 192.168.1.13:1203
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
Seq: 0x546B278A
(20 more bytes of original packet)
** END OF DUMP

12/12-07:15:26.465507 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:43944 IpLen:20 DgmLen:68
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:22 -> 192.168.1.13:1204
TCP TTL:64 TOS:0x0 ID:5502 IpLen:20 DgmLen:40 DF
Seq: 0xD25DB009
(12 more bytes of original packet)
** END OF DUMP

12/12-07:34:30.926554 192.168.2.30 -> 10.10.1.5
ICMP TTL:64 TOS:0xC0 ID:8643 IpLen:20 DgmLen:68
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
10.10.1.5:80 -> 192.168.1.13:1419
TCP TTL:64 TOS:0x0 ID:5116 IpLen:20 DgmLen:40 DF
Seq: 0xDCDCl009
(12 more bytes of original packet)
** END OF DUMP

12/12-09:42:01.762837 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:14947 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.1.13:1419
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0xBECHAl9C
(20 more bytes of original packet)
** END OF DUMP

12/12-09:42:22.874992 192.168.2.30 -> 10.10.1.3
ICMP TTL:64 TOS:0xC0 ID:14953 IpLen:20 DgmLen:76
Type:5 Code:1 REDIRECT HOST NEW GW: 192.168.1.13
** ORIGINAL DATAGRAM DUMP:
10.10.1.3:80 -> 192.168.1.13:1423
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
Seq: 0xD2EEF1FA
(20 more bytes of original packet)
** END OF DUMP
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
APPENDIX B

MANUAL AND LISTING PROGRAM

1. Extract of File nips-nids2s3.tar.gz
   # tar xvzf nips-nids2s3.tar.gz

2. Change to the directory nips-nids2s3:
   # cd nips-nids2s3

3. Compile:
   # make

4. Before run program, to see the rule in firewall iptables:
   # iptables -nL

5. If there is rules, flush the rules:
   # iptables -F

6. Run program using command:
   # ./syh -a sdpd

7. See again rules at iptables using command:
   # iptables -nL

Note:

- This software, dataset for data training in file 'data' at directory nips-nid2s3. Change the content from in file 'data' to another data training then run its program and see again the result into iptables firewall rules.

- This software, if using Slackware OS file execute iptables into directory /usr/sbin/iptables, For another distro Linux like Fedora and Ubuntu, file execute iptables into directory /sbin/iptables, so must change path at content source code in ipt.c file and replace become /sbin/iptables and then to compile again.
entr.h

#ifndef _ENTR_H
#define _ENTR_H

#include "lst.h"

float entr_smpl(struct lst_pkt_t *);
float entr_attr(struct lst_grp_intr_t *);
float entr_attr2(struct lst_grp_intr2_t *);

#endif /* _ENTR_H */

/* vim:ts=4:sw=4:tw=80:fdm=marker:cin:
*/
```
#include <stdio.h>
#include <math.h>
#include "lst.h"

/* entr() {{{
float entr(float yes, float no)
{
    float sum;
    if (((yes == 0) || (no == 0)) {
        return 0;
    }
    sum = yes + no;
    return (- (yes / sum) * log2f(yes / sum) - (no / sum) * log2f(no / sum));
}
/* }}} */
/* entr_smpl() {{*/
float entr_smpl(struct lst_pkt_t *head)
{
    struct lst_pkt_t *walk;
    unsigned int intr_y = 0, intr_n = 0;
    walk = head;
    while (walk != NULL) {
        if (walk->intrusion == 0) {
            intr_n++;
        } else {
            intr_y++;
        }
        walk = walk->next;
    }
    return entr(intr_y, intr_n);
}
/* }}*/
/* entr_attr() {{*/
float entr_attr(struct lst_grp_intr_t *head)
{
    struct lst_grp_intr_t *walk;
    float sum, ret;
    /* calculate sample sum */
    sum = 0;
    walk = head;
    while (walk != NULL) {
        sum += walk->yes + walk->no;
        walk = walk->next;
    }
    walk = head;
    ret = 0;
    while (walk != NULL) {
        ret += ((walk->yes + walk->no) / sum) * entr(walk->yes, walk->no);
        walk = walk->next;
    }
    return ret;
}
/* }}*/
/* entr_attr2() {{*/
float entr_attr2(struct lst_grp_intr2_t *head)
{
    struct lst_grp_intr2_t *walk;
    float sum, ret;
```
/* calculate sample sum */
sum = 0;
walk = head;
while (walk != NULL) {
    sum += walk->yes + walk->no;
    walk = walk->next;
}
walk = head;
ret = 0;
while (walk != NULL) {
    ret += ((walk->yes + walk->no) / sum) * entr(walk->yes, walk->no);
    walk = walk->next;
}
return ret;
/* }}) */

/* vim:ts=4:sw=4:tw=80:fdm=marker:cin:
 */
#include <stdio.h>
#include "lst.h"
#include "pkt.h"

/* grp_intr_src() {{{ */
void grp_intr_src(struct lst_pkt_t *pkt_head,
                 struct lst_grp_intr_t **grp_src_head,
                 struct lst_grp_intr_t **grp_src_tail)
{
    struct lst_pkt_t *pkt_walk;
    struct lst_grp_intr_t *grp_src_tmp;
    lst_grp_intr_init(grp_src_head, grp_src_tail);
    pkt_walk = pkt_head;
    while (pkt_walk != NULL) {
        grp_src_tmp = lst_grp_intr_find(*grp_src_head, pkt_walk->src);
        if (pkt_walk->intrusion == 1) {
            if (grp_src_tmp == NULL) {
                lst_grp_intr_add(grp_src_head, grp_src_tail,
                                 pkt_walk->src, 1, 0);
            } else {
                grp_src_tmp->yes++;
            }
        } else {
            if (grp_src_tmp == NULL) {
                lst_grp_intr_add(grp_src_head, grp_src_tail,
                                 pkt_walk->src, 0, 1);
            } else {
                grp_src_tmp->no++;
            }
        }
        pkt_walk = pkt_walk->next;
    }
}
/* }}} */
/* grp_intr_dst() {{{ */
void grp_intr_dst(struct lst_pkt_t *pkt_head,
                 struct lst_grp_intr_t **grp_dst_head,
                 struct lst_grp_intr_t **grp_dst_tail)
{
    struct lst_pkt_t *pkt_walk;
    struct lst_grp_intr_t *grp_dst_tmp;
    lst_grp_intr_init(grp_dst_head, grp_dst_tail);
    pkt_walk = pkt_head;
    while (pkt_walk != NULL) {
        grp_dst_tmp = lst_grp_intr_find(*grp_dst_head, pkt_walk->dst);
        if (pkt_walk->intrusion == 1) {
            if (grp_dst_tmp == NULL) {
                lst_grp_intr_add(grp_dst_head, grp_dst_tail,
                                 pkt_walk->dst, 1, 0);
            } else {
                grp_dst_tmp->yes++;
            }
        } else {
            if (grp_dst_tmp == NULL) {
                lst_grp_intr_add(grp_dst_head, grp_dst_tail,
                                 pkt_walk->dst, 0, 1);
            } else {
                grp_dst_tmp->no++;
            }
        }
    }
}
/* }}} */
pkt_walk = pkt_walk->next;
}
/* }} */

