Linux Firewall Performance Analysis

Noe Nevarez and Huy Duc Vo, Graduate Students, University of Houston

***Abstract* – The goal of this experiment is to show how a linux based firewall can perform as well as most of the more expensive commercial appliances that are out there. This type of firewall may be beneficial for small business or businesses at home that don't have enough financial resources for the more expensive firewall appliances. We will be using an application called iptables which runs natively in most if not all linux flavors. This linux firewall will serve as a gateway for internal users as they access external networks outside of the firewall and the internet. We will be using a program called httping and Wireshark to measure the following metrics: Throughput, Protocol Latency, and Concurrent connections (per-second).**

Index Terms – egress, ingress, Stateful

I. INTRODUCTION

A firewall is basically a barrier to protect services that are often mission critical to an organization. Network Engineers often implement these devices at specific locations within a network to protect against many forms of attacks. Every type of firewall has a set of configurable rules that define a global security policy. This policy can be binded to specific interfaces of the device while the firewall runs a Stateful Inspection of the traffic entering or leaving an interface. Stateful Inspection means that the firewall can maintain a record of all the connections entering/leaving the firewall, by monitoring each packet at layer 4 of the OSI model. This helps enforce security policies by allowing transport layer filtering to either allow or deny a packet from entering your protected network. In order to determine what type of firewall is base for a given organization, you must know the current and future data usage for the network you looking to implement this with. This is usually the job of the Network Architect to define the requirements before a vendor is chosen. Once the firewall is implemented, the firewall must be thoroughly tested using a predetermined set of metrics. Some of the most important metrics are:

- **Throughput:** The rate at which device sends or receives data.
- **Protocol Latency:** This is the time between when the request was sent to the reply was received.
- **Concurrent connections (per-second):** The rate at which new TCP connections are established per second.

These metrics are frequency used to test the performance of many different types of architectures and configurations.

II. ANALYSIS OF METRICS

I will now explain some common metrics that are used to test the performance of a firewall. One of the first metrics we will be testing is the Throughput performance. **Throughput** is usually measured in bits per second (bps). In this experiment we will be converting (bps) to (Mbps). We will run our tests by sending HTTP traffic from a client located on the external network (outside firewall) to a server located in the private network (inside firewall). All the HTTP traffic will pass through the Linux Firewall to reach the HTTP server. Throughput will be calculated by the following formula:

- *Throughput (Max BW) = RWIN/ RTT*
  - *RWIN*= TCP Receive Window
  - *RTT*= Round-trip-time for the path

The default window receive size is 65,535 bytes which is 524,280 bits. This is the value we will use for RWIN in our throughput calculation. For the purpose of this test, I will be using a linux application called httping to measure the throughput of our webserver. The full command I will be using will be:

- $httping –Gbg http://www.lynx-ste.lynx.com/

In our next test we will be testing the **Protocol Latency** of our Web Server. This is a very important parameter since it is noticeable to the user trying to access content on the webserver. We will run this test by sending HTTP Traffic from a client to the server while measuring the following items:

1) **Connect Time**: This is the time from when the client sends the initial TCP segment with

the SYN flag set and the type it takes for the server to respond with a TCP packet with the SYN-ACK flag set.

2) **Time-to-first-byte (TTFB):** This is the time that elapses before the client receives the first byte of the HTTP response.
3) **Time-to-last-byte (TTLB):** This is the elapse time it takes for the client to receive the last byte.

Our last test will be measuring the rate at which new connections are established in seconds. This metric will show how well our linux firewall can handle connections based on the scale of the network and on the number of nodes that access content on the server. The number of connections that can be established on the webserver is configurable. We will start this test by sending HTTP traffic from our client to the HTTP server. I will be using a HTTP Traffic generator (Developed by NSASOFT) to send 200 connection requests to the server and measure time.

III. IPTABLES COMPONENTS

A firewall policy is basically and ordered set of rules that gets loaded into memory for the kernel to take action upon. Each iptables rule is applied to a chain inside a table. TABLE 1 shows the different types of tables:

TABLE 1

FILTER	Filtering rules are applied here
NAT	Nat rules are applied here
MANGLE	Rules that alter the packet are applied here
RAW	Rules that should run independently of iptables connection-tracking subsystem are applied to this table.

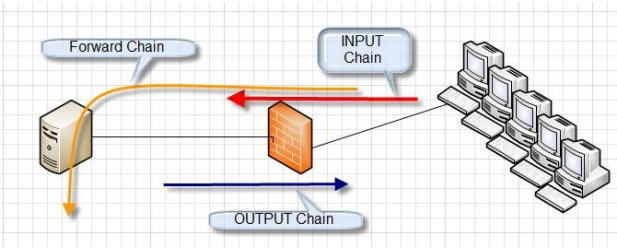
There are many different types of chains but I will only discuss the most widely used ones. You must thoroughly understand these chains before you start implementing the rules for your security policy. A description of each chain is listed in TABLE 2.

TABLE 2

INPUT	Packets destined to the firewall's ingress interface.
OUTPUT	Packets destined out of the firewall's egress interface.
FORWARD	Packets destined to another NIC on the same host.

A visual representation of the chains from a traffic perspective is listed in FIGURE 1.

FIGURE 1



IV. DEMONSTRATION REQUIREMENTS

