Cornell University

Defending Computer Networks

Fall 2013

Firewall Final Project

Author:
Stacey Wrazien

NetID:
saw298

December 10, 2013

Contents

1 Introduction

For my final project in Defending Computer Networks, I have choosen to implement a stateful firewall program in C. In this document, I will the discuss how to setup and run the firewall, along with a complete description of the rules accepted by the firewall. I will also discuss the algorithms and data structures used in writing this program and the test plan to ensure it is working as desired.

2 Commands

2.0.1 Compiling the Firewall

The Firewall implementation can be compiled by typing 'make', which will run the Makefile that is included with the C Firewall files. Once the files have been compiled and you have created your rules file as you desire, you can start the firewall.

2.0.2 Starting the Firewall

After comiling the program and writing any necessary rules, the Firewall could be started by typing './firewall interface interface [interface ..]' where interface would be the name of an interface that the firewall will monitor. The firewall can run with multiple interfaces but must have at least two distinct interface names in order to run correctly.

3 The Rules Language

Rules for the firewall need to be specified before starting the firewall in order for the rules to be in effect. The rules should be specified in a file named 'rules.conf'. If the firewall is started without a rules.conf file in the same directory as the compiled program then all traffic between the specified interfaces will be passed by default.

If the rules.conf file does exist when the firewall is started, then the firewall will apply all of the rules to the traffic that is being passed by the specified interfaces. The rules in the rules.conf file should be specified with the most general rules first, followed by the more specific rules, as the firewall will read in the rules starting

with the first line and overrite any conflicitng rules it encounters with the most recent rule it evaluates.

The user cannot add or change existing rules while the firewall is running. Instead the user must stop the running firewall, change or add any rules for the firewall, and then restart the firewall program in order for the new rules to be active.

The firewall implementation supports three distinct rules. The pass rule, which allows the user to specify specific IPs to pass traffic through the Firewall, the block rule, which allows users to specify IPs to block traffic from passing through the Firewall, and the reject rule, which allows the user to specify IP's to reject traffic from passing through the Firewall.

3.0.3 Pass

The pass rule allows the user to specify what will pass through the firewall. The format of the pass rule is as follows.

***source_inteface destination_interface source_ip source_port dest_ip dest_port* pass**

where

source_inteface is the name of the source interface where the packet was sent from

destination_interface is the name of the destination interface of the packet

source_ip is the source IP address of the packet

dest_ip is the destimation IP address of the packet

source_port is the source port of the packet

dest_port is the destination port of the packet

Or

any any any any any any pass

which would indicate that all traffic should be allowed through the firewall.

Any can also be specified for all of the above fields individually.

3.0.4 Block

The block rule allows the users to specify what packets the firewall will drop without sending any messages back to the sender. The format of the block rule is as follows.

***source_inteface destination_interface source_ip source_port dest_ip dest_port* block**

where
source_inteface is the name of the source interface where the packet was sent from
destination_interface is the name of the destination interface of the packet
source_ip is the source IP address of the packet
dest_ip is the destimation IP address of the packet
source_port is the source port of the packet
dest_port is the destination port of the packet

Or

any any any any any any block

which would indicate that all traffic should not be allowed through the firewall and responses should be sent to the source address. Any can also be specified for all of the above fields individually.

3.0.5 Reject

The reject rule allows the user to specify which packets will not be allowed through the firewall and an ICMP reject message will be sent to the sender. The format of the reject rule is as follows.

***source_inteface destination_interface source_ip source_port dest_ip dest_port* reject**

where

source_inteface is the name of the source interface where the packet was sent from

destination_interface is the name of the destination interface of the packet

source_ip is the source IP address of the packet

dest_ip is the destimation IP address of the packet

source_port is the source port of the packet

dest_port is the destination port of the packet

Or

any any any any any any reject

which would indicate that all traffic not should be allowed through the firewall and ICMP reject messages should be sent to the sender. Any can also be specified for all of the above fields individually.

4 Implementation

4.0.6 ARP Package Resolution

In order to pass packets between interfaces on different subnets, we need to handle and maintain an updated ARP table. The ARP table in this implementation is represented using a hash table. The keys of the hash table are the destination IPs and the values of the hash table are structs containing the destination MAC address, and the time that the address was last seen. If the elapsed time from the time the address was last seen and the current time is greater than 60 seconds, an arp entry in the table is considered stale and removed from the table.

When the firewall encounters a packet that does not have a valid entry for the destination IP in the ARP table, an appropriate ARP packet is sent on the

corresponding interfaces in order to get the correct MAC address for the IP. The packet is then placed in a hash table which has the destination IP as the key and a linked list with structs that contain the packet data as the values. When the firewall sees ARP responses, it adds the MAC addresses to the ARP hash table and then the waiting hash table is checked to see if there are any packets waiting to be sent. If it finds waiting packets, the packets are then sent.