/* grp_intr_proto() {{{
void grp_intr_proto(struct lst_pkt_t *pkt_head,
        struct lst_grp_intr_t **grp_proto_head,
        struct lst_grp_intr_t **grp_proto_tail)
{
    struct lst_pkt_t *pkt_walk;
    struct lst_grp_intr_t *grp_proto_tmp;
    lst_grp_intr_init(grp_proto_head, grp_proto_tail);
    pkt_walk = pkt_head;
    while (pkt_walk != NULL) {
        grp_proto_tmp = lst_grp_intr_find(*grp_proto_head, pkt_walk->proto);
        if (pkt_walk->intrusion == 1) {
            if (grp_proto_tmp == NULL) {
                lst_grp_intr_add(grp_proto_head, grp_proto_tail,
                                    pkt_walk->proto, 1, 0);
            } else {
                grp_proto_tmp->yes++;
            }
        } else {
            if (grp_proto_tmp == NULL) {
                lst_grp_intr_add(grp_proto_head, grp_proto_tail,
                                    pkt_walk->proto, 0, 1);
            } else {
                grp_proto_tmp->no++;
            }
        }
        pkt_walk = pkt_walk->next;
    }
}
/* }}} */

/* grp_intr_sport() {{{
void grp_intr_sport(struct lst_pkt_t *pkt_head,
        struct lst_grp_intr_t **grp_sport_head,
        struct lst_grp_intr_t **grp_sport_tail)
{
    struct lst_pkt_t *pkt_walk;
    struct lst_grp_intr_t *grp_sport_tmp;
    lst_grp_intr_init(grp_sport_head, grp_sport_tail);
    pkt_walk = pkt_head;
    while (pkt_walk != NULL) {
        grp_sport_tmp = lst_grp_intr_find("grp_sport_head, pkt_walk->sport);"}  
        if (pkt_walk->intrusion == 1) {
            if (grp_sport_tmp == NULL) {
                lst_grp_intr_add(grp_sport_head, grp_sport_tail,
                                    pkt_walk->proto, 1, 0);
            } else {
                grp_sport_tmp->yes++;
            }
        } else {
            if (grp_sport_tmp == NULL) {
                lst_grp_intr_add(grp_sport_head, grp_sport_tail,
                                    pkt_walk->proto, 0, 1);
            } else {
                grp_sport_tmp->no++;
            }
        }
        pkt_walk = pkt_walk->next;
    }
/* }}} */
/* grp_intr_dport() {{{ */
void grp_intr_dport(struct lst_pkt_t *pkt_head,
                     struct lst_grp_intr_t **grp_dport_head,
                     struct lst_grp_intr_t **grp_dport_tail)
{
    struct lst_pkt_t *pkt_walk;
    struct lst_grp_intr_t *grp_dport_tmp;

    lst_grp_intr_init(grp_dport_head, grp_dport_tail);
    pkt_walk = pkt_head;
    while (pkt_walk != NULL) {
        grp_dport_tmp = lst_grp_intr_find(*grp_dport_head, pkt_walk->dport);
        if (pkt_walk->intrusion == 1) {
            if (grp_dport_tmp == NULL) {
                lst_grp_intr_add(grp_dport_head, grp_dport_tail,
                                 pkt_walk->dport, 1, 0);
            } else {
                grp_dport_tmp->yes++;
            }
        } else {
            if (grp_dport_tmp == NULL) {
                lst_grp_intr_add(grp_dport_head, grp_dport_tail,
                                 pkt_walk->dport, 0, 1);
            } else {
                grp_dport_tmp->no++;
            }
        }
        pkt_walk = pkt_walk->next;
    }
}
/* }}} */
/* grp_intr_proto_dport() {{{ */
void grp_intr_proto_dport(struct lst_pkt_t *pkt_head,
                           struct lst_grp_intr2_t **grp_proto_dport_head,
                           struct lst_grp_intr2_t **grp_proto_dport_tail)
{
    struct lst_pkt_t *pkt_walk;
    struct lst_grp_intr2_t *grp_proto_dport_tmp;

    lst_grp_intr2_init(grp_proto_dport_head, grp_proto_dport_tail);
    pkt_walk = pkt_head;
    while (pkt_walk != NULL) {
        grp_proto_dport_tmp = lst_grp_intr2_find(*grp_proto_dport_head, 
                                             pkt_walk->proto,
                                             pkt_walk->dport);
        if (pkt_walk->intrusion == 1) {
            if (grp_proto_dport_tmp == NULL) {
                lst_grp_intr2_add(grp_proto_dport_head, grp_proto_dport_tail,
                                   pkt_walk->proto, pkt_walk->dport, 1, 0);
            } else {
                grp_proto_dport_tmp->yes++;
            }
        } else {
            if (grp_proto_dport_tmp == NULL) {
                lst_grp_intr2_add(grp_proto_dport_head, grp_proto_dport_tail,
                                   pkt_walk->proto, pkt_walk->dport, 0, 1);
            } else {
                grp_proto_dport_tmp->no++;
            }
        }
        pkt_walk = pkt_walk->next;
    }
}
#include <stdio.h>
#include "lst.h"
#include "pkt.h"
#include "grp.h"
#include "ipt.h"
#include "entr.h"

/* id3() {{{ */
void id3(char level, struct lst_pkt_t *pkt_head, char shift, unsigned int src,
       unsigned int dst, unsigned int proto, unsigned int dport)
{
  /* vars {{{ */
  struct lst_grp_intr_t *grp_src_head = NULL,
       *grp_src_tail,
  struct lst_grp_intr_t *grp_dst_head = NULL,
       *grp_dst_tail,
  struct lst_grp_intr_t *grp_proto_head = NULL,
       *grp_proto_tail,
  struct lst_grp_intr_t *grp_cmpl_head = NULL,
       *grp_cmpl_tail,
       *grp_walk;

  struct lst_pkt_t *l_pkt_head = NULL,
      *l_pkt_tail;
  float entr_s,
      entr_src,
      entr_dst,
      entr_proto,
  char tmp;

