AN INITIAL DESIGN OF FIREWALL INFORMATION EXCHANGE PROTOCOL
(FIEP)

Sandeep Reddy Pedditi
B.S., Jawaharlal Nehru Technological University, India, 2009

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

SPRING
2012
AN INITIAL DESIGN OF FIREWALL INFORMATION EXCHANGE PROTOCOL (FIEP)

A Project

by

Sandeep Reddy Pedditi

Approved by:

______________________________, Committee Chair
Du Zhang, Ph.D.

______________________________, Second Reader
Chung-E Wang, Ph.D.

______________________________
Date
Student: Sandeep Reddy Pedditi

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

__________________________, Graduate Coordinator       ________________
Nikrouz Faroughi, Ph.D.                                      Date

Department of Computer Science
Abstract

of

AN INITIAL DESIGN OF FIREWALL INFORMATION EXCHANGE PROTOCOL (FIEP)

by

Sandeep Reddy Pedditi

The FIEP is a mechanism that enables firewalls to communicate with each other’s firewalls and form firewall groups in a network. The information the firewalls communicate with each other would improve their ability to detect any attack and thus protects the network from attack. The FIEP also improves the ability to adapt to changes in the network, informing other firewalls when there is an attack, it also informs about an update in the access control rules in the firewalls in a secure way. Besides this, the FIEP keeps all the firewalls in the group informed about the activity going on in the group such as messaging the entire group about a new firewall joining in or moving away from the group etc. This improves the ability to detect any attack in a more efficient way.

In the current scenario, there is no protocol that enables firewalls to communicate with each other and exchange information. Until recently, not much thought was given to the need for firewalls to talk to each other; firewalled network is isolated from the rest of the network and considered to be secure. But that is not true firewalled network is safe but not totally secure it is prone to distributed attacks. To overcome this drawback, I propose
the FIEP, using which firewalls can talk to each other and exchange information. The FIEP is like the Border Gateway Protocol (BGP) which enables routers to exchange routing information and keeps them updated. Similarly, FIEP will enable the firewalls to update firewall rules, form groups and alert the other firewall in the network about attacks, this method will improve the security and increase the robustness of the network.

The Goal of this project is to create an initial design of the FIEP which specifies how the firewalls interact with each other and how they can be formed into groups. In version 1 of FIEP, I intend to show the detailed steps involved in communication with other firewalls, for example what type of connection is required, TCP or UDP, how these connections should be established, its requirements and what information will be exchanged e.g. access control rules and establishing a group such as having a lead firewall which will maintain the group information etc. To design FIEP an example network with firewalls in it will be designed first and this example network will be used throughout the project and to finally aid in designing the FIEP. The project will also show current best practices in firewall deployment. The FIEP will be a breakthrough in not just the Network Security domain but will also pave the way for firewall communications. Future and extensive study in this regard can help improve the current problems that the networks face.

______________________
Committee Chair
Du Zhang, Ph.D.

______________________
Date
I would like to take this opportunity to thank all the people who have been involved in this project, without whose professional knowledge, guidance and encouragement, this project would not have been successfully completed.

I would like to profoundly thank my project advisors, Dr. Du Zhang and Dr. Chung-E Wang, for giving me an opportunity to work under their guidance. Dr. Du Zhang’s constant support and valuable feedback gave shape to my project.

A special thanks to my friend Tina John, for helping me manage my time and work things through a tight schedule. Lastly and most importantly, I would like to thank my family for encouraging me to purse my Masters.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Firewalls</td>
<td>1</td>
</tr>
<tr>
<td>1.2. What is Firewall Inconsistency?</td>
<td>1</td>
</tr>
<tr>
<td>2. BACKGROUND</td>
<td>5</td>
</tr>
<tr>
<td>2.1. Evolution of Firewalls</td>
<td>5</td>
</tr>
<tr>
<td>2.2. Research on Firewall Issues</td>
<td>6</td>
</tr>
<tr>
<td>3. FIEP SPECIFICATIONS</td>
<td>9</td>
</tr>
<tr>
<td>3.1. Design of FIEP</td>
<td>9</td>
</tr>
<tr>
<td>3.2. FIEP Modes</td>
<td>13</td>
</tr>
<tr>
<td>3.3. FIEP States</td>
<td>14</td>
</tr>
<tr>
<td>3.4. Finite State Machine of FIEP</td>
<td>32</td>
</tr>
</tbody>
</table>

vii
3.5. Firewall Groups in FIEP ................................................................. 39
3.6. FIEP Fall Back Mechanism ............................................................ 42
4. FIEP SIMULATION ............................................................................ 43
  4.1. Simulation Environment ............................................................... 43
  4.2. How to Run Simulation? ............................................................... 44
  4.3. Simulation Output ....................................................................... 48
5. CONCLUSIONS AND FUTURE WORK ........................................... 61
  5.1. Conclusion .................................................................................. 61
  5.2. Future Work ............................................................................... 62
Appendix ............................................................................................... 63
  Source Code ...................................................................................... 63
  Bibliography ...................................................................................... 85
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1 Idle State</td>
<td>23</td>
</tr>
<tr>
<td>Table 2 Connect State</td>
<td>24</td>
</tr>
<tr>
<td>Table 3 Echo State</td>
<td>24</td>
</tr>
<tr>
<td>Table 4 Report Attack State</td>
<td>26</td>
</tr>
<tr>
<td>Table 5 Report Rule Changed State</td>
<td>26</td>
</tr>
<tr>
<td>Table 6 Attack Respond State</td>
<td>27</td>
</tr>
<tr>
<td>Table 7 Rule Changed Response State</td>
<td>28</td>
</tr>
<tr>
<td>Table 8 Queue Description</td>
<td>29</td>
</tr>
<tr>
<td>Table 9 Database Table MyPro Description</td>
<td>44</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Figure 1 Example of Static Configuration</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2 Internal FSM of Echo State</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3 FSM of FIEP</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4 FSP Firewall Grouping</td>
<td>40</td>
</tr>
<tr>
<td>Figure 5 Simulation Network</td>
<td>46</td>
</tr>
<tr>
<td>Figure 6 Simulation Scenario One</td>
<td>49</td>
</tr>
<tr>
<td>Figure 7 Simulation Scenario Two</td>
<td>50</td>
</tr>
<tr>
<td>Figure 8 Simulation Scenario Three</td>
<td>52</td>
</tr>
<tr>
<td>Figure 9 Simulation Scenario Four</td>
<td>53</td>
</tr>
<tr>
<td>Figure 10 Simulation Scenario Five</td>
<td>53</td>
</tr>
<tr>
<td>Figure 11 Simulation Scenario Six</td>
<td>54</td>
</tr>
<tr>
<td>Figure 12 Simulation Scenario Seven</td>
<td>55</td>
</tr>
<tr>
<td>Figure 13 Simulation Scenario Eight</td>
<td>56</td>
</tr>
<tr>
<td>Figure 14 Simulation Scenario Nine</td>
<td>57</td>
</tr>
<tr>
<td>Figure 15 Simulation Scenario Ten</td>
<td>58</td>
</tr>
<tr>
<td>Figure 16 Simulation Scenario Eleven</td>
<td>59</td>
</tr>
</tbody>
</table>
1.1. Firewalls

Firewall is one of the key tools in the cyber security of any organization. This important tool has evolved from the late 80s to the present time. In this evolution, the key role of firewalls hasn’t changed much. The First generation of firewalls implemented stateless packet filters for networks, the Second generation improved to incorporate stateful packet filters and in Third generation, we went a step ahead with application level firewalls. The primary goal of firewalls is packet filtering, apart from this feature remaining common among all generations of firewalls, the other commonly shared feature was that of firewall issues such as Firewall Inconsistency and Inability to detect and react to attacks.

1.2. What is Firewall Inconsistency?

Access Control Lists (ACLs) play the key role in governing a firewall. It is the ACL that determines what packets the firewall should allow to pass through and what packets to block. A network engineer configures the ACLs into a firewall, an ACL can go on to be
several hundreds of lines large. The larger the size, the more the possibility of there being inconsistencies in the ACL. For e.g. Example 1: ACL rule 100 might instruct the firewall to allow a particular packet but rule 1999 might state that the same packet be blocked. The firewall, being designed to consider the first rule it strikes, will go ahead and let the packet pass through, without checking the other rule that asks for it to be blocked.

Example 2: There is also a possibility of redundant rules existing, due to which the time to process the ACL rules increases for the Firewall. Many ideas and solutions have been proposed to overcome the firewall inconsistencies but there is a problem on a larger scale, known as Network Inconsistency that needs to be considered as well; an example of which could be explained by the following scenario: Consider 2 firewalls A and B such that firewall A is the root firewall and firewall B is the node. If firewall A has an ACL that instructs it to block out a particular set of packets but firewall B is instructed to allow those set of packets, it gets into a scenario known as Network Inconsistency.

To explain my next point on firewalls, I will use a scenario based approach; in this scenario, we will consider each network as a house and a firewall as the gate to the house. Just like a gate was designed to keep out unwanted entities and permit only few, a firewall was designed to work the same, thus ensuring the safety of the house aka network. But as time passed, just like folk began breaking into houses, networks too have been exposed to the threat of being attacked by hackers. To keep thieves away, gates were equipped with automated alarm systems and secure locking systems, firewalls too if
equipped with such a design, can work more efficiently in securing the network. Firewalls if designed to have a basic mind of their own, can detect an attack and alert the firewalls surrounding it so that the intruder will not just be shut out of one firewall but his access to the network will be completely cut out.

