CS 5434

NETWORK FIREWALL

NAME - PIYUSH MAHESHWARI
NETID – pm489

Table of Contents

1 Introduction

This report describes the architecture of a network firewall implemented as part of CS 5434 project. To describe it briefly firewall is a piece of software that controls the incoming and outgoing network traffic by analysing the data packets and determining whether they should be allowed or not, based on applied rule set. It will basically act as a barrier between a trusted and an untrusted network.

1.1 Brief Overview

The firewall designed here is Stateful. This basically means that this firewall performs stateful packet inspection that keeps track of state of network connection travelling across it. Only packets matching a known active connection would be allowed to pass through it, others will be rejected.

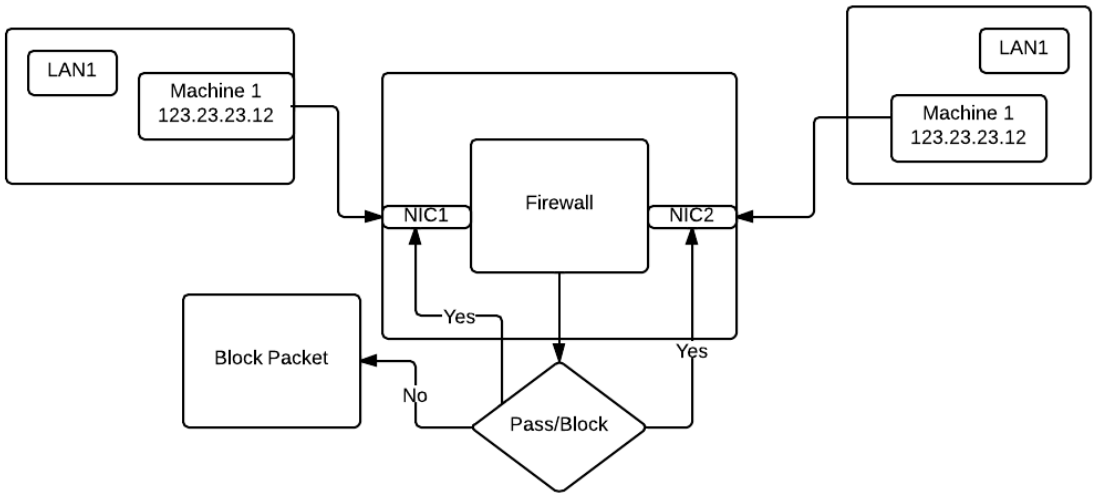
It defines a rule base language in which user can specify firewall rules which would be applied to packets crossing the network. These rules can be read off a file on the disk. It currently support up to 10 network interfaces which can be predefined and read off a file at start-up.

1.2 Structure of this Document

The remainder of this document is arranged as follows: section two describes the software architecture and the various design decisions made. It describes in detail the working of the firewall. Section three describes the tests that have been/would be performed to test the correctness and scalability of this firewall. Section four briefly describes how this firewall can be expanded to add more features and further improvements that can be made to it.

2 Architecture

This sector details the architecture of the firewall. The diagram below describes the interaction of the main external components of the system at a high level. The independent modules of the system are mentioned thereafter.



2.1 Network Interfaces

When the firewall starts up, it reads a list of network interfaces that the firewall would listen to. This file is called “interfaces.txt” and is present in the source folder. The file contents are of the following form

DEVICE NAME1 MACADDRESS1

DEVICE NAME2 MACADDRESS2

These devices basically define the input and output points of a network. All interfaces are treated equally and each follow the set of firewall rules defined for them.The main thread reads every device in this list and starts a new thread for each network interface. It creates an object called struct network_interface defined in “network_interface_card.h” which contains fields like the subnet mask, network mask and the handle for the network interface. This corresponding structure object is then passed to its threads and it is started. Also each network interface object is added to a list so that they can be referenced later. Currently the list is defined to be of size 10 but this can be increased later easily.