  /* }}} */
  if (level == 0) { /* {{{ */
    /* grouping {{{ */
    grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
    grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
    grp_intr_proto(pkt_head, &grp_proto_head, &grp_proto_tail);
    /* }}} */
    /* calculate entropy and gain {{{ */
    entr_s   = entr_smpl(pkt_head);
    entr_src = entr_attr(grp_src_head);
    entr_dst = entr_attr(grp_dst_head);
    entr_proto = entr_attr(grp_proto_head);
    gain_src   = entr_s - entr_src;
    gain_dst   = entr_s - entr_dst;
    gain_proto = entr_s - entr_proto;
    /* }}} */
    if ((gain_src > gain_dst) && (gain_src > gain_proto)) {
      /* gain source is higest {{{ */
      grp_walk = grp_src_head;
      while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0, 0,
                         &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
          /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
          ipt_blk(grp_walk->data, 0, 0, 0);
        } else {
          id3(1, l_pkt_head, 1, grp_walk->data, 0, 0, 0);
        }
        grp_walk = grp_walk->next;
      }
    }
  }
/* id3() }}*/
}
/* }}} */
} else if ((gain_dst > gain_src) && (gain_dst > gain_proto)) {
  /* gain destination is higest {{{ */
  grp_walk = grp_dst_head;
  while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0, 0, &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
      /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
      ipt_blk(0, grp_walk->data, 0, 0, 0);
    } else {
      id3(1, l_pkt_head, 2, 0, grp_walk->data, 0, 0);
    }
    grp_walk = grp_walk->next;
  }
  /* }}} */
} else {
  /* gain protocol is higest {{{ */
  grp_walk = grp_proto_head;
  while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, 0, 0, grp_walk->data, 0, 0,
                     &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
      /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
      ipt_blk(0, grp_walk->data, 0, 0, 0);
    } else {
      id3(1, l_pkt_head, 4, 0, grp_walk->data, 0, 0);
    }
    grp_walk = grp_walk->next;
  }
  /* }}} */
}
/* }}*/
else if (level == 1) { /* {{{ */
  entr_s = entr_smpl(pkt_head);
  if (shift == 1) {
    /* gain source is higest {{{ */
    /* gain {{{ */
    grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
    grp_intr_proto(pkt_head, &grp_proto_head, &grp_proto_tail);
    gain_proto = entr_s - entr_proto;
    /* }}} */
    if (gain_dst > gain_proto) {
      /* gain destination is higer {{{ */
      grp_walk = grp_dst_head;
      while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0, 0,
                         &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
          /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
          ipt_blk(src, grp_walk->data, 0, 0, 0);
        } else {
          id3(2, l_pkt_head, 3, src, grp_walk->data, 0, 0);
        }
        grp_walk = grp_walk->next;
      }
      /* }}} */
    }
  } else if (level == 2) { /* {{{ */
    entr_s = entr_smpl(pkt_head);
    if (shift == 0) {
      /* gain source is higest {{{ */
      /* gain {{{ */
      grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
      grp_intr_proto(pkt_head, &grp_proto_head, &grp_proto_tail);
      gain_proto = entr_s - entr_proto;
      /* }}} */
      if (gain_dst > gain_proto) {
        /* gain destination is higer {{{ */
        grp_walk = grp_dst_head;
        while (grp_walk != NULL) {
          tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0, 0,
                           &l_pkt_head, &l_pkt_tail);
          if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
          } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(src, grp_walk->data, 0, 0, 0);
          } else {
            id3(2, l_pkt_head, 3, src, grp_walk->data, 0, 0);
          }
          grp_walk = grp_walk->next;
        }
        /* }}} */
      }
    } else { /* }}} */
  } else { /* }}} */
}
&l_pkt_head, &l_pkt_tail);
if (tmp == PKT_ALL_NORM) {
    /* pruning / ignore */
} else if (tmp == PKT_ALL_INTR) {
    ipt_blk(src, 0, grp_walk->data, 0, 0);
} else {
    id3(2, l_pkt_head, 5, src, 0, grp_walk->data, 0);
}
grp_walk = grp_walk->next;
}
/* }}
*/
} else if (shift == 2) {
    /* gain destination is highest {{}} */
    grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
    grp_intr_proto(pkt_head, &grp_proto_head, &grp_proto_tail);
    entr_src = entr_attr(grp_src_head);
    entr_proto = entr_attr(grp_proto_head);
    gain_src = entr_s - entr_src;
    gain_proto = entr_s - entr_proto;
    /* }} */
    if (gain_src > gain_proto) {
        /* gain source is higher {{}} */
        grp_walk = grp_src_head;
        while (grp_walk != NULL) {
            tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0,
                &l_pkt_head, &l_pkt_tail);
            if (tmp == PKT_ALL_NORM) {
                /* pruning / ignore */
            } else if (tmp == PKT_ALL_INTR) {
                ipt_blk(grp_walk->data, dst, 0, 0, 0);
            } else {
                id3(2, l_pkt_head, 3, grp_walk->data, dst, 0, 0);
            }
            grp_walk = grp_walk->next;
        }
    }
    /* }} */
} else {
    /* gain protocol is higher {{}} */
    grp_walk = grp_proto_head;
    while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0,
            &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(0, dst, grp_walk->data, dst, 0, 0);
        } else {
            id3(2, l_pkt_head, 6, 0, dst, grp_walk->data, 0);
        }
        grp_walk = grp_walk->next;
    }
    /* }} */
} else {
    /* gain proto is higest {{}} */
    /* gain {{}} */
    grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
    grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
    entr_src = entr_attr(grp_src_head);
    entr_dst = entr_attr(grp_dst_head);
    gain_src = entr_s - entr_src;
gain_dst = entr_s - entr_dst;
/* */
if (gain_src > gain_dst) {
/* gain source is higher */
grp_walk = grp_src_head;
while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0,
                     &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
        /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
        ipt_blk(grp_walk->data, 0, proto, 0, 0);
    } else {
        id3(2, l_pkt_head, 5, grp_walk->data, 0, proto, 0);
    }
    grp_walk = grp_walk->next;
/* */
} else {
/* gain destination is higher */
grp_walk = grp_dst_head;
while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0, 0,
                     &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
        /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
        ipt_blk(0, grp_walk->data, proto, 0, 0);
    } else {
        id3(2, l_pkt_head, 3, 0, grp_walk->data, proto, 0);
    }
    grp_walk = grp_walk->next;
/* */
} else if (level == 2) {
/* */
if (shift == 3) {
    grp_intr_proto(pkt_head, &grp_proto_head, &grp_proto_tail);
    grp_walk = grp_proto_head;
    while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0,
                         &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(0, grp_walk->data, proto, 0, 0);
        } else {
            id3(3, l_pkt_head, 0, src, dst, grp_walk->data, 0);
        }
        grp_walk = grp_walk->next;
/* */
    } else if (shift == 5) {
    grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
    grp_walk = grp_dst_head;
    while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0,
                         &lpkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(src, grp_walk->data, proto, 0, 0);
        } else {
            id3(3, l_pkt_head, 0, src, dst, grp_walk->data, 0);
        }
        grp_walk = grp_walk->next;
/* */
    }
} else if (shift == 7) {
...
id3(3, _l_pkt_head, 0, src, grp_walk->data, proto, 0);
}
grp_walk = grp_walk->next;
}
} else if (shift == 6) {
    grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
    grp_walk = grp_src_head;
    while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0, &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(grp_walk->data, dst, proto, 0, 0);
        } else {
            id3(3, _l_pkt_head, 0, grp_walk->data, dst, proto, 0);
        }
        grp_walk = grp_walk->next;
    }
    /* }}} */
} else if (level == 3) { /* {{{ */
    grp_intr_dport(pkt_head, &grp_cmpl_head, &grp_cmpl_tail);
    grp_walk = grp_cmpl_head;
    while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, 0, 0, 0, 0, grp_walk->data, &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(src, dst, proto, 0, grp_walk->data);
        } else {
            id3(4, _l_pkt_head, 0, src, dst, proto, grp_walk->data);
        }
        grp_walk = grp_walk->next;
    }
    /* }}} */
} else if (level == 4) { /* {{{ */
    grp_intr_sport(pkt_head, &grp_cmpl_head, &grp_cmpl_tail);
    grp_walk = grp_cmpl_head;
    while (grp_walk != NULL) {
        tmp = pkt_filter(pkt_head, 0, 0, 0, grp_walk->data, dport, &l_pkt_head, &l_pkt_tail);
        if (tmp == PKT_ALL_NORM) {
            /* pruning / ignore */
        } else if (tmp == PKT_ALL_INTR) {
            ipt_blk(src, dst, proto, dport, grp_walk->data);
        }
        grp_walk = grp_walk->next;
    }
    /* }}} */
/* freeing {{{ */
lst_pkt_free(_l_pkt_head);
lst_grp_intr_free(grp_src_head);
lst_grp_intr_free(grp_cmpl_head);
/* }}} */
/* id3_sdpd() {{{ */
void id3_sdpd(char level, struct lst_pkt_t *pkt_head, char shift,
              unsigned int src, unsigned int dst, unsigned int proto,
              unsigned int dport)
{
    /* vars {{{ */
/* }}} */
/* id3_sdpd() }}*/
/* id3_sdpd() */
struct lst_grp_intr_t *grp_src_head = NULL,
    *grp_src_tail,
    *grp_dst_head = NULL,
    *grp_dst_tail,
    *grp_cmpl_head = NULL,
    *grp_cmpl_tail,
    *grp_walk;