I thus propose the idea of taking firewalls to its next generation, only this time making sure that the issues of Firewall Inconsistency and inability to detect and react to attacks, is resolved. I would like to present the ‘Firewall Information Exchange Protocol’ (FIEP), an approach that has been designed to enable firewalls to communicate with each other so that they can share information relevant to detecting and fighting against an attack on the network.

The FIEP is a mechanism that enables Firewalls to communicate with each other, and form firewall groups in a network. The information the firewalls communicate with each other would improve their ability to detect any attack and thus protects the network from being compromised. The FIEP also improves the ability to adapt to changes in the network, informing other firewalls when there is an attack, it also informs about an update in the access control rules in the firewalls in a secure way. Besides this, the FIEP keeps all the firewalls in the group informed about the activity going on in the group such as messaging the entire group about a new firewall joining in or moving away from the group etc. This improves the ability to detect any attack in a more efficient way.
In the current practice, there is no protocol that enables firewalls to communicate with each other and exchange information. Until recently, not much thought was given to the need for firewalls to talk to each other; firewalled network is isolated from the rest of the network and considered to be secure. Yet it is not true to assert that a firewalled network is safe, it can still be prone to distributed attacks. To overcome this drawback, I propose the FIEP, using which firewalls can talk to each other and exchange information. The FIEP is like the Border Gateway Protocol (BGP) which enables routers to exchange routing information and keeps them updated. Similarly, FIEP will enable the firewalls to update firewall rules, form groups and alert the other firewall in the network about attacks, this method will improve the security and increase the robustness of the network.

The Goal of this project is to create an initial design of the FIEP which specifies how the firewalls interact with each other. Version 1 of FIEP, will show the detailed steps involved in communication with other firewalls, for example what type of connection is required, TCP or UDP, how these connections should be established, its requirements and what information will be exchanged e.g. access control rules and establishing a group such as having a lead firewall which will maintain the group information etc. We use an example network with firewalls throughout the project to explain how FIEP works. The project will also show current best practices in firewall deployment. The fully developed FIEP will be a breakthrough in not just the Network Security domain but will also pave the way for firewall communications.
Chapter 2

BACKGROUND

2.1. Evolution of Firewalls

During the early years, after firewalls were introduced, a lot of research was done to understand how best to use firewalls and the defensive strategies that could be implemented by them. Authors had different perceptions of firewalls and used various analogies to properly explain firewalls, one analogy that still is mentioned in regard to firewalls, was given by Dr. William Hancock, a well-known firewall expert, he describes firewalls as: “The concept [of security barriers] is much like that of the strong castle being protected by a series of moats around the castle. As the storming hoards gets close to the castle, they must traverse the series of moats. It is possible to traverse some moats with pole vault activities, but eventually the leaper of the moat is bound to fall into one of the moats and is caught. If there is only one moat and the leaper is good, there is not much protection. If there are moats, concertina wire, razor wire, tall fences with broken glass on them, land mines, cans full of pennies suspended by trip wires, Doberman pinschers and other such traps in the path from the intruder to the "jewels," one or more of the obstacles is going to alert the keepers of the castle that someone is trying to infiltrate the castle and something must be done to protect the assets and destroy the
intruder.” Author Tom Sheldon in his paper, “Firewall White Paper”, quotes Dr. Hancock and uses his analogy to explain firewalls, he rightly points out that “While the storming hoard analogy might be appropriate in some cases, the real threat is often the stealthy spy who slips over walls in the dark of night and scales every barrier undetected to reach his target of attack.”

2.2. Research on Firewall Issues

In the words of authors Mohamed G. Gouda and Alex X. Liu, “A firewall is a security guard placed at the point of entry between a private network and the outside Internet such that all incoming and outgoing packets have to pass through it. The function of a firewall is to examine every incoming or outgoing packet and decide whether to accept or discard it.” Simple in definition but complex in function, firewalls have certainly evolved over the years. From a single firewall guarding one network, to multiple firewalls fencing a bigger and complex network, firewalls have grown in efficiency and ability. With growth, came the complications of ensuring that the firewalls worked appropriately and were able to incorporate all the rules that they needed to be imbibed in them. Thus the design of a firewall depends completely on a sequence of rules, simple as it sounds, these

2 “General Firewall White Paper” by Tom Sheldon
3 “General Firewall White Paper” by Tom Sheldon
4 “Structured Firewall Design”, Mohamed G Gouda and Alex X Liu, April 2006.
Available online: www.cs.utexas.edu/~gouda/papers/journal/Firewall.pdf
set of rules brought with them new issues. Several years of research and study led to
white papers and projects being done, to understand firewalls and their functionality so
that their efficiency could be extended and the issues eliminated.

A white paper, “Structured firewall design”, by authors Mohamed G. Gouda and Alex X.
Liu, outlines these issues in a clear and concise manner; “The current practice of
designing a firewall directly as a sequence of rules suffers from three types of major
problems:

- The consistency problem, which means that it is difficult to order the rules
correctly;
- The completeness problem, which means that it is difficult to ensure thorough
consideration for all types of traffic;
- The compactness problem, which means that it is difficult to keep the number of
rules small (because some rules may be redundant and some rules may be
combined into one rule)”

Algorithms and tools were implemented over the years, to help solve the
inconsistency issue. Algorithms by authors S. Pozo, R. Ceballos, R. M. Gasca, were

5 “Structured Firewall Design”, Mohamed G Gouda and Alex X Liu, April 2006.
Available online: www.cs.utexas.edu/~gouda/papers/journal/Firewall.pdf
proposed to identify the inconsistencies in firewalls while others were written to eliminate the inconsistencies\(^6\).

Research on firewalls and the work of several authors raised a question in my mind; “Even with years of research and so many strong solutions, why does the Network inconsistency issue still linger?” To fix this network inconsistency issue, I propose the FIEP. This is a new protocol that chalks out an initial design that proposes a means of firewall communication. Using this protocol, firewalls will communicate network inconsistency issues and distributed attack information, thus forming a stronger, well informed barrier against attacks posed on it.

Chapter 3

FIEP SPECIFICATIONS

FIEP is a TCP/IP based protocol which provides a communication mechanism for 2 or more firewalls participating in the FIEP to communicate with each other. This design of the FIEP is intended to solve the 2 problems introduced in chapter 1 of the paper, i.e. Network Inconsistency in Firewalls and Inability to detect and respond to distributed attacks.

3.1. Design of FIEP

There are 2 aspects that need to be considered in the design of the FIEP, i) the individual components that participate in the FIEP; ii) The connection between these components. Like any other TCP IP network protocols, FIEP has common states like Idle state and Connect state. Firewalls are considered to be in the Idle state by default. The protocol starts from the Idle state and transitions to the connect state, the other states that follow are; Echo state, Report Attack state, Report Rule Changed state, Attack Respond state and Rule Changed Respond state. In the connect state it establishes a connection with other firewalls participating in FIEP. In order to establish a connection with the other firewalls, first they need to be identified; there are 2 ways of discovering other firewalls participating in FIEP;
FIEP Configuration:

Like any other protocol configuration, the FIEP configuration is a startup or boot data of the protocol. This boot data or configuration will help the FIEP determine what it should do once the FIEP starts.

1. Static Configuration;
2. Dynamic Configuration.

1. **Static Configuration:**

In this method, a firewall relies on the FIEP configuration to discover the firewalls it can establish a connection with, these firewalls could either be its peers or children firewalls. The network engineer has to assign the roles of parent, peer and children firewalls to each firewall in the network, he then loads this information into a Configuration file. This file is thus referred by the firewall when the FIEP starts, in order for it to identify its peers or children firewalls and establish a connection with them. This is the process of Static Configuration.

**Firewall Levels**

- **Same Level:** One or more firewalls connected to a common network node (firewall or router) through which the data flows uplink and downlink, are said to be on the same level.
- **Level Below:** One or more firewalls connected to the network node (firewall or router) are said to be on the level below if the data flows through the network node to reach the firewall both for uplink and downlink flow.

- **Level Above:** A Firewall connected to the network node (firewall or router) is said to be on the level above, if the data flows through the firewall to reach the network node both for uplink and downlink flow.

We categorize firewalls as parents, peers and children, based on the following criteria:

A peer firewall is a firewall which is on the same level in the network, as the firewall communicating with it.

A child firewall is a firewall which is on a level below in the network, compared to the firewall communicating with it.

A parent firewall is a firewall which is one level above other firewalls connected to directly to it in a network.

A firewall can be a parent and a child at the same time.

The figure below, explains the different firewalls types:

- Parent firewall - F1 is a parent of F2, F3 and F4
  - F2 is a parent of F5

- Peer firewall - F2, F3 and F4 are Peers

- Child firewall - F2, F3 and F4 are children of F1
F5 is a child of F2

F2 is thus a parent of F5 and a child of F1

Figure 1: Example of Static Configuration

2. **Dynamic Configuration:**

   In a Dynamic Connection, the firewall sends a message to all the other firewalls participating in the FIEP. The firewall will send an “Open Connection” message to its peers, children and parents. The firewalls that received the Open Connection message will reply back with a “Ready to Connect” message if they are willing to participate in the FIEP else no message will be sent. Irrespective of the response to the firewall
from its peers or children, its state changes from Connect state to Echo state. This method is known as Dynamic Configuration.