4.0.7 State Table

Because the firewall is stateful, we need to keep track of the state of the connections that are allowed through the firewall. Once a TCP connection has been checked against all of the rules, the connection information, which would include the source and destination IPs and ports, is added to a linked list with the current time. A character string is also created from the connection information and used as a hash table key to store a struct containing the state of the connection and a pointer to the corresponding linked list node. We will refer to this structure as our state hash table going forward. Periodically the state liked list is checked to see if the delta of the time in the node and the current time is beyond a certain threshold, if it is, then the node is removed from the linked list and the connection is marked inactive. We then continue to check the rest of the linked list until we encounter a node that is not beyond the specified threshold. We are able to stop checking at this point because we know all nodes added after this node, were added at a later time and will therefore also not meet our criteria to be expunged.

When our firewall first recieves new packets, it first generates the character string from the connection information. It then uses this string to check our state hash table for a corresponding value. If a value is found, we know that we have an active connection. We then use the pointer in the state hash table to get the node in the linked list in order to update the time in the node. The node is then moved to the end of the linked list to preserve the ordering by time of our linked list.

4.0.8 Rules

The rules for our firewall are read in from the ‘rules.conf‘ file. As rules are read in, they are added to the beggining of a linked list, which means that the rules at the

bottom of the file are applied before the rules at the top of the file. The default rules for the firewall should therefore be placed at the very top of the 'rules.conf' file.

After we check the state table, if we found that a current connection is not open for the corresponding IPs and ports, we then check the current packet against our rule. We start searching at the beginning of our linked list. If we find a rule that matches, then we stop searching the linked list and apply the appropriate action. If the action specified is pass, we forward the packet onto its specified destination. If the action specified is block, we do nothing with the packet. If the action specified is reject, we create a reject ICMP message for UDP and ICMP packets and a TC RST packet for TCP packets. We then forward that packet onto the source and do nothing further with the packet.

5 Test Plan

The firewall will be tested on an Ubuntu version 12.04 virtual machine running CORE [1]. CORE allows us to create several virtual machines on different networks.

In order to test that our ARP table is working correctly in resolving the appropriate MAC addressess for the packets we are sending, we will create three virtual machines, called M1, M2, and M3, with two networks, N1 and N2 on different subnets using CORE. M1 and M2 will be connected to the network N1, and M2 and M3 will be connected to the network N2. We will then start our firewall on machine M2 and attempt to pass packets from M1 to M3 and M3 to M1. Because M1 and M3 are on different subnets, if our firewall is able to resolve and route the packets to the correct machine, then we can conclude our ARP table and ARP packet creation is working correctly. We use tcpdump to monitor the traffic on both M1 and M3, to assure all packets are passed as expected. For this test, we are not concerned with checking the correctness of our rule evaluation yet so we omit setting the rules.conf file for this test.

For our next stage in testing, we will add another virtual machine to our N2 network called M4. We will then test our rules by creating a 'rules.conf' file containing a rule that says reject all packets by default followed by a rule that allows M1 to send packets to M4. We will then use tcpdump on M1, M3 and M4

to monitor the traffic on each. From M1, we will then attempt to send packets to both M3 and M4. M3 should recieve properly constructed ICMP reject messages and M4 should recieve the packets.

We can then alter the rules in 'rules.conf' so that different orderings of the rules should be tested. We should also test adding several rules to the to our file so that we know the firewall can scale well.

Our final test should include checking that our firewall can scale appropriately when given a large number of packets at any given time. Because this is difficult to test using CORE or any virtual machine setup, we will use pcap files to emulate the situation. We will start the firewall will two filenames instead of two interfaces. We will also create an extensive 'rule.conf' file that will specifically allow only certain packets through. Our firewall will then read packets from the first file and the actions of the firewall in regards to the packets will be written to the second file. Because we are using a large pcap file, we will know our data structures aren't working properly if our porgram crashes before completely transcribing all packets to the second file with the appropriate actions.

6 Test Results

6.1 Pcap Files

The firewall was tested using a pcap file generated from live network traffic. The pcap file was 3.5GB. This file needs to be sufficiently large so that we can be sure we do not have memory leaks or other similar problems in our data structures. The first test ran with this file was to run the firewall without any rules in order to verify the state table is functioning properly. By not adding any rules we will have the maximum number of connections open in our state table, which should sufficiently test the memory allocation and deallocation is working as it should be. When ran with the pcap file, there were no reported memory leaks or craches. We can also deduce that the packets are being forwarded appropriately by looking at the output of our firewall.

The second test ran with pcap files was to verify that the rules and appropriate reject packets were being generated appropriately. I decided to use a mcuh smaller pcap file so that it would be easier to verify using wireshark. For this test I used

a 38K pcap file that was also collected from live netowrk traffic. I then created the rules.conf file to have at least one of each of the pass, reject and block rules. The test was then repeated with all the permutation of the rules and verified for correctness with the output pcap file.

6.2 Live Network Testing

Using Core, I added 6 nodes in my network with 1 behind the firewall. I then added rules that specify which of the nodes can talk to the node behind the firewall. I also disallowed traffic to one of the nodes from my internal node. I then used ping and hping to generate TCP, UDP, and ICMP messages between my nodes. Using tcpdump on each of the nodes, I can verify that the correct messages are being passed and the correct reject messages are generated.

All my permutations of networks setup in core worked as I expected them to.

References

[1] Common Open Research Emulator. http://www.nrl.navy.mil/itd/ncs/products/core.