struct lst_grp_intr2_t *grp_proto_dport_head = NULL,
    *grp_proto_dport_tail,
    *grp_walk2;

struct lst_pkt_t *l_pkt_head = NULL,
    *l_pkt_tail;

float entr_s,
    entr_src,
    entr_dst,
    entr_proto_dport,
    gain_src,
    gain_dst,
    gain_proto_dport;

char tmp;
/* }}} */
if (level == 0) { /* {{{ */
    /* grouping {{{ */
    grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
    grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
    grp_intr_proto_dport(pkt_head, &grp_proto_dport_head,
                         &grp_proto_dport_tail);
    /* }}} */
    /* calculate entropy and gain {{{ */
    entr_s           = entr_smpl(pkt_head);
    entr_src         = entr_attr(grp_src_head);
    entr_dst         = entr_attr(grp_dst_head);
    entr_proto_dport = entr_attr2(grp_proto_dport_head);
    gain_src         = entr_s - entr_src;
    gain_dst         = entr_s - entr_dst;
    gain_proto_dport = entr_s - entr_proto_dport;
    /* }}} */
    if ((gain_src > gain_dst) && (gain_src > gain_proto_dport)) {
        /* gain source is higest {{{ */
        grp_walk = grp_src_head;
        while (grp_walk != NULL) {
            tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0, 0,
                             &l_pkt_head, &l_pkt_tail);
            if (tmp == PKT_ALL_NORM) {
                /* pruning / ignore */
            } else if (tmp == PKT_ALL_INTR) {
                ipt_blk(grp_walk->data, 0, 0, 0);
            } else {
                id3_sdpd(1, l_pkt_head, 1, grp_walk->data, 0, 0, 0);
            }
            grp_walk = grp_walk->next;
        }
        /* }}} */
    } else if ((gain_dst > gain_src) && (gain_dst > gain_proto_dport)) {
        /* gain destination is higest {{{ */
        grp_walk = grp_dst_head;
        while (grp_walk != NULL) {
            tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0, 0,
                             &l_pkt_head, &l_pkt_tail);
            if (tmp == PKT_ALL_NORM) {
                /* pruning / ignore */
            } else if (tmp == PKT_ALL_INTR) {
                ipt_blk(0, grp_walk->data, 0, 0, 0);
            } else {
                id3_sdpd(1, l_pkt_head, 2, 0, grp_walk->data, 0, 0);
            }
            grp_walk = grp_walk->next;
        }
        /* }}} */
    } else if ((gain_proto_dport > gain_src) && (gain_proto_dport > gain_dst)) {
        /* gain protocol port is higest {{{ */
        grp_walk = grp_proto_dport_head;
        while (grp_walk != NULL) {
            tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0, 0,
                             &l_pkt_head, &l_pkt_tail);
            if (tmp == PKT_ALL_NORM) {
                /* pruning / ignore */
            } else if (tmp == PKT_ALL_INTR) {
                ipt_blk(grp_walk->data, 0, 0, 0);
            } else {
                id3_sdpd(1, l_pkt_head, 3, 0, grp_walk->data, 0, 0);
            }
            grp_walk = grp_walk->next;
        }
        /* }}} */
    } else { /* }}} */

```c
/* }}} */
} else if (level == 1) { /* {{{ */
    entr_s = entr_smpl(pkt_head);
    if (shift == 1) {
        /* gain source is higest {{{ */
        /* gain {{{ */
            grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
            grp_intr_proto_dport(pkt_head, &grp_proto_dport_head, &grp_proto_dport_tail);
            entr_dst = entr_attr(grp_dst_head);
            entr_proto_dport = entr_attr2(grp_proto_dport_head);
            gain_dst = entr_s - entr_dst;
            gain_proto_dport = entr_s - entr_proto_dport;
        /* }}}} */
        if (gain_dst > gain_proto_dport) {
            /* gain destination is higher {{{ */
                grp_walk = grp_dst_head;
                while (grp_walk != NULL) {
                    tmp = pkt_filter(pkt_head, 0, 0, grp_walk->data, 0, grp_walk->datb, &l_pkt_head, &l_pkt_tail);
                    if (tmp == PKT_ALL_NORM) {
                        /* pruning / ignore */
                    } else if (tmp == PKT_ALL_INTR) {
                        ipt_blk(src, grp_walk->data, 0, grp_walk->datb);
                    } else {
                        id3_sdpd(2, l_pkt_head, 3, src, grp_walk->data, 0);
                    }
                    grp_walk = grp_walk->next;
                }
            /* }}}) */
        } else {
            /* gain protocol and dport is higest {{{ */
                grp_walk2 = grp_proto_dport_head;
                while (grp_walk2 != NULL) {
                    tmp = pkt_filter(pkt_head, 0, 0, grp_walk2->data, 0, grp_walk2->datb, &l_pkt_head, &l_pkt_tail);
                    if (tmp == PKT_ALL_NORM) {
                        /* pruning / ignore */
                    } else if (tmp == PKT_ALL_INTR) {
                        ipt_blk(0, 0, grp_walk2->data, 0, grp_walk2->datb);
                    } else {
                        id3_sdpd(1, l_pkt_head, 12, 0, 0, grp_walk2->data, grp_walk2->datb);
                    }
                    grp_walk2 = grp_walk2->next;
                }
                /* }}} */
            } else {
                /* gain protocol and dport is higest {{{ */
                    grp_walk = grp_walk->next;
                    /* }}} */
                } else {
                    /* gain protocol and dport is higest {{{ */
                        grp_walk2 = grp_proto_dport_head;
                        while (grp_walk2 != NULL) {
                            tmp = pkt_filter(pkt_head, 0, 0, grp_walk2->data, 0, grp_walk2->datb, &l_pkt_head, &l_pkt_tail);
                            if (tmp == PKT_ALL_NORM) {
                                /* pruning / ignore */
                            } else if (tmp == PKT_ALL_INTR) {
                                ipt_blk(src, 0, grp_walk2->data, 0, grp_walk2->datb);
                            } else {
                                } else {
```
id3_sdpd(2, l_pkt_head, 13, src, 0, grp_walk2->data,
grp_walk2->datb);
}
grp_walk2 = grp_walk2->next;
/* }}*/
/* }}*/
/* }}*/
if (shift == 2) {
/* gain destination is highest {{{ */
/* gain {{{ */
grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
grp_intr_proto_dport(pkt_head, &grp_proto_dport_head, 
&grp_proto_dport_tail);
entr_src = entr_attr(grp_src_head);
entr_proto_dport = entr_attr2(grp_proto_dport_head);
gain_src = entr_s - entr_src;
gain_proto_dport = entr_s - entr_proto_dport;
/* }}*/