2.2 Firewall rules

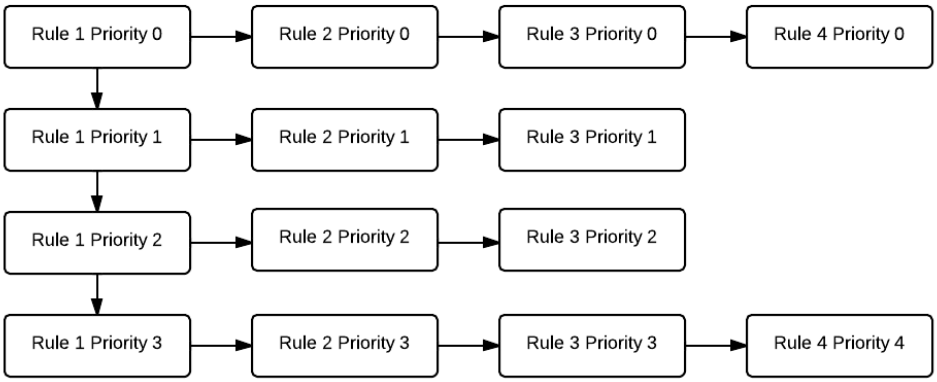
The rules are fetched from a text file called “rules.txt”. The rules can take the following form:

Action SRCIP=x/y SRCPORT=p-q DSTIP=a/b DSTPORT=m-n PRIORITY=p

Some example rules are:

```
Pass SRCIP=60.50.40.30/24 SRCPORT=23 DSTIP=55.255.255.255 DSTPORT=34 PRIORITY=9
Pass SRCIP=60.50.40.31/24 SRCPORT=23 DSTIP=55.255.255.254 DSTPORT=34 PRIORITY=8
Pass SRCIP=60.50.40.32/24 SRCPORT=23 DSTIP=55.255.255.253 DSTPORT=34 PRIORITY=8
Pass SRCIP=60.50.40.33/24 SRCPORT=23 DSTIP=55.255.255.252 DSTPORT=34 PRIORITY=0
```

Possible values of ‘Action’ is Pass or Block. Priority value 0 signifies the highest priority and priority value 9 signifies lowest priority. The rules are parsed and maintained in a maze of link list like structure where each link list contains rules of the same priority. Rules are applied to in the order of highest to lowest priorities.



The rules are expressive enough to give enough flexibility to the user to block or pass many types of packets from various sources.

2.3 Packet Processing

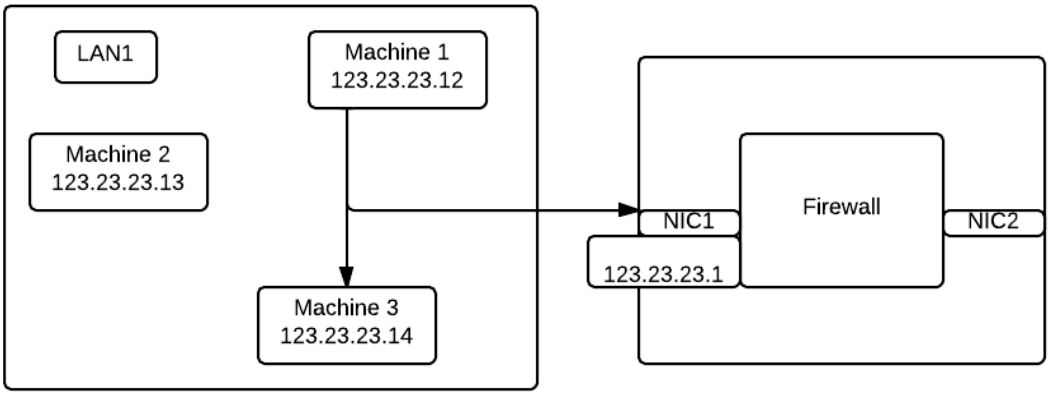
Each thread listening to a network interface reads packet in a loop. PCAP library is used for reading packets off wire. Then each packet is passed through a packet parser which disassembles the Ethernet and IP header. Different packets are handles differently -

a) ARP Packet – In this case we will simply update our local ARP cache for that local network, if the request was sent by us, otherwise ignore the packet.

b) TCP|UDP|ICMP – Here we parse the IP packet. There can be multiple cases depending upon the content of the packet.

- The packet has source mac address same as the network interface that received it. This can happen when some other interface has injected this packet to this interface and we simply read this packet over the wire. In this case we will simple ignore the packet.

- The packet's destination IP address belongs to the same local network in which the given network interface is listening to. This means that the packet is some other machine in the same local network. We can simply ignore this packet too as this is not going to cross the network. The figure below shows this situation –



- The packet destination IP belongs to some other network. There can be a couple of cases in this –

    - There is a connection already open between these two hosts. Since we maintain the state of every connection, we can simple query the state information to find if this packet belongs to an already open connection. If the packet is part of an already open connection, we will allow it to pass.