3.2. FIEP Modes

A FIEP Mode is an operational manner in which FIEP states perform their task. Modes in FIEP are not switchable like in an operating system during runtime (e.g. switching user mode to kernel mode). Modes in FIEP tell the firewalls what they should do when an event occurs. FIEP modes are a part of the FIEP configuration. To change the FIEP modes, the FIEP configuration file must be changed. When firewalls communicate with each other by sending and receiving messages, the extent of the firewalls’ communication depends on one of the three modes of the FIEP. The three modes in which the FIEP can run are listed below:

1. **High mode:** In this mode, the firewalls participating in the FIEP will be highly alert, keeping track of every change and/or events (e.g. ACL changes and attacks), and also notifying all its peers and children whenever any event happens.

2. **Normal mode:** In this mode, the firewalls participating in FIEP will notify about events, only to the necessary participating parties.

3. **Low mode:** In this mode, the firewalls participating in FIEP will notify only about critical events, to the necessary participating parties.

The IDS determines the intensity of an event that is to help identify its criticality.
3.3. FIEP States

The FIEP has a set of 7 states that the firewalls transitions into, depending on what the FIEP is trying to accomplish. Each state is designed to perform a specific task which helps the firewalls to communicate with each other. Listed below are the 7 states of the FIEP:

1. Idle State:

   The Idle state is the first state in the FIEP. When the FIEP starts, the firewall is in the Idle state by default. In this state the firewall check for any errors in its FIEP configuration. If an error is found in the configuration; the FIEP will fail to start. Once the configuration check is done without any errors, the firewall will move to the next state i.e. the Connect State.

2. Connect State:

   A firewall transitions into the Connect State from the Idle state. In the Connect State, the firewall will check the FIEP configuration to see if it is a static configuration or dynamic configuration. Depending on the type of configuration, the firewall will send a “Ready to Connect” message to the participating firewalls, e.g. If the configuration type is static, then the firewall will send the “ready to connect” message to its peer, child and/or parent if any, the information about peer, child and parent is obtained from the FIEP configuration. The firewall will then wait for a response from the connecting firewalls
till a timeout occurs (a timeout implies a lapse of approximately 5 secs without any response from the connecting firewalls). If the firewall receives an Acknowledgement response from the firewalls that it sent the message to, it sends the information of these firewalls to a “connected stack”. If a timeout occurs and the firewall has not received an acknowledgement from the other firewalls it sent the message to, it puts the information of the firewalls that it had an unsuccessful connection attempt with, in a "not connected stack" to retry to establish the connection later.

Once the firewall accomplishes the tasks listed above, either by receiving a response from its peers, children and/or parent firewalls or by facing a timeout, the connected firewall will move to next state, i.e. Echo State.

The following is a summary of the step by step happenings in the Connect State:

- The firewall that transitions from Idle state to Connect state, obtains the list of peer, child and parent firewalls from the FIEP configuration file present in the firewall, if in static configuration mode else it obtains the list of peer, child and parent dynamically by sending an “open connection” message to zero hop firewalls in the dynamic mode.
- Send “ready to connect” request to peer, child and parent if any.
- Wait for response from connecting firewall till a timeout occurs.
If connecting firewalls’ response is “connected”, put the connecting firewall information in the list of “connected” peer, child or parent.

If connecting firewall doesn’t respond and a timeout occurs, put the connecting firewall information in the list of “not connected” peer, child or parent, this list is used to re-establish a connection with the not connected peer, child and parent firewalls later.

The firewalls transition from the Connect state to the Echo state. This transition can be reversed as well (details listed in the Echo state). If the firewall transitions from Echo state back to the Connect state, it obtains the list of peer, child and parent from list of “not connected” peer, child and parent and continues the process involved in the Connect state.

3. Echo State:

A firewall transitions from Connect state to the Echo state. In the Echo state, the firewall will send an “echo” message after a specific time (e.g. 5 seconds), to the firewalls participating in the FIEP with it. The firewall also monitors itself for any ACL changes and to check if there are any attacks on the firewall (these attacks are reported by IDS). An assumption is made that the firewall is connected to an IDS which monitors all the packets flow and whenever there is an attack on the firewall, it will notify the firewall. The firewall then goes into a listening mode, whenever there is a message from the participating firewalls, an appropriate state transition is made.
**Event 1:** No previous event in the Event Queue

**Event 2:** After sending “echo” message to the connected firewall and receiving an “echo” message from the connected firewalls or after a timeout.

**Event 3:** Timeout completed in Listening mode of the connected firewall.

**Event 3:** Timeout completed in Listening mode of IDS.

**Event 4:** Connected Firewall sent message i.e. other than “echo”.

**Event 5:** IDS Reported Attack on firewall.

**Event 6:** Firewall detected ACL change.

**Event 7:** No State change required.

**Event 8:** State change required.
The following is a summary of the step by step happenings in the Echo State:

- In this state, the firewall will go into the listening mode. After a timeout, the firewall will send an “echo” message to its connected firewalls (i.e. peer, child and parent if any). In the listening mode, the firewall remains dormant and listens for messages from its connected firewalls, these messages will be any of the messages listed below:

  o Listens for “echo” message from its connected firewalls.

  o Listens for “Attack Reported” message from its connected firewalls (this depends on the mode in which the FIEP is running). Once the message is received, the firewall puts the message information in the “RA” Queue (i.e. Respond to Attack queue) of peer, child or parent from which the “attack reported” message was received. The “RA” queue is used to send an acknowledgement. The FIEP state is now changed to the “Attack Respond” state. Firewall can be configured to respond or not to respond, depending on FIEP modes (i.e. high, medium, low).

  o Listens for “Rule Changed Reported” message from its connected firewall (this depends on the mode in which the FIEP is running). Once the message is sent, the firewall puts the message information in the “RCR” Queue (i.e. “Rule Changed Respond” queue) of peer, child or parent from which the “rule changed reported” message was received. The RCR queue
is used to send acknowledgements. The FIEP state now changes to “Rule Changed Response” state. Firewall can be configured to respond or not to respond depending on FIEP modes (i.e. high, medium, low); if the firewall wants to respond, the state will be changed to “Response Rule Change”.

- Listens for attack messages from the IDS. Whenever IDS reports an attack to the firewall, depending on the mode in which the FIEP is running, the firewall state will be changed to “Report Attack” state.
- The firewall checks itself, for any changes in its ACL. If the ACLs are changed, the firewall state will be changed to the “Report Rule Changed” state.
- Listens for the “Response to Attack” message, from the “AR” queue.
- Listens for the “response to rule change” message, from “RC” queue.
- Listens for warnings from connected firewalls.
- The firewall will remain in this state till its state changes when an event occurs.
- Any error in this state, leads to the FIEP state being changed to the “Idle” state.

4. Report Attack State:

When the IDS reports an attack to the firewall; its state changes from Echo state to Report Attack state. In this state, the firewall will report the attack that was reported by the IDS to its peer, child and parent if any, depending on the FIEP mode (i.e. high, medium, low). The firewall will send an “Attack Reported” message to its peer, child and/or parent if any are available.
The following is a summary of the step by step happenings in the Report Attack State:

- The firewall checks the mode in which the FIEP is running;
- The firewall will select the list of peer, child and/or parent from the list of connected peer, child and parent firewalls. This selection is based on the mode in which the FIEP is running. The selections of peer, child and/or parent can also be pre-configured, that is static configuration; in which case, the firewall can ignore the mode in which the FIEP is running. This gives more flexibility to send the “Attack Reported” message to the desired firewalls.
- List of firewall peer, child and parent to which the “Attack Reported” message has been sent is placed in a Queue called “AR” (Attack reported Queue). This queue is used to get an acknowledgement message (“Response to attack”) from the firewalls that the “Attack Reported” message was sent to.
- Any error in this state, leads to the FIEP state being changed to the “Idle” state.
- If no errors occur, the FIEP state is changed to Echo state.

5. Report Rule Changed State:

In the Echo state, the firewall checks itself for any changes in its ACL. If the ACLs are changed, the firewall state will be changed to the “Report Rule Changed” state. In this state, the firewall will report ACL changes if any, to its child, based on the mode in which the FIEP is running.
The following is a summary of the step by step happenings in the Report Rule Changed State:

- The firewall checks the mode in which the FIEP is running;
- If the mode allows the firewall to report an ACL change, the firewall will send a “Rule Changed Reported” message to its child.
- The list of child firewalls to which the “Rule Changed Reported” message has been sent are placed in a Queue “RC” (Rule Changed Queue). This queue is used to get acknowledgements (“Response Rule Change”) from the firewalls that the “Rule Changed Reported” message was sent to.
- Any error in this state, leads to the FIEP state being changed to the “Idle” state.
- If no error occurs, the FIEP state is changed to Echo state.