if (gain_src > gain_proto_dport) {
/* gain source is higher {{{ */
grp_walk = grp_src_head;
while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0, 0,
&l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) { /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
        ipt_blk(grp_walk->data, dst, 0, 0, 0);
    } else {
        id3_sdpd(2, l_pkt_head, 3, grp_walk->data, dst, 0, 
    }
}
grp_walk = grp_walk->next;
/* }}*/
} else {
/* gain protocol and dport is higher {{{ */
grp_walk2 = grp_proto_dport_head;
while (grp_walk2 != NULL) {
    tmp = pkt_filter(pkt_head, 0, 0, grp_walk2->data, 0,
&l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) { /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
        ipt_blk(0, dst, grp_walk2->data, 0, grp_walk2->datb);
    } else {
        id3_sdpd(2, l_pkt_head, 14, 0, dst, grp_walk2->data,
grp_walk2->datb);
    }
}
grp_walk = grp_walk->next;
/* }}*/
} else {
/* gain protocol and dport is highest {{{ */
/* gain {{{ */
grp_intr_src(pkt_head, &grp_src_head, &grp_src_tail);
grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
entr_src = entr_attr(grp_src_head);
entr_dst = entr_attr(grp_dst_head);
gain_src = entr_s - entr_src;
gain_dst = entr_s - entr_dst;
/* */
if (gain_src > gain_dst) {
  /* gain source is higher */
  grp_walk = grp_src_head;
  while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0,
                     &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
      /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
      ipt_blk(grp_walk->data, 0, proto, 0, dport);
    } else {
      id3_sdpd(2, l_pkt_head, 13, grp_walk->data, 0,
               dport);
    }
    grp_walk = grp_walk->next;
  }
}
/* */
else {
  /* gain destination is higher */
  grp_walk = grp_dst_head;
  while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0,
                     &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
      /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
      ipt_blk(0, grp_walk->data, proto, 0, dport);
    } else {
      id3_sdpd(2, l_pkt_head, 14, 0, grp_walk->data,
               dport);
    }
    grp_walk = grp_walk->next;
  }
}
/* */
} else if (level == 2) {
  if (shift == 3) {
    grp_intr_proto_dport(pkt_head, &grp_proto_dport_head,
                         &grp_proto_dport_tail);
    grp_walk2 = grp_proto_dport_head;
    while (grp_walk2 != NULL) {
      tmp = pkt_filter(pkt_head, 0, 0, grp_walk2->data, 0,
                       grp_walk2->datb, &l_pkt_head, &l_pkt_tail);
      if (tmp == PKT_ALL_NORM) {
        /* pruning / ignore */
      } else if (tmp == PKT_ALL_INTR) {
        ipt_blk(src, dst, grp_walk2->data, 0, grp_walk2->datb);
      } else {
        id3_sdpd(3, l_pkt_head, src, dst, grp_walk2->data,
                 grp_walk2->datb);
      }
      grp_walk2 = grp_walk2->next;
    }
  } else if (shift == 13) {
    grp_intr_dst(pkt_head, &grp_dst_head, &grp_dst_tail);
    grp_walk = grp_dst_head;
    while (grp_walk != NULL) {
    }
tmp = pkt_filter(pkt_head, 0, grp_walk->data, 0, 0, &l_pkt_head, &l_pkt_tail);
if (tmp == PKT_ALL_NORM) {
    /* pruning / ignore */
} else if (tmp == PKT_ALL_INTR) {
    ipt_blk(src, grp_walk->data, proto, 0, dport);
} else {
    id3_blockd(3, l_pkt_head, 0, src, grp_walk->data, proto, dport);
}
grp_walk = grp_walk->next;
}
}
/* }} */
} else if (level == 3) {
/* {{{ */
grp_intr_sport(pkt_head, &grp_cmpl_head, &grp_cmpl_tail);
grp_walk = grp_cmpl_head;
while (grp_walk != NULL) {
    tmp = pkt_filter(pkt_head, grp_walk->data, 0, 0, 0, grp_walk->data, &l_pkt_head, &l_pkt_tail);
    if (tmp == PKT_ALL_NORM) {
        /* pruning / ignore */
    } else if (tmp == PKT_ALL_INTR) {
        ipt_blk(grp_walk->data, dst, proto, 0, dport);
    } else {
        id3_blockd(3, l_pkt_head, 0, grp_walk->data, dst, proto, dport);
    }
    grp_walk = grp_walk->next;
}
/* }} */
/* freeing {{{ */
lst_pkt_free(l_pkt_head);
lst_grp_intr_free(grp_src_head);
lst_grp_intr_free(grp_dst_head);
lst_grp_intr_free(grp_proto_dport_head);
lst_grp_intr_free(grp_cmpl_head);
/* }} */
/* vim:ts=4:sw=4:tw=80:fdm=marker:cin:
 */
```c
#include <stdio.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/wait.h>

/* ipt_blk() {{{ */
void ipt_blk(unsigned int src, unsigned int dst, unsigned int proto,
             unsigned int sport, unsigned int dport)
{
    struct protoent *protoe;
    char saddr[INET_ADDRSTRLEN];
    char *env[] = {NULL};
    char **argv;
    char *arg;
    char buff[11];
    int idx;
    struct in_addr iaddr;
    pid_t pid;
    argv = (char **) malloc(16 * sizeof(char *));
    arg = strdup("iptables");
    argv[0] = arg;
    arg = strdup("-A");
    argv[1] = arg;
    arg = strdup("FORWARD");
    argv[2] = arg;
    idx = 2;

    argv = (char **) malloc(16 * sizeof(char *));
    arg = strdup("iptables");
    argv[0] = arg;
    arg = strdup("-A");
    argv[1] = arg;
    arg = strdup("FORWARD");
    argv[2] = arg;
    idx = 2;

    if (src) {
        iaddr.s_addr = src;
        inet_ntop(AF_INET, &iaddr, saddr, INET_ADDRSTRLEN);
        idx++;
        arg = strdup("-s");
        argv[idx] = arg;
        idx++;
        argv[idx] = strdup(saddr);
    }
    if (dst) {
        iaddr.s_addr = dst;
        inet_ntop(AF_INET, &iaddr, saddr, INET_ADDRSTRLEN);
        idx++;
        arg = strdup("-d");
        argv[idx] = arg;
        idx++;
        argv[idx] = strdup(saddr);
    }
    if (proto) {
        protoe = getprotobynumber(proto);
        idx++;
        arg = strdup("-p");
        argv[idx] = arg;
        idx++;
        argv[idx] = strdup(protoe->p_name);
    }
    if (sport) {
```
idx++;  
arg = strdup("--sport");
argv[idx] = arg;
idx++;  
sprintf(buff, "%d", sport);
arg = strdup(buff);
argv[idx] = arg;
}
if (dport) {
idx++;  
arg = strdup("--dport");
argv[idx] = arg;
idx++;  
sprintf(buff, "%d", dport);
arg = strdup(buff);
argv[idx] = arg;
}
idx++;  
arg = strdup("-j");
argv[idx] = arg;
idx++;  
arg = strdup("DROP");
argv[idx] = arg;
idx++;  
argv[idx] = NULL;
pid = fork();
if (pid == 0) {  
/* child */
  execute("/usr/sbin/iptables", argv, env);
  exit(EXIT_FAILURE);
} else if (pid > 0) {  
/* parent */
  wait(NULL);
}
for (idx = 0; idx < 16; idx++) {
  if (argv[idx] == NULL) {
    break;
  }
}
free(argv[idx]);
}
free(argv);
*/
/* ipt_reset() {{}} */
void ipt_reset()
{
  char *argv[] = {"iptables", "-F", "FORWARD", NULL};
  char *env[] = {NULL};
  pid_t pid;