    - If this packet does not belong to any open connection, we will then traverse the rule chain to find if any rule applies to this packet. If no rule applies we simply drop this packet as the default action is block. If some rule applies and the action associated with that rule is PASS, we allow the packet through, otherwise we drop the packet.

If we decide to packet through, we will have to find which network interface is open to that local network. This can be done by iterating over the list of network interfaces that we have had maintained initially. Since we know the subnet mask and the network address of each interface, this information is easy to find. Also we need the mac address of the destination IP address. We query the ARP cache associated with the corresponding network interface for this.

    - If the entry is found in the cache and it hasn't exceeded the timeout associated with the cache entry, we will use the mac address

    - If the entry is missing or the entry has timed out, we spawn another thread which would make the APR request and update the cache with the value. The previous thread waits for this thread to complete, and then resume its

execution. pthread library is used for spawning child threads and waiting on them.

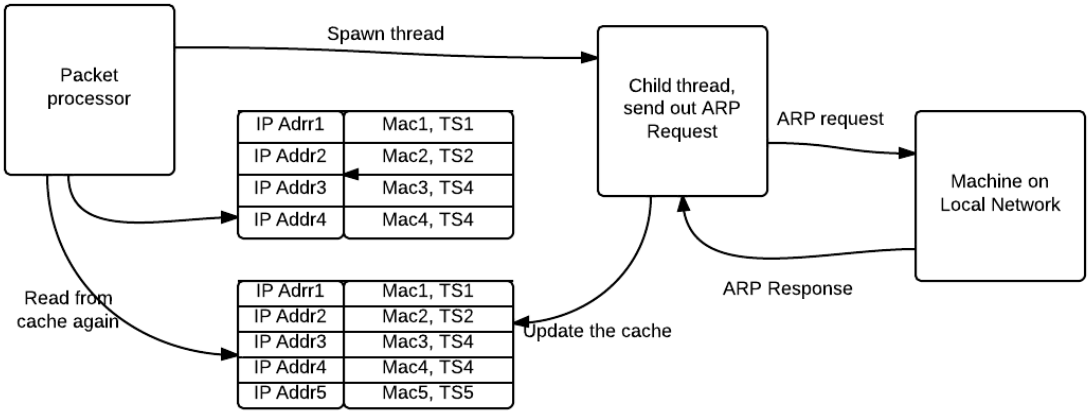
We can then take the network mac address of that interface, overwrite the packet and then simple inject the packet in that interface.

2.4 ARP Cache

ARP cache is basically a key value pair table which has a mac address associated with an IP address entry. There is a different cache per network interface since each network interface is facing the different local network. The following is a format of an ARP table entry.

IP Address	Mac Address + Time Stamp

There would be a **mutex lock** associated with each APR table since different threads might be simultaneously reading and writing on the same cache. Each threads before reading or writing into the cache would first acquire the lock and then release it. This would prevent the cache from containing inconsistent entries.

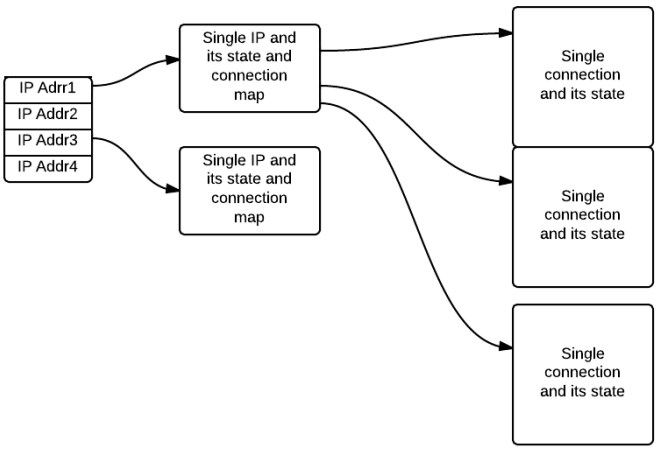


2.5 Network State Information

The state of the connections going across the network would be stored in a Hash Table. We would update the hash table whenever we find a packet which matches a rule PASS. We would also use this hash table to look up whether there is an active connection between two hosts sitting on different networks. Since we need this operation to be constant time, we are using a hash table and not a linear list. We would maintain a hierarchy of hash tables to for space efficiency. The first level hash table would hash on the source IP address. The value of the first level hash table would be a structure which would again contain a second level hash table along with statistics on the host such as the number of connections formed etc. The