6. Attack Respond State:

In the Echo state, the firewall listens for an “Attack Reported” message from its connected firewalls (this depends on the mode in which the FIEP is running). Once the message is received, the firewall puts the message information in the “RA” Queue (i.e. Respond to Attack queue) of peer, child or parent from which the “attack reported” message was received. The queue “RA” is used to send an acknowledgement. The FIEP state is now changed to the “Attack Respond” state. In this state, the firewall will respond (sends acknowledgement) to the list of firewalls from the queue “RA”.
The following is a summary of the step by step happenings in the Report Rule Changed State:

- The firewall obtains the attack message from the “RA” queue. It analyzes the message and identifies if any ACL changes are required. If required, it makes the respective ACL changes, e.g. If the message from the “RA” queue contains the IP address of the attacker (e.g. 192.168.24.30 - attacker’s IP), this IP will be blocked by updating the ACL, if required.
- After the message is processed, the acknowledgement is sent back to the sender. The sender information is obtained from the “RA” queue.
- Any error in this state, leads to the FIEP state being changed to the “Idle” state.
- If no errors occur, the FIEP state is changed to Echo state.

7. Rule Changed Response State:

In the Echo state, the firewall listens for “Rule Changed Reported” message from its connected firewall (this depends on the mode in which the FIEP is running). Once the message is sent, the firewall puts the message information in the “RCR” Queue (i.e. “Rule Changed Respond” queue) of peer, child or parent from which the “rule changed reported” message was received. The RCR queue is used to send acknowledgements. The FIEP state now changes to “Rule Changed Response” state. In this state, the firewall will respond (sends acknowledgement) to the list of firewalls from the queue “RCR”.
The following is a summary of the step by step happenings in the Report Rule Changed State:

- The firewall obtains the ACL rule change message from the “RCR” queue. It checks for network inconsistency, this is done by comparing blocked ACL rule of parent with allow ACL rule of child, e.g. If the message from parent contains “Block 192.168.24.30” and the child ACL rule has “Allow 192.168.24.30” this is a network inconsistency and a warning message is generated.
- After the messages are processed, the acknowledgement is sent back to the sender. The sender information is obtained from the “RCR” queue.
- Any error in this state, leads to the FIEP state being changed to the “Idle” state.
- If no errors occur, the FIEP state is changed to Echo state.

Tables 1 to 7 describe the various activities that go on in the seven states of FIEP running in the high, medium and low mode.

**Idle State**

<table>
<thead>
<tr>
<th>Table 1: Idle State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIEP High Mode</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Check the FIEP Configuration</td>
</tr>
</tbody>
</table>
Connect State

Table 2: Connect State

<table>
<thead>
<tr>
<th></th>
<th>FIEP High Mode</th>
<th>FIEP Normal Mode</th>
<th>FIEP Low Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to peer, child, and parent if any.</td>
<td>Connect to peer, child, and parent if any.</td>
<td>Connect to peer, child, and parent if any.</td>
<td></td>
</tr>
<tr>
<td>In Static Configuration, information about peer, child and parent comes from the configuration file.</td>
<td>In Static Configuration, information about peer, child and parent comes from the configuration file.</td>
<td>In Static Configuration, information about peer, child and parent comes from the configuration file.</td>
<td></td>
</tr>
<tr>
<td>In Dynamic Configuration, information about peer, child and parent is determined in connect state.</td>
<td>In Dynamic Configuration, information about peer, child and parent is determined in connect state.</td>
<td>In Dynamic Configuration, information about peer, child and parent is determined in connect state.</td>
<td></td>
</tr>
</tbody>
</table>

Echo State

Table 3: Echo State

<table>
<thead>
<tr>
<th></th>
<th>FIEP High Mode</th>
<th>FIEP Normal Mode</th>
<th>FIEP Low Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listens for “echo” message</td>
<td>Listens for “echo” message</td>
<td>Listens for “echo” message</td>
<td></td>
</tr>
<tr>
<td>from its connected firewall.</td>
<td>from its connected firewall.</td>
<td>from its connected firewall.</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td></td>
</tr>
<tr>
<td>Listens for “Attack Reported” message from peer, child and parent.</td>
<td>Listens for “Attack Reported” message from peer, and child.</td>
<td>Listens for “Attack Reported” message from child.</td>
<td></td>
</tr>
<tr>
<td>Listens for “Rule Change Reported” message from parent.</td>
<td>Listens for “Rule Change Reported” message from parent.</td>
<td>Listens for “Rule Change Reported” message from parent.</td>
<td></td>
</tr>
<tr>
<td>Check for ACL change and reports all changes to child.</td>
<td>Check for ACL change and reports all changes to child.</td>
<td>Check for ACL change and reports all changes to child.</td>
<td></td>
</tr>
<tr>
<td>Listens for “Attack Respond” message from all firewall in “AR” queue.</td>
<td>Listens for “Attack Respond” message from all firewall in “AR” queue.</td>
<td>Listens for “Attack Respond” message from all firewall in “AR” queue.</td>
<td></td>
</tr>
</tbody>
</table>
Listens for warning message from all connected firewall

Listens for warning message from all connected firewall

Listens for warning message from all connected firewall

Report Attack State

Table 4: Report Attack State

<table>
<thead>
<tr>
<th>Mode</th>
<th>FIEP High Mode</th>
<th>FIEP Normal Mode</th>
<th>FIEP Low Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Listens for “Attack Reported” message from peer, child and parent.</td>
<td>Listens for “Attack Reported” message from peer and child.</td>
<td>Listens for “Attack Reported” message from child.</td>
</tr>
<tr>
<td></td>
<td>List of firewall peer, child and parent to which the “Attack Reported” message has been sent are placed in a Queue “AR” (Attack reported Queue).</td>
<td>List of firewall peer and child to which the “Attack Reported” message has been sent are placed in a Queue “AR” (Attack reported Queue).</td>
<td>List of firewall child to which the “Attack Reported” message has been sent are placed in a Queue “AR” (Attack reported Queue).</td>
</tr>
</tbody>
</table>

Report Rule Changed State

Table 5: Report Rule Changed State

<table>
<thead>
<tr>
<th>Mode</th>
<th>FIEP High Mode</th>
<th>FIEP Normal Mode</th>
<th>FIEP Low Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If the mode allows the</td>
<td>If the mode allows the firewall</td>
<td>If the mode allows the</td>
</tr>
</tbody>
</table>
A firewall is designed to report an ACL change, the firewall will send a “Rule Changed” message to its child. The list of firewall child to which the “Rule Changed” message has been sent are placed in a Queue “RC” (Rule Changed Queue). This queue is used to get acknowledgements (“Response to rule change”) from the firewalls that the “Rule Changed” message was sent to.

### Attack Respond State

**Table 6: Attack Respond State**

<table>
<thead>
<tr>
<th>FIEP High Mode</th>
<th>FIEP Normal Mode</th>
<th>FIEP Low Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>The firewall obtains the</td>
<td>The firewall obtains the</td>
<td>The firewall obtains the</td>
</tr>
</tbody>
</table>
attack message from the “RA” queue. It analyzes the message and identifies if any ACL changes are required. The message is processed, and the acknowledgement is sent back to the sender. The sender information is obtained from the “AR” queue.

Table 7: Rule Changed Response State

<table>
<thead>
<tr>
<th>FIEP High Mode</th>
<th>FIEP Normal Mode</th>
<th>FIEP Low Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>The firewall obtains the ACL rule change message from the “RCR” queue. It checks for network inconsistency.</td>
<td>The firewall obtains the ACL rule change message from the “RCR” queue. It checks for network inconsistency.</td>
<td>The firewall obtains the ACL rule change message from the “RCR” queue. It checks for network inconsistency.</td>
</tr>
</tbody>
</table>
The messages are processed, and the acknowledgement is sent back to the sender. The sender information is obtained from the “RCR” queue.

Table 8: Queue Description

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue description</th>
<th>State Used In</th>
</tr>
</thead>
</table>
| Respond to Attack Queue (RA queue) | The RA queue is used to send acknowledgement message to “Attack Report” message received from peer, child or parent. In echo state data will be pushed to RA queue and in Attack Respond state data will be popped from the RA queue | • Echo State  
  • Attack Respond State |
<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule Changed Respond Queue (RCR queue)</td>
<td>The RCR queue is used to send acknowledgement message to “Report Rule Change” message from parent. In echo state data will be pushed to RCR queue and in Rule Changed Respond state data will be popped from the RCR queue</td>
<td>• Echo State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rule Change Respond state</td>
</tr>
<tr>
<td>Attack reported Queue (AR queue)</td>
<td>The AR queue is used to listens acknowledgement message to “Attack Reported” message which was sent to peer, child or parent. In Report Attack State data will be pushed to AR queue and in echo state data will be popped from</td>
<td>• Report Attack State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Echo State</td>
</tr>
</tbody>
</table>
| Rule Changed Queue (RC queue) | The RC queue is used to listens for acknowledgement message to “Report Rule Changed” message from child. In Report Rule Change State data will be pushed to RC queue and in the echo state data will be popped from RC queue. | • Report Rule Change State  
• Echo State |
3.4. Finite State Machine of FIEP

![Finite State Machine of FIEP](Image)

**Figure 3**: FSM of FIEP
• Event 1: Error occurred in this state

• Event 2: FIEP configuration check in Idle state is successful

• Event 3: Firewall tried to connect to peer, child and/or parent present in the configuration. Either the connection is successful or unsuccessful, after a timeout or after all successful connections with peer, child and/or parent. State transitions from Connect state to Echo state.

• Event 4: No external or internal event occurred. Firewall remains in Echo state.

• Event 5: A firewall trying to establish the connection with not connected firewalls from the configuration. State transitions from Echo state to Connect state.

• Event 6: IDS reported an attack on the Firewall. State transitions from Echo to Report Attack state.