  pid = fork();
  if (pid == 0) {
    /* child */
    execute("/usr/sbin/iptables", argv, env);
    exit(EXIT_FAILURE);
  } else if (pid > 0) {
    /* parent */
    wait(NULL);
  }
}
#ifndef _LST_H
#define _LST_H

struct lst_grp_intr_t {
    unsigned int data, yes, no;
    struct lst_grp_intr_t *next;
};

struct lst_grp_intr2_t {
    unsigned int data, datb, yes, no;
    struct lst_grp_intr2_t *next;
};

struct lst_pkt_t {
    unsigned int src, dst, proto, sport, dport, intrusion;
    struct lst_pkt_t *next;
};

void lst_grp_intr_init(struct lst_grp_intr_t **, struct lst_grp_intr_t **);
void lst_grp_intr_add(struct lst_grp_intr_t **, struct lst_grp_intr_t **,
unsigned int, unsigned int, unsigned int);
void lst_grp_intr_view(struct lst_grp_intr_t *);
struct lst_grp_intr_t *lst_grp_intr_find(struct lst_grp_intr_t *, unsigned int);
void lst_grp_intr_free(struct lst_grp_intr_t *);

void lst_grp_intr2_init(struct lst_grp_intr2_t **, struct lst_grp_intr2_t **);
void lst_grp_intr2_add(struct lst_grpintr2_t **, struct lst_grp_intr2_t **,
unsigned int, unsigned int, unsigned int, unsigned int);
void lst_grp_intr2_view(struct lst_grp_intr2_t *);
struct lst_grp_intr2_t *lst_grp_intr2_find(struct lst_grp_intr2_t *,
unsigned int, unsigned int);
void lst_grp_intr2_free(struct lst_grp_intr2_t *);

void lst_pkt_init(struct lst_pkt_t **, struct lst_pkt_t **);
void lst_pkt_add(struct lst_pkt_t **, struct lst_pkt_t **, unsigned int,
unsigned int, unsigned int, unsigned int, unsigned int, unsigned int);
void lst_pkt_view(struct lst_pkt_t *);
void lst_pkt_free(struct lst_pkt_t *);

#endif /* _LST_H */
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <arpa/inet.h>
#include "lst.h"

/* lst_grp_intr_init() {{{ */
void lst_grp_intr_init(struct lst_grp_intr_t **head, struct lst_grp_intr_t **tail)
{
    *head = *tail = NULL;
}
/* }}} */

/* lst_grp_intr_add() {{{ */
void lst_grp_intr_add(struct lst_grp_intr_t **head, struct lst_grp_intr_t **tail,
                      unsigned int data, unsigned int yes, unsigned int no)
{
    struct lst_grp_intr_t *tmp;
    tmp = (struct lst_grp_intr_t *) malloc(sizeof(struct lst_grp_intr_t));
    tmp->data = data;
    tmp->yes  = yes;
    tmp->no   = no;
    if (*head == NULL) {
        *head = *tail = tmp;
    } else {
        (*tail)->next = tmp;
        *tail = (*tail)->next;
    }
    tmp->next = NULL;
}
/* }}} */

/* lst_grp_intr_view() {{{ */
void lst_grp_intr_view(struct lst_grp_intr_t *head)
{
    struct lst_grp_intr_t *walk;
    walk = head;
    while (walk != NULL) {
        printf("%d:%d:%d:%d\n", walk->data, walk->yes, walk->no,
                   walk->yes + walk->no);
        walk = walk->next;
    }
}
/* }}} */

/* lst_grp_intr_find() {{{ */
struct lst_grp_intr_t *lst_grp_intr_find(struct lst_grp_intr_t *head,
                                         unsigned int search)
{
    struct lst_grp_intr_t *walk;
    walk = head;
    while ((walk != NULL) && (walk->data != search)) {
        walk = walk->next;
    }
    return walk;
}
/* }}} */

/* lst_grp_intr_free() {{{ */
void lst_grp_intr_free(struct lst_grp_intr_t *head)
{
    struct lst_grp_intr_t *walk;
    walk = head;
while (walk != NULL) {
    head = head->next;
    free(walk);
    walk = head;
}
/*}}}*/
/* lst_grp_intr2_init() {{{*
void lst_grp_intr2_init(struct lst_grp_intr2_t **head,
                     struct lst_grp_intr2_t **tail)
{
    *head = *tail = NULL;
}/*}}}*/
/* lst_grp_intr2_add() {{{*
void lst_grp_intr2_add(struct lst_grp_intr2_t **head,
                     struct lst_grp_intr2_t **tail,
                     unsigned int data, unsigned int datb,
                     unsigned int yes, unsigned int no)
{
    struct lst_grp_intr2_t *tmp;
    tmp = (struct lst_grp_intr2_t *) malloc(sizeof(struct lst_grp_intr2_t));
    tmp->data = data;
    tmp->datb = datb;
    tmp->yes = yes;
    tmp->no = no;
    if (*head == NULL) {
        *head = *tail = tmp;
    } else {
        (*tail)->next = tmp;
        *tail = (*tail)->next;
    }
    tmp->next = NULL;
}/*}}}*/
/* lst_grp_intr2_view() {{{*
void lst_grp_intr2_view(struct lst_grp_intr2_t *head)
{
    struct lst_grp_intr2_t *walk;
    walk = head;
    while (walk != NULL) {
        printf("%d:%d:%d:%d:%d
", walk->data, walk->datb, walk->yes, walk->no,
             walk->yes + walk->no);
        walk = walk->next;
    }
}/*}}}*/
/* lst_grp_intr2_find() {{{*
struct lst_grp_intr2_t *lst_grp_intr2_find(struct lst_grp_intr2_t *head,
                                           unsigned int search,
                                           unsigned int searci)
{
    struct lst_grp_intr2_t *walk;
    walk = head;
    while (walk != NULL) {
        if ((walk->data != search) || (walk->datb != searci)) {
            walk = walk->next;
        } else {
            return walk;
        }
    }
}/*}}}*/
/* lst_grp_intr2_free() {{{*/
void lst_grp_intr2_free(struct lst_grp_intr2_t *head)
{
    struct lst_grp_intr2_t *walk;
    walk = head;
    while (walk != NULL) {
        head = head->next;
        free(walk);
        walk = head;
    }
}