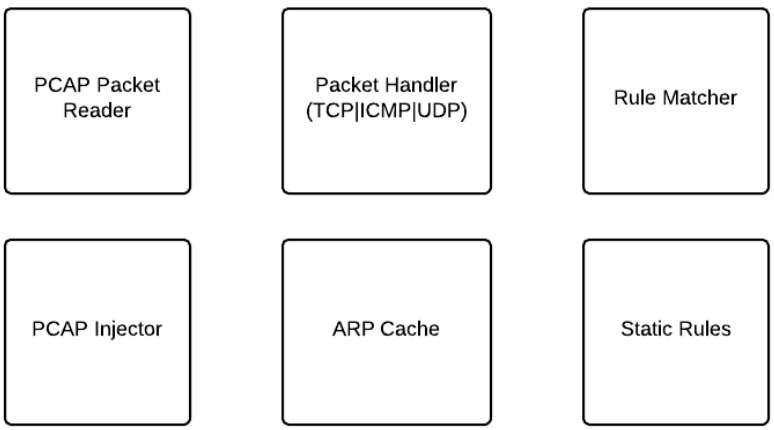
second level hash table would be keyed on a combination of source port, destination port and destination IP address. The value of the second level hash table which represents a valid connection. It would also contain other information like the state of the connection.

When a TCP connection finishes, we would remove the corresponding entry from the second level hash table. Once we know that a host does not have any active connections, we can remove its entry from the first level hash table as well. This would make sure that the hash tables do not blow up in size over a period of time.



2.6 Modules

The following diagram explains the complete code flow with different modules.



3 Testing

There are basically two setups through which testing would be done - Virtual Machine and through pcap files. We can test all possible cases through sending packets through two virtual machines connected to different local area networks and we can stress test using big pcap files

3.1 Virtual Machine Setup

We will set up 2 virtual machines through virtualization software like virtual box. Let us call these machines VM1 and VM2. Let the machine running the firewall be F. VM1 will connected to a local area network and F will be connected to the same LAN network through a virtual network interface. Same procedure will be followed for VM2. Since both machines are connected on different networks, they would have different network address and different subnet mask. F will be set as the default gateway for both VM1 and VM2. Since the VM's would not be connected to any other network, the only way to pass a packet from VM1 to VM2 and vice versa would be through F.

With this set up we can run the following test case to check the correctness of the firewall –

1) Send an ICMP packet from VM1 to VM2 with a firewall rule set to PASS.
2) Send an ICMP packet from VM1 to VM2 with a firewall rule set to BLOCK.
3) Send a TCP packet from VM1 to VM2 to set up a connection with firewall rule set to PASS.
4) Send a TCP packet from VM1 to VM2 to set up a connection with firewall rule set to BLOCK.
5) Send a TCP packet from VM1 to VM2 with an already setup connection.
6) Send a TCP packet from VM1 to VM2 to tear down an existing connection
7) Send a TCP connection VM1 to VM2 with the corresponding ARP entry missing

These are just some of the test cases running. We can similarly run all these tests for UDP as well. We can also write some more tests to check the more corner case that can be encountered in a real life system

3.2 Testing through PCAP File

The firewall would be set to consume from a pcap file rather than a network interface. Since the size of the pcap file would be large, this would allow us to stress test and test the scalability of the data structures used internally.

4 Conclusions and Further Work

This document describes the architecture of a stateful firewall. There are many improvements that can be done in the current architecture to make this firewall work faster. The following points list some ways to achieve this.

1) Make the firewall completely multithreaded. We can use thread pooling and assign every packet to a separate thread of its own. This would need a synchronized thread pool. This would make the packet processing even faster and more concurrent. If time permits I would try to modify the current architecture to incorporate this idea.

2) We can also make the discovery of mac address using ARP requests more concurrent. We can for example scan the local network connected to a network interface and send ARP requests to find their mac addresses. We can do this as a background jobs to make sure that this does not slow down the packet processing time.

3) Also we can have a request queue for the waiting requests to unblock the serving thread. Currently the current thread waits until the child threads returns from its ARP call. We can for example make this packet sit in a wait queue and make the child thread process this packet when the ARP response returns.