• Event 7: IDS reported attack is taken care and message has been sent to peer, child and/or parent depending on the FIEP mode. State transitions from Report Attack to Echo state.

• Event 8: “Attack Reported” message received from the connected peer, child and/or parent. State transitions from Echo state to Attack Respond state.

• Event 9: Attack reported by peer, child and/or parent is taken care and acknowledgement is sent back to the reported firewall depending on the FIEP running mode. State transitioned from Attack Respond to Echo state.

• Event 10: Firewall identified and ACL changed. State transitions from Echo to Report Rule Change state.
• Event 11: Firewall will send the “Rule Changed Reported” message to its child. 
  State transitions from Report Rule Change state to Echo state.

• Event 12: “Rule Changed Reported” message received from parent. State 
  transitions from Echo to Rule Change Respond state.

• Event 13: Firewall will check for network inconsistency between the parent and 
  itself, and generates a warning message if necessary. State transitions from Rule 
  Change Respond state to Echo state.
Let’s take a closer look at each state and the transition from one state to another;

1. **Idle State**
   - Refuses all incoming FIEP connections
   - Start event triggers the initialization of the protocol data
   - Initiates a TCP connection with configured FIEP peers and FIEP child
   - Listens for TCP connection with its peers and child
   - Changes its state to Connect.
   - If an error occurs at any state of the FSM process, the FIEP session is terminated immediately and returned to Idle state.
   - Some of the reasons why a firewall does not progress from the Idle state are:
     - Peer address configured incorrectly on either firewall

2. **Connect state:**
   - Waits for successful TCP negotiation with peer and children.
   - FIEP does not spend much time in this state if the TCP session has been successfully established.
   - Sends PeConnect, CConnect, PaConnect message to peer, child and parent respectively.
   - On the receipt of acknowledgement of connection from peers and child, the state is changed to Echo.
• In case of any unsuccessful attempts to connect, the FIEP will try to restart another TCP session with the peer and child.

3. Echo state:

• If the firewall was unable to establish a successful TCP session, then it ends up in the Echo state.

• FIEP FSM will change the state to Connect and will try to establish a connection with firewalls that are not connected.

• FIEP FSM listens for messages from its peer and child.

• Once the message has been received, the firewall checks the validity of the message and depending on the message, it changes its state;
  
  o If the message is ‘Attack Reported’, the FIEP will change to ‘Attack Respond’ state
  
  o If the message is ‘Rule Changed’, the FIEP will change to ‘Rule Change Respond’ state

• Each individual firewall will also monitor for attacks and ACL (or Rules) changed on itself. Depending on the events (i.e. either attack or rule change) the FIEP state gets changed
  
  o If the event is an attack reported by the IDS (Intrusion Detection System), the FIEP state is changed to ‘Report Attack’.
o If the event is an ACL or Rule change, the FIEP state is changed to ‘Report Rule changed’.

- FIEP listens for traffic block alerts from children.

4. Report Attack state:

- A firewall reaches this state when the IDS reports an attack.
- Depending on the priority level, the FIEP will notify its peer and child;
  - High priority: Notifies peers and children
  - Medium priority: Notifies only peers
  - Low priority: Notifies only critical attacks to peers
  - The nature of an attack (i.e. Critical, high, medium and low) is determined from the message received from the IDS.
- Based on the IDS message information, if the rules are updated, the state is changed to Report Rule Changed else the state is then changed to Echo.

5. Attack Respond state:

- A firewall reaches this state when a peer or parent reports an attack.
- Based on the attack information received from the peer or parent, rules are updated if needed.
If the rules are changed, the state is changed to Rule Changed else the state is changed to Echo

6. Report Rule Changed state:

- A firewall reaches this state if rules are changed.
- A message is sent to its child to check if the new rules are blocking any of its traffic.
- The state is changed to Echo.

7. Rule Changed Respond state:

- A firewall reaches this state when it receives a “Rule Changed” message from its parent.
- The child checks if the parent is blocking any of its traffic.
- If the traffic is being blocked, an alert is sent back to the parent along with the traffic details.
- If there is no block in the traffic, a “Traffic OK” message is sent to the parent.
- The state is changed to Echo.
3.5. Firewall Groups in FIEP

In FIEP, firewalls can form groups and communicate with firewalls within the same group. These firewall groups will help decompose the network and define different security mechanisms within the groups. One firewall can be part of multiple groups. In FIEP, firewall grouping is done by using static configuration, where information of the groups that the firewall belongs to are defined within the configuration file present in the firewall. All the firewalls in the FIEP are grouped only based on the 3 categories of peer, child and parent; there is no mechanism of grouping other than this.

Example of Groups in FIEP

Figure 3 shows two groups of firewalls participating in FIEP and there is a common firewall in both groups. The two groups of firewalls are Group A and Group B:

- Group A contains
  - F1, F2, F4, F5, F6

- Group B contains
  - F2, F3, F6, F7

- Common firewall in both Group A and Group B
  - F2 and F6
Static configuration of firewall

- Firewall F1
  - Peer: F2
  - Child: F4, F5, F6
  - Parent: N/A

- Firewall F2
o Peer: F1,F3
o Child: N/A
o Parent: N/A

- Firewall F3
  o Peer: F2
  o Child:F6,F7
  o Parent: N/A

- Firewall F4 & F5
  o Peer: N/A
  o Child :N/A
  o Parent: F1

- Firewall F6
  o Peer: N/A
  o Child :N/A
  o Parent: F1,F3

- Firewall F7
  o Peer: N/A
  o Child :N/A
  o Parent: F3
3.6. FIEP Fall Back Mechanism

This section describes how the design of FIEP helps it overcome protocol state transition problems.

1. FIEP State Looping:

FIEP doesn’t have state looping because all circular loops between states have breaking condition to break circular loop in between them. E.g. circular loop between connect and echo state, has a breaking condition, which is when there are no items in not connected stack (i.e. where firewall connects to all the peer, child or parent firewall if any in configuration) the loop will break. And the circular loop also has second breaking condition i.e. timeout mechanism, After sending a timeout in connect state the FIEP will move to echo state and after certain retries FIEP will not try to connect with not connected peer, child or parent if any.

2. FIEP State Deadlock:

State Deadlock happened when two node of protocol are waiting on each other to finish their task FIEP is free from Deadlock state because FIEP state translations don’t only depend on event or message from other FIEP but state translation also depends on the timeout. When every a timeout occur that is an event for the FIEP which break any deadlock situation that the firewall is in.
Chapter 4

FIEP SIMULATION

This chapter will go through FIEP with an example network; this network is shown in Figure 3.

4.1. Simulation Environment

To simulate the protocol, Java threads have been used. Each thread plays the role of a firewall running the FIEP. The communication between the threads is done by using a common message bus. Each firewall (Java thread), when it wants to communicate with other firewalls, puts the information on the message bus.

1. Data Structure of Message Bus:
   - Message ID – This is an incremental value of the message bus.
   - MFrom (Message From) – Indicates the source of the message.
   - MTo (Message To) – Indicates the destination of the message.
   - Payload – This element of the data structure contains the actual message.

4.2. How to Run Simulation?

This section discusses how to run an FIEP simulation. Before running the simulation, an environment needs to be setup. Follow the steps listed below to setup this environment.

- Install NetBeans and MySQL;
- Create 2 tables in MySQL;
  - Table 1: table name, message (ID, MFrom, MTo, PayLoad)
  - Table 2: table name, MPro (ID, MTo, PayLoad)
- Create a Java project in NetBeans and create the classes and code from Appendix.
- Configure the peer, child, parent in the main function and execute the main function.
- Once the Configure is done, execute the class with main function, all the firewall will start in FIEP with idle state as first state.
- Once all the firewall come into echo state, external event such IDS reporting attack, ACL change on a firewall and updating ACL due to an attack can be performed.

<table>
<thead>
<tr>
<th>Table 9: Database Table MyPro Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column name</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>MTo</td>
</tr>
<tr>
<td>Payload</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- **IDS Reporting Attack**: To simulate IDS reporting attack to firewall f1, insert a row in mypro table in the database. This row should follow the table structure ID, MTo, Payload. First element of the row should be an integer this is the primary key of the table. Second element is of data type varchar, here the firewall name to which the message is intended for is entered and the third element is payload which contains the message itself.
In the above figure all the firewall are running FIEP. Static configuration for FIEP is as follows:

- F1
  - peer: F2, F3
  - child: F5, F6, F7

Figure 5: Simulation Network
○ parent: NA (Not available)

• F2
  ○ peer: F1, F4
  ○ child: NA
  ○ parent: NA

• F3
  ○ peer: F1, F4
  ○ child: F7, F8
  ○ parent: NA

• F4
  ○ peer: F3, F2
  ○ child: NA
  ○ parent: NA

• F5
  ○ peer: NA
  ○ child: NA
  ○ parent: F1

• F6
  ○ peer: NA
  ○ child: NA
  ○ parent: F1
4.3. Simulation Output

To explain the FIEP in this example network, I will be using timeline and time will be defined in seconds. If the timeline says “timeline: 100 sec” it implies 100 sec after the experiment has started.

**Timeline: 0-5sec**

- F7
  - peer: NA
  - child: NA
  - parent: F1, F3

- F8
  - peer: NA
  - child: NA
  - parent: F8

- F1 firewall starts FIEP and F1 is in Idle state.
- F2,F3...F8 are waiting to starts FIEP
Scenario One:

- In this scenario, firewall F1 has stated the FIEP and it’s in idle state. Remaining firewalls F2, F3… F8 are waiting to start FIEP.