/* }}} */
/* lst_pkt_init() {{{ */
void lst_pkt_init(struct lst_pkt_t **head, struct lst_pkt_t **tail)
{
    *head = *tail = NULL;
}
/* }}} */
/* lst_pkt_add() {{{ */
void lst_pkt_add(struct lst_pkt_t **head, struct lst_pkt_t **tail,
                 unsigned int src, unsigned int dst, unsigned int proto,
                 unsigned int sport, unsigned int dport, unsigned int intrusion)
{
    struct lst_pkt_t *tmp;
    tmp = (struct lst_pkt_t *) malloc(sizeof(struct lst_pkt_t));
    tmp->src       = src;
    tmp->dst       = dst;
    tmp->proto     = proto;
    tmp->sport     = sport;
    tmp->dport     = dport;
    tmp->intrusion = intrusion;
    if (*head == NULL) {
        *head = *tail = tmp;
    } else {
        (*tail)->next = tmp;
        *tail = (*tail)->next;
    }
    tmp->next = NULL;
}
/* }}} */
/* lst_pkt_view() {{{ */
void lst_pkt_view(struct lst_pkt_t *head)
{
    struct lst_pkt_t *walk;
    struct protoent *proto;
    struct in_addr src, dst;
    char psrc[INET_ADDRSTRLEN], pdst[INET_ADDRSTRLEN];
    walk = head;
    while (walk != NULL) {
        proto = getprotobynumber(walk->proto);
        src.s_addr = walk->src;
        inet_ntop(AF_INET, &src, psrc, INET_ADDRSTRLEN);
        dst.s_addr = walk->dst;
        inet_ntop(AF_INET, &dst, pdst, INET_ADDRSTRLEN);
        printf("%s|%s:%d->%s:%d%c\n", proto->p_name, psrc, walk->sport, pdst, walk->dport, (walk->intrusion) ? '+' : '-');
        walk = walk->next;
    }
}
/* }}} */
/* lst_pkt_free() {{{ */
void lst_pkt_free(struct lst_pkt_t *head)
{
    struct lst_pkt_t *walk;
walk = head;
while (walk != NULL) {
    head = head->next;
    free(walk);
    walk = head;
}

/* }}} */
/* vim:ts=4:sw=4:tw=80:fdm=marker:cin:
*/
#ifndef _PKT_H
#define _PKT_H

#include "lst.h"

#define PKT_NO_MATCH 0 /* no packet match */
#define PKT_ALL_NORM 1 /* all packet(s) is normal (there is no intrusion) */
#define PKT_ALL_INTR 2 /* all packet(s) is intrusion */
#define PKT_NORM_INTR 3
#define PKT_INTR_NORM 3

char pkt_filter(struct lst_pkt_t *, unsigned int, unsigned int, unsigned int,
                unsigned int, unsigned int, struct lst_pkt_t **,
                struct lst_pkt_t **);

#endif /* _PKT_H */
#include <stdio.h>
#include "pkt.h"
#include "lst.h"

/* pkt_filter() {{{ */
char pkt_filter(struct lst_pkt_t *pkt_head, unsigned int src, unsigned int dst,
                  unsigned int proto, unsigned int sport, unsigned int dport,
                  struct lst_pkt_t **n_pkt_head, struct lst_pkt_t **n_pkt_tail)
{
    struct lst_pkt_t *pkt_walk;
    int pkt_intr_exists = 0, pkt_norm_exists = 0;
    pkt_walk = pkt_head;
    lst_pkt_init(n_pkt_head, n_pkt_tail);
    while (pkt_walk != NULL) {
        if ((!(src   || (src   == pkt_walk->src))   &&
             !(dst   || (dst   == pkt_walk->dst))   &&
             !(proto || (proto == pkt_walk->proto)) &&
             !(sport || (sport == pkt_walk->sport)) &&
             !(dport || (dport == pkt_walk->dport))) {
            if (pkt_walk->intrusion == 1) {
                pkt_intr_exists = 1;
            } else {
                pkt_norm_exists = 1;
            }
            lst_pkt_add(n_pkt_head, n_pkt_tail, pkt_walk->src, pkt_walk->dst,
                         pkt_walk->proto, pkt_walk->sport, pkt_walk->dport,
                         pkt_walk->intrusion);
        }
        pkt_walk = pkt_walk->next;
    }
    if (*n_pkt_head == NULL) {
        return PKT_NO_MATCH;
    }
    if (!pkt_intr_exists) {
        return PKT_ALL_NORM;
    } else if (!pkt_norm_exists) {
        return PKT_ALL_INTR;
    }
    return PKT_NORM_INTR;
}
/* }}*/ */
main.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <getopt.h>
#include "lst.h"
#include "grp.h"
#include "id3.h"
#include "pkt.h"
#include "ipt.h"

/* main() {{{ */
int main(int argc, char **argv)
{
    /* vars {{{ */
    FILE *fp;         /* file pointer */
    char buff[256],   /* buffer */
    *p;          /* (help char) pointer */
    struct lst_pkt_t *pkt_head, /* (list) packet head */
    *pkt_tail; /* (list) packet tail */
    struct in_addr src, /* source address */
    dst; /* destination address */
    struct protoent *proto; /* protocol */
    unsigned int sport, /* source port */
    dport, /* destination port */
    intrusion;

    int opt, opt_idx = 0;
    struct option opt_long[] = {
        {"reset",    0, 0, 'r'},
        {"version", 0, 0, 'v'},
        {"debug",    0, 0, 'd'},
        {"log-packet", 1, 0, 'p'},
        {"log-intrusion", 1, 0, 'i'},
        {"file-time-reference", 1, 0, 'f'},
        {"algorithm", 1, 0, 'a'},
        {"usage",    0, 0, 'u'},
        {"help",     0, 0, 'h'},
        {"schedule", 1, 0, 's'},
        {0, 0, 0, 0}
    };
    int debug = 0, reset, schedule = 0, algo = 0;
    char *file_packet = NULL,
    *file_intr = NULL,
    *file_ref = NULL;

    /* function prototype {{{ */
    void show_version();
    void show_usage();

    /* }}} */
    /* parse option {{{ */
    while ((opt = getopt_long(argc, argv, "rvVduha:p:i:f:s:", opt_long, &opt_idx)) != -1) {
        switch (opt) {
            case 'r':
                reset = 1;
                break;
            case 'v':
                show_version();
                exit(EXIT_SUCCESS);
            case 'd':
                break;
        }
    } /* }}} */
} /* main() */

/* main() }}}
int main(int argc, char **argv) { /*
    /* vars {{{ */
    FILE *fp;         /* file pointer */
    char buff[256],   /* buffer */
    *p;          /* (help char) pointer */
    struct lst_pkt_t *pkt_head, /* (list) packet head */
    *pkt_tail; /* (list) packet tail */
    struct in_addr src, /* source address */
    dst; /* destination address */
    struct protoent *proto; /* protocol */
    unsigned int sport, /* source port */
    dport, /* destination port */
    intrusion;

    int opt, opt_idx = 0;
    struct option opt_long[] = {
        {"reset",    0, 0, 'r'},
        {"version", 0, 0, 'v'},
        {"debug",    0, 0, 'd'},
        {"log-packet", 1, 0, 'p'},
        {"log-intrusion", 1, 0, 'i'},
        {"file-time-reference", 1, 0, 'f'},
        {"algorithm", 1, 0, 'a'},
        {"usage",    0, 0, 'u'},
        {"help",     0, 0, 'h'},
        {"schedule", 1, 0, 's'},
        {0, 0, 0, 0}
    };
    int debug = 0, reset, schedule = 0, algo = 0;
    char *file_packet = NULL,
    *file_intr = NULL,
    *file_ref = NULL;

    /* function prototype {{{ */
    void show_version();
    void show_usage();

    /* }}} */
    /* parse option {{{ */
    while ((opt = getopt_long(argc, argv, "rvVduha:p:i:f:s:", opt_long, &opt_idx)) != -1) {
        switch (opt) {
            case 'r':
                reset = 1;
                break;
            case 'v':
                show_version();
                exit(EXIT_SUCCESS);
            case 'd':
                break;
        }
    } /* }}} */
} /* main() */

/* main() {{*/
int main(int argc, char **argv) {
    /* vars {{{ */
    FILE *fp;         /* file pointer */
    char buff[256],   /* buffer */
    *p;          /* (help char) pointer */
    struct lst_pkt_t *pkt_head, /* (list) packet head */
    *pkt_tail; /* (list) packet tail */
    struct in_addr src, /* source address */
    dst; /* destination address */
    struct protoent *proto; /* protocol */
    unsigned int sport, /* source port */
    dport, /* destination port */
    intrusion;

    int opt, opt_idx = 0;
    struct option opt_long[] = {
        {"reset",    0, 0, 'r'},
        {"version", 0, 0, 'v'},
        {"debug",    0, 0, 'd'},
        {"log-packet", 1, 0, 'p'},
        {"log-intrusion", 1, 0, 'i'},
        {"file-time-reference", 1, 0, 'f'},
        {"algorithm", 1, 0, 'a'},
        {"usage",    0, 0, 'u'},
        {"help",     0, 0, 'h'},
        {"schedule", 1, 0, 's'},
        {0, 0, 0, 0}
    };
    int debug = 0, reset, schedule = 0, algo = 0;
    char *file_packet = NULL,
    *file_intr = NULL,
    *file_ref = NULL;

    /* function prototype {{{ */
    void show_version();
    void show_usage();

    /* }}} */
    /* parse option {{{ */
    while ((opt = getopt_long(argc, argv, "rvVduha:p:i:f:s:", opt_long, &opt_idx)) != -1) {
        switch (opt) {
            case 'r':
                reset = 1;
                break;
            case 'v':
                show_version();
                exit(EXIT_SUCCESS);
            case 'd':
                break;
        }
    } /* }}} */
} /* main() */

/* main() {{*/
int main(int argc, char **argv) {
    /* vars {{{ */
    FILE *fp;         /* file pointer */
    char buff[256],   /* buffer */
    *p;          /* (help char) pointer */
    struct lst_pkt_t *pkt_head, /* (list) packet head */
    *pkt_tail; /* (list) packet tail */
    struct in_addr src, /* source address */
    dst; /* destination address */
    struct protoent *proto; /* protocol */
    unsigned int sport, /* source port */
    dport, /* destination port */
    intrusion;

    int opt, opt_idx = 0;
    struct option opt_long[] = {
        {"reset",    0, 0, 'r'},
        {"version", 0, 0, 'v'},
        {"debug",    0, 0, 'd'},
        {"log-packet", 1, 0, 'p'},
        {"log-intrusion", 1, 0, 'i'},
        {"file-time-reference", 1, 0, 'f'},
        {"algorithm", 1, 0, 'a'},
        {"usage",    0, 0, 'u'},
        {"help",     0, 0, 'h'},
        {"schedule", 1, 0, 's'},
        {0, 0, 0, 0}
    };
    int debug = 0, reset, schedule = 0, algo = 0;
    char *file_packet = NULL,
    *file_intr = NULL,
    *file_ref = NULL;

    /* function prototype {{{ */
    void show_version();
    void show_usage();

    /* }}} */
    /* parse option {{{ */
    while ((opt = getopt_long(argc, argv, "rvVduha:p:i:f:s:", opt_long, &opt_idx)) != -1) {
        switch (opt) {
            case 'r':
                reset = 1;
                break;
            case 'v':
                show_version();
                exit(EXIT_SUCCESS);
            case 'd':
                break;
        }
    } /* }}} */
} /* main() */
debug = 1;
break;
case 'h':
case 'u':
    show_usage();
    exit(EXIT_SUCCESS);
case 's':
    schedule = atoi(optarg);
    break;
case 'p':
    file_packet = strdup(optarg);
    break;
case 'v':
    file_intr = strdup(optarg);
    break;
case 'f':
    file_ref = strdup(optarg);
    break;
case 'a':
    if (strcmp(optarg, "sdpd", 4) == 0) {
        algo = 1;
    } else if (strcmp(optarg, "sdp", 3) == 0) {
        algo = 0;
    } else {
        show_usage();
        exit(EXIT_SUCCESS);
    }
    break;
}
/* }}} */
lst_pkt_init(&pkt_head, &pkt_tail);
/* read data {{{
fp = fopen("data", "r");
while (fgets(buff, 255, fp) != NULL) {
    p = strtok(buff, ":"); /* number */
    p = strtok(NULL, ":"); /* src */
    inet_pton(AF_INET, p, &src);
    p = strtok(NULL, ":"); /* dst */
    inet_pton(AF_INET, p, &dst);
    p = strtok(NULL, ":"); /* sport */
    sport = (unsigned int) atoi(p);
    p = strtok(NULL, ":"); /* dport */
    dport = (unsigned int) atoi(p);
    p = strtok(NULL, ":"); /* proto */
    proto = getprotobyname(p);
    p = strtok(NULL, ":"); /* intrusion */
    intrusion = (strncasecmp(p, "yes", 3) == 0) ? 1 : 0;
    lst_pkt_add(&pkt_head, &pkt_tail, src.s_addr, dst.s_addr,
                proto->p_proto, sport, dport, intrusion);
}
fclose(fp);
/* }}} */
ipt_reset();
if (algo != 0) {
    id3(0, pkt_head, 0, 0, 0, 0, 0);
} else {
    id3_sdpd(0, pkt_head, 0, 0, 0, 0, 0);
}
lst_pkt_free(pkt_head);
if (file_packet != NULL) free(file_packet);
if (file_ref != NULL) free(file_ref);
if (file_intr != NULL) free(file_intr);
return EXIT_SUCCESS;
void show_version()
{
    printf("NIPS-NID2S3 version 0.1\n");
}

void show_usage()
{
    printf("Usage: syh [--algorithm] [-a [sdp|sdpd]]\n");
}