Time 6ms - 10ms

- F2, F3 … F8 starts the FIEP and they are in idle state.
- F1 moves from Idle state to Connect state and its checking its static configuration to find its peer, child and parent if any.
Figure 7: Simulation Scenario Two

Scenario Two:

- In this scenario, firewall F1 has complete checking FIEP configuration file in Idle state and has moved to Echo state. The remaining firewalls F2, F3...F8 have started the FIEP and are in Idle state.

Timeline 11-15sec

- The events that are below happen in a time range from 12 sec to 15 sec.
- F1, F2, F3… F8 FIEP state will be changed to connect state from idle state.
  - F1
Will send a “Ready to Connect” message to its peers F2 and F3

Will send a “Ready to Connect” message to its children F5,F6,F7

- F2
  - Will send an “Acknowledgement” message to its peer F1.
  - Will send a “Ready to Connect” message to its peer F4.

- F3
  - Will send a “Ready to Connect” message to its peer F4
  - Will send an “Acknowledgement” message to its peer F1.
  - Will send a “Ready to Connect” message to its children F7 and F8

- F4
  - Will send an “Acknowledgement” message to its peers F2 and F3

- F5
  - Will send an “Acknowledgement” message to its parent F1.

- F6
  - Will send an “Acknowledgement” message to its parent F1.

- F7
  - Will send an “Acknowledgement” message to its parents F1 and F3.

- F8
  - Will send an “Acknowledgement” message to its parent F3.
Scenario Three:

- In this scenario, firewall F1 is still in Connect state and waiting to establish connection with peer, child and/or parent. F1 timeout in Connect state is not completed.
- F2, F3... F8 move to Connect state.
- All firewalls establish a connection with peer, child and parent.

Time 16-20sec

- All Firewall move to Echo state.
Scenario Four:

- In this scenario all the firewalls F1, F2, F3...F8 move to Echo State.

Time 20-45sec

- All firewalls are in Echo state.

Scenario Five:

- In this scenario all the firewalls F1, F2, F3...F8 are still waiting in Echo State.
Time 46-50 sec

- IDS report attack on F3.
- F3 sends in attack reported message to F1, F2, F4 and child F7, F8.
- F1, F2, F4, F5, F6, F7, F8 remain in Echo State.

![Figure 11: Simulation Scenario Six](image)

Scenario Six:

- In this scenario, IDS reports an attack on firewall F3. F3 is in Echo state.
- F3 moves to Report Attack state and sends a message to F1, F2, F4, F7 and F8.
- F1, F2, F4, F5, F6, F7 and F8 remain in Echo state.

Time 56-60 Sec

- F1, F2, F4, F7, F8 move to Attack Response state.
- F3 move to Echo state.
• F5, F6 remain in Echo state.

![Diagram](image.png)

**Figure 12: Simulation Scenario Seven**

**Scenario Seven:**

• In this scenario, F3 Move to Echo state from Report Attack state.

• F1, F2, F4, F7 and F8 move to Attack Respond state from Echo state.

**Time 61-65 ms**

• F1, F2, F4, F7, F8 checks there ACL to block the Attack.

• Only F1 need to update ACL to block the attack, remain firewall have ACL in place to block the attack already.

• F1, F2, F4, F7, F8 move back to echo state
Scenario Eight:

- F2, F4, F7 and F8 check their ACLs to block the attack and find that there are already ACLs in place to block the attack. So there is no need to update the ACLs. F2, F4, F7 and F8 move to the Echo state from Attack Respond state.
- F1 checks its ACL to block the attack and finds that there are no current ACLs to block the attack. So F1 updates the ACL and F2, F4, F7 and F8 move to Echo state from the Attack Respond state.
**Time 66-70ms**

- F1 move to Report Rule change state.
- F2, F3, F4...F8 remain in Echo state.
- F1 in Report Rule change state, send the information about ACL update to its child F5, F6, F7.

---

**Scenario Nine:**

- In this scenario, F1 has updated its ACL due to an attack reported by F3.
- F1 moves from Echo state to Report Rule Change state.

---

*Figure 14: Simulation Scenario Nine*
• F1 in Report Rule Change state sends “Rule Changed” message to F5, F6 and F7.
• F2, F3…F8 remain in Echo State.

**Time 71-75ms**

• F1 moves back to echo state.
• F5, F6, F7 move to Rule change Respond State and check their uplink and downlink.
• F5 uplink is blocked due to the ACL update in F1, F5 send warning message to F1

![Diagram](image-url)

*Figure 15: Simulation Scenario Ten*
Scenario Ten:

- In this scenario, F5, F6 and F7 move to Rule Change Respond state, due to message from F1.
- F5, F6 and F7 check for network inconsistency between them and F1.
- F1, F2, F3, F4 and F8 remain in Echo state

Time 76-80ms

- F5, F6, F7 move to Echo state.

![Diagram](image)

Figure 16: Simulation Scenario Eleven

Scenario Eleven:

- In this scenario, F5, F6 and F7 move to Echo state from Rule Change Respond state.
- F1, F2, F3, F4 and F8 remain in Echo state.
5.1. Conclusion

The Firewall Information Exchange Protocol (FIEP) thus yearns to overcome the Network Inconsistency issue by proposing an initial design of a protocol that works as an alert mechanism for Network Administrators so that they can be well informed about any inconsistencies in the ACLs that govern the network. It also brings to light the new technique of firewall communication, a process by which firewalls talk and inform each other of any attacks, thus creating a better awareness of not just the attack but the entire state of the network.

The contributions of this project are:

- A new protocol to incorporate firewall communication mechanism;
- An initial proposal of Firewall grouping in FIEP;
- This protocol introduces the concept of firewall awareness, not just of itself but of the network as a whole;
- Solves the problem of Network Inconsistency;
- Ability to detect Distributed Attacks.
5.2. Future Work

This project is an initial design of the Firewall Information Exchange Protocol (FIEP). To explain the protocol’s state transitions, a simple simulation was developed. A more realistic simulation can be created as part of the future work. This simulation needs to be implemented using socket programming, where each firewall will be associated with a socket which mimics a real network. This simulation should be able to follow real firewall syntax for ACLs with source address, destination address and action (Allow/Block). Additional parameters should be placed to monitor the performance of the protocol.
APPENDIX

Source Code

/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package fiep;

import java.util.concurrent.Semaphore;

/**
* @author Sandeep Pedditi
*/
public class FIEP {

/**
* @param args the command line arguments
*/
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        data de = new data();
        //configuration set
        Semaphore s1 = new Semaphore(1);
        String[] f1peers = {"f2","f3");
        String[] f2peers = {"f1","f4");
        String[] f3peers = {"f1","f4");
        String[] f4peers = {"f2","f3");
        String[] f1child = {"f5","f6","f7");
        String[] f3child = {"f7","f8");
        String[] f5parent = {"f1");
            String[] f5peers = {"f6");
            String[] f6peers = {"f5");
            String[] f5child = {"f9");
        String[] f6parent ="f1");
        String[] f7parent ="f1","f3");
        String[] f8parent ="f3");
            String[] f9parent ="f5");

    }
}
de.connt();

de.deldta();

//newfe f1 = new newfe("f1", f1peers, f1child, s1);
//    newfe f2 = new newfe("f2", f2peers, s1);
//newfe f3 = new newfe("f3", f3peers, f3child, s1);
//newfe f4 = new newfe("f4", f4peers, s1);
//newfe f5 = new newfe("f5", s1, f5parent);
//newfe f6 = new newfe("f6", s1, f6parent);
//newfe f7 = new newfe("f7", s1, f7parent);
//newfe f8 = new newfe("f8", s1, f8parent);

newfe f1 = new newfe(s1, "f1", f1peers, f1child, null);
newfe f2 = new newfe(s1, "f2", f2peers, null, null);
newfe f3 = new newfe(s1, "f3", f3peers, f3child, null);
newfe f4 = new newfe(s1, "f4", f4peers, null, null);
newfe f5 = new newfe(s1, "f5", f5peers, f5child, f5parent);
newfe f6 = new newfe(s1, "f6", f6peers, null, f6parent);
newfe f7 = new newfe(s1, "f7", null, null, f7parent);
newfe f8 = new newfe(s1, "f8", null, null, f8parent);
newfe f9 = new newfe(s1, "f9", null, null, f9parent);

Thread t1 = new Thread(f1);
Thread t2 = new Thread(f2);
Thread t3 = new Thread(f3);
Thread t4 = new Thread(f4);
Thread t5 = new Thread(f5);
Thread t6 = new Thread(f6);
Thread t7 = new Thread(f7);
Thread t8 = new Thread(f8);
Thread t9 = new Thread(f9);
t1.start();
t2.start();
t3.start();
t4.start();
t5.start();
t6.start();
t7.start();
t8.start();
t9.start();

try {
```java
t1.join();
t2.join();
t3.join();
t4.join();
t5.join();
t6.join();
t7.join();
t8.join();
t9.join();
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
dc.closecon();
```

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package fiep;

/**
 * @author Sandeep
 */
import java.util.concurrent.Semaphore;

public class newfe implements Runnable {

data dc = new data();
String hostname;
String ipaddress;
String MessageFrom = null;
String Message = null;
String[] conTo;
String[] ConnectedTO;
String[] childFirewall;
String[] childTO;
```
String[] parent;
String[] parentTo;
int connectIndex = 0;
int parentIndex = 0;
int childIndex = 0;
String priority;
String stat = null;
int Imsg;
Semaphore s1;
ftime fx = new ftime();

newfe(String hostname, Semaphore s1, String[] Parent) {
    this.hostname = hostname;
    this.s1 = s1;
    this.stat = "idel";
    this.conTo = null;
    this.childFirewall = null;
    this.parent = Parent;
    this.parentTo = new String[Parent.length];
    this.childTO = null;
    priority = "medium";
    Imsg = 0;
}

newfe(Semaphore s1, String hostname, String[] peers, String[] child, String[] parent) {
    this.s1 = s1;
    this.hostname = hostname;
    if (peers != null) {
        ConnectedTO = new String[peers.length];
        this.conTo = peers;
    } else {
        this.conTo = null;
    }
    if (child != null) {
        childFirewall = new String[child.length];
        this.childFirewall = child;
        childTO = new String[child.length];
    } else {

his.childFirewall = null;
}
if (parent != null) {
    this.parent = parent;
    this.parentTo = new String[parent.length];
} else {
    this.parent = null;
}
priority = "medium";
Imsg = 0;
stat = "idel";

newfe(String hostname, String[] conTO, Semaphore s1) {
    this.hostname = hostname;
    this.conTo = conTO;
    this.s1 = s1;
    this.stat = "idel";
    ConnectedTO = new String[conTo.length];
    childFirewall = new String[conTo.length];
    childTO = null;
    parent = null;
priority = "medium";
Imsg = 0;
stat = "idel";
}

newfe(String hostname, String[] conTO, String[] child, Semaphore s1) {
    this.hostname = hostname;
    this.conTo = conTO;
    this.s1 = s1;
    this.stat = "idel";
    childFirewall = new String[child.length];
    this.childFirewall = child;
    ConnectedTO = new String[conTo.length];
    childTO = new String[child.length];
    parent = null;
priority = "medium";
Imsg = 0;
stat = "idel";
}
public newfe() {
    // TODO Auto-generated constructor stub
}

@Override
public void run() {
    try {

        while (lmsg > -1) {
            fx.nowtime();

            if (stat == "idel") {
                Thread.sleep(1000);
                s1.acquire();
                System.out.println(this.hostname + " in idel state");
                this.idel();
                s1.release();
                if (lmsg == 1) {
                    this.stat = "connect";
                } else {
                    this.stat = "idel";
                }
                Thread.sleep(1000);
            } else if (stat == "connect") {
                s1.acquire();
                Thread.sleep(500);
                FIEPconnect();
                s1.release();
                Thread.sleep(1000);
            } else if (stat == "echo") {
                s1.acquire();
                Thread.sleep(2000);
                System.out.println(this.hostname + " in " + stat + " State");
                FIEPecho();
                s1.release();
                Thread.sleep(2000);
            } else if (stat == "RespoAttack") {
                s1.acquire();
                Thread.sleep(1000);
            }
        }
    }
}
System.out.println(this.hostname + " in Attack Repound state ");
attackresponnse();
s1.release();
Thread.sleep(1000);

} else if (stat == "RespoRules") {
    s1.acquire();
    Thread.sleep(1000);
    System.out.println(this.hostname + " in Rule changed Repound state ");
    rulechangeresponnse();
s1.release();
    Thread.sleep(1000);
}

}

} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

private void idel() {
    // TODO Auto-generated method stub
    if (lmsg == 0) {
        de.connt();
        if (conTo != null) {
            for (int i = 0; i < conTo.length; i++) {
                if (!conTo[i].equals(null)) {
                    de.inData(this.hostname, this.conTo[i], "NConnect");
                }
            }
        }

        if (childFirewall != null) {
            for (int i = 0; i < childFirewall.length; i++) {
                if (!childFirewall[i].equals(null)) {
                    
                }
            }
        }
    }
    de.inData(this.hostname, this.childFirewall[i], "CConnect");

    }

    }

    if (parent != null) {
    for (int i = 0; i < parent.length; i++) {
    if (!parent[i].equals(null)) {
        de.inData(this.hostname, this.parent[i], "PConnect");
    }
    }

    }

    }

    Imsg = 1;
    de.closecon();

    }

private void FIEPconnect() {
    //int x=conTo.length;

    de.connt();
    String[] peer = de.myread(this.hostname, "Nconnect");
    if(this.hostname=="f2")
    {
        int ne=1;
    }

    if (peer != null) {
    if (peer[3].equals("NConnect")) {
        int te = 0;
        if (conTo != null) {
            for (int i = 0; i < connectIndex; i++) {
                if (peer[1].equals(ConnectedTO[i])) {
                    te = 1;
                }
            }
        }

        if (connectIndex < conTo.length && te == 0) {

ConnectedTO[connectIndex] = peer[1];
connectIndex++;
for (int i = 0; i < conTo.length; i++) {
    if (conTo[i] == peer[1]) {
        conTo[i] = "-1";
    }
}
de.delrow(Long.valueOf(peer[0]), "connect");
System.out.println(this.hostname + " Conneted to peer " + peer[1]);
}

String[] child = de.myread(this.hostname, "Cconnect");
if (child != null) {
    if (child[3].equals("PConnect")) {
        int te = 0;
        for (int i = 0; i < childindex; i++) {
            if (child[1].equals(childTO[i])) {
                te = 1;
            }
        }
    }
    if (childindex < childFirewall.length && te == 0) {
        childTO[childindex] = child[1];
        childindex++;
        for (int i = 0; i < childFirewall.length; i++) {
            if (childFirewall[i].equals(child[1])) {
                childFirewall[i] = "-1";
            }
        }
        de.delrow(Long.valueOf(child[0]), "connect");
        System.out.println(this.hostname + " Conneted to child " + child[1]);
    }
}
String[] parentout = de.myread(this.hostname, "Pconnect");
if (parentout != null) {
    if (parentout[3].equals("CConnect")) {
        int te = 0;
    }
for (int i = 0; i < parentIndex; i++) {
    if (parentout[1].equals(parentTo[i])) {
        te = 1;
    }
}

if (parentIndex <= parent.length && te == 0) {
    if (this.hostname.equals("f7")) {
        int kks = 10;
    }
    parentTo[parentIndex] = parentout[1];
    parentIndex++;
    for (int i = 0; i < parent.length; i++) {
        if (parent[i].equals(parentout[1])) {
            parent[i] = "-1";
        }
    }
    de.delrow(Long.valueOf(parentout[0]), "connect");
    System.out.println(this.hostname + " Conneted to parent " +
    parentout[1]);
}

int cout = 0;
if (conTo != null) {
    if (connectIndex == conTo.length) {
        cout++;
    }
} else {
    cout++;
}
if (childFirewall != null) {
    if (childIndex == childFirewall.length) {
        cout++;
    }
} else {
    cout++;
}
if (parent != null) {
    if (parentIndex == parent.length) {
        cout++;
    }
}
else {
    cout++;
}

if (cout == 3) {
    stat = "echo";
}

de.closecon();

private void FIEPecho() {
    // TODO Auto-generated method stub
    if (this.hostname.equals("f2")) {
        int stoep = 99;
    }
    mychange();
    de.connt();
    String[] out = de.myread(this.hostname, "echo");

    if (out != null) {
        if ("RespoRules".equals(out[3])) {
            this.MessageFrom = out[1];
            this.Message = "RespoRules";
            //System.out.println("Rule change report by "+out[1]+" and Repo by "+out[2]);
            de.delrow(Long.valueOf(out[0]), "RespoAttack");
            this.stat = "RespoRules";
        }
        else if ("RespoAttack".equals(out[3])) {
            this.MessageFrom = out[1];
            this.Message = "RespoAttack";
            //System.out.println("Attack report by "+out[1]+" and Repo by"+out[2]);
            de.delrow(Long.valueOf(out[0]), "RespoAttack");
            this.stat = "RespoAttack";
        }
        else {
            de.delrow(Long.valueOf(out[0]), "RespoAttack");
        }
    }
}
de.closecon();

private void mychange() {
    de.connt();
    String[] out = de.mychanges(this.hostname);
    if (out != null) {
        if (out[2].equals("ReportAttack")) {
            System.out.println(this.hostname + "in Report Attack State");
            if (this.priority.equals("high") || this.priority.equals("medium")) {
                int temp = 0;
                if (ConnectedTO != null) {
                    for (int i = 0; i < ConnectedTO.length; i++) {
                        System.out.println(this.hostname + " and message sent to peer "+ ConnectedTO[i]);
                    }
                    temp = 1;
                }
                if (temp == 0) {
                    System.out.println("Attack Report by " + this.hostname + " but no peer or child");
                    de.delrow(Long.valueOf(out[0]), "ReportAttack");
                }
            } else {
                de.inData(this.hostname, this.ConnectedTO[i], "RespoAttack");
                System.out.println("Report Attack by " + this.hostname + " and message sent to peer "+ ConnectedTO[i]);
            }
        }
    }
}
else if (this.priority.equals("medium")) {
    if (ConnectedTO != null) {
        for (int i = 0; i < ConnectedTO.length; i++) {
            de.inData(this.hostname, this.ConnectedTO[i], "RespoAttack");
            System.out.print(this.hostname + " (in Report Attack State) ");
            System.out.println("Report Attack by " + this.hostname + " and 
message sent to peer " + ConnectedTO[i]);
        }
    }
    if (childTO != null) {
        for (int i = 0; i < childTO.length; i++) {
            de.inData(this.hostname, this.childTO[i], "RespoAttack");
            System.out.print(this.hostname + " (in Report Attack State) ");
            System.out.println("Report Attack by " + this.hostname + " and 
message sent to child " + childTO[i]);
        }
    }
    de.delrow(Long.valueOf(out[0]), "ReportAttack");
}

else if (out[2].equals("ReportRule")) {
    System.out.println(this.hostname + " in Report Rule change State");
    if (this.priority.equals("high") || this.priority.equals("medium")) {
        int temp=0;
        if (childTO != null) {
            temp=1;
            for (int i = 0; i < childTO.length; i++) {
                de.inData(this.hostname, this.childTO[i], "RespoRules");
                System.out.println(this.hostname + " (in Report Rule Changed State 
) Message sent to child " + childTO[i]);
                System.out.println("ReportRule by " + this.hostname);
            }
        }
    }
}
private void attackrespondd() {
    System.out.println(this.hostname + " is in Attack Repoud state");
    de.connt();
    String[] out = de.myatt(this.hostname);

    if(out!=null)
    {
        System.out.println(this.hostname + " Attack is taken care, ACL updated ");
        de.delrow(Long.valueOf(out[0]), "ReportAttack");
    } else
    {
        System.out.println(this.hostname + " Attack is taken care, Rule already in place to block the attack ");
    }
    de.closecon();
    this.stat = "echo";
}

private void rulechangerespondd() {
    de.connt();
    String[] out=de.mywar(this.hostname);
if(out!=null)
{
    System.out.println("Uplink blocked, Warning message to "+out[3]);
    de.delrow(Long.valueOf(out[0]),"ReportAttack");
}
else
{
    System.out.println(this.hostname + "No network inconsistency ");
    this.stat = "echo";
}
    this.stat = "echo";
    de.closecon();
}

/*****************/
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package fiep;

/**
* @author Sandeep Peddit
*/
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import com.mysql.jdbc.PreparedStatement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import com.mysql.jdbc.PreparedStatement;

public class data {

    public int count = 0;
    String MFrom;
    String MTo;
    String payLoad;
    Connection con;
public void updatecount() {
    count++;
}

void connt() {
    try {
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/TEST", "root", "root");
        con.isReadOnly();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

void closecon() {
    try {
        con.close();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

void deldata() {
    PreparedStatement stm;
    try {
        stm = (PreparedStatement) con.prepareStatement("delete from messages");
        stm.executeUpdate();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

String[] mywar(String hostname) {

PreparedStatement stm = null;
    long li;

    try {

        stm = (PreparedStatement) con.prepareStatement("Select * from test.mypro
where hostname='" + hostname + "' and payload = 'Warning' ");
        ResultSet rs = stm.executeQuery();
        String[] output = new String[4];

        if (rs.first()) {

            li = rs.getLong("ID");
            output[0] = String.valueOf(li);
            output[1] = rs.getString(2);
            output[2] = rs.getString(3);
            output[3] = rs.getString(4);
            return output;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;

    String[] myatt(String hostname) {

        PreparedStatement stm = null;
        PreparedStatement stm1 = null;
        PreparedStatement stm2 = null;
        long li;
        try {

            stm = (PreparedStatement) con.prepareStatement("Select * from test.mypro
where hostname='" + hostname + "' and payload='ACL' ");
            ResultSet rs = stm.executeQuery();
            String[] output = new String[3];
            if (rs.first()) {

                } catch (Exception e) {
                    e.printStackTrace();
                }
            return null;
        }
     }
li = rs.getLong("ID");
output[0] = String.valueOf(li);
output[1] = rs.getString(2);
output[2] = rs.getString(3);
stm1 = (PreparedStatement) con.prepareStatement("Select id from test.mypro ");
ResultSet rs1 = stm1.executeQuery();
int x = 0;
if (rs1.last()) {
    x = rs1.getInt("ID") + 1;
}
String temp = "ReportRule";
stm2 = (PreparedStatement) con.prepareStatement("insert into test.mypro (id,hostname,Payload) values(" + x + "," + hostname + "," + temp + ")");
stm2.executeUpdate();
return output;
}

} catch (Exception e) {
e.printStackTrace();
}

return null;

}
String[] myread(String MTo, String op) {
    ResultSet rs = null;
    String[] output = new String[5];
    output[4] = "false";
    try {
        PreparedStatement stm = null;
        if (op == "Nconnect") {
            stm = (PreparedStatement) con.prepareStatement("Select * from messages where MTo='" + MTo + "' and conAwk=1 and payload='NConnect'");
            rs = stm.executeQuery();
        } else if (op == "Cconnect") {
            stm = (PreparedStatement) con.prepareStatement("Select * from messages where MTo='" + MTo + "' and conAwk=1 and payload='PConnect'");
            rs = stm.executeQuery();
        } else if (op == "Pconnect") {
            stm = (PreparedStatement) con.prepareStatement("Select * from messages where MTo='" + MTo + "' and conAwk=1 and payload='CConnect'");
            rs = stm.executeQuery();
        } else if (op == "echo") {
            stm = (PreparedStatement) con.prepareStatement("Select * from messages where MTo='" + MTo + "' and payload in ('RespoAttack','RespoRules')");
            rs = stm.executeQuery();
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    int x = 0;
    return output;
}

output[0] = String.valueOf(ll);
output[1] = rs.getString(2);
output[2] = rs.getString(3);
return output;
} else {
    return null;
}
if (rs.first()) {
  long li = rs.getLong("ID");
  output[0] = String.valueOf(li);
  output[1] = rs.getString(2);
  output[2] = rs.getString(3);
  output[3] = rs.getString(4);
  if (op.equals("Nconnect") || op.equals("Cconnect") || op.equals("Pconnect")) {
    PreparedStatement stm1 = (PreparedStatement) con.prepareStatement("update messages set conAwk = 2 where ID=" + li + " ");
    stm1.executeUpdate();
    return output;
  } else {
    int t = 1;
    return null;
  }
}

} catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
}

return null;

}

int delrow(long id, String op) {
  PreparedStatement stm;
  int out = 0;
  try {
    if (op.equals("Nconnect") || op.equals("Cconnect") || op.equals("Pconnect")) {
      stm = (PreparedStatement) con.prepareStatement("delete from messages where ID=" + id + " and conAwk=2");
      stm.executeUpdate();
    } else if (op.equals("RespoAttack")) {
      stm = (PreparedStatement) con.prepareStatement("delete from messages where ID=" + id + " ");
      stm.executeUpdate();
    } else if (op.equals("ReportAttack")) {
      stm = (PreparedStatement) con.prepareStatement("delete from mypro
where ID=" + id + ");
    stm.executeUpdate();
}

out = 1;
} catch (SQLException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
}
return out;

int inData(String MFrom, String MTo, String payLoad) {
  try {

    //String Query = "INSERT INTO TEST (ID, MFrom, MTo, Payload) VALUES
    //(?,?,?,?)";
    PreparedStatement stm;
    long x = 0;
    PreparedStatement stm1 = null;
    stm = (PreparedStatement) con.prepareStatement("select ID from
    messages");
    ResultSet rs = stm.executeQuery();
    if (rs.last()) {
      rs.last();
      x = rs.getInt("ID") + 1;
    } else {
      x = 1;
    }
    rs.close();

    if (payLoad.equals("NConnect") || payLoad.equals("CConnect") ||
    payLoad.equals("PConnect")) {
      int connectAWk = 1;
      stm1 = (PreparedStatement) con.prepareStatement("insert into
      messages(id,MFrom,MTo,Payload,conAwk) values(" + x + "," + MFrom + "," + MTo + "," + payLoad + "," + connectAWk + ")");
      stm1.executeUpdate();
    } else if ("RespoAttack".equals(payLoad) || "RespoRules".equals(payLoad)) {
      stm1 = (PreparedStatement) con.prepareStatement("insert into
      messages(id,MFrom,MTo,Payload) values(" + x + "," + MFrom + "," + MTo + "," + payLoad + ")");
      stm1.executeUpdate();
    } else {
      // do something else
    }
  }
}
package fiep;

public class ftime {

    static int time=0;
    static int outt=5;

    public static void nowtime() {
        time++;
        if(time%5==0) {
            System.out.println();
            System.out.println("time "+outt+"ms"+"----------------------------------------
--");
            System.out.println();
            outt=outt+5;
        }
    }

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return 0;
}

/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/

@Author Sandeep Pedditi

public class ftime {
    static int time=0;
    static int outt=5;

    public static void nowtime() {
        time++;
        if(time%5==0) {
            System.out.println();
            System.out.println("time "+outt+"ms"+"----------------------------------------
--");
            System.out.println();
            outt=outt+5;
        }
    }

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return 0;
}

}
BIBLIOGRAPHY

1. “Structured Firewall Design”, Mohamed G Gouda and Alex X Liu, April 2006.
   Available online: www.cs.utexas.edu/~gouda/papers/journal/Firewall.pdf


4. “General Firewall White Paper” by Tom Sheldon


   Available online: www.springerlink.com/index/K35502244377N375.pdf

   Available online: http://cyberdna.uncc.edu/pubs/commag.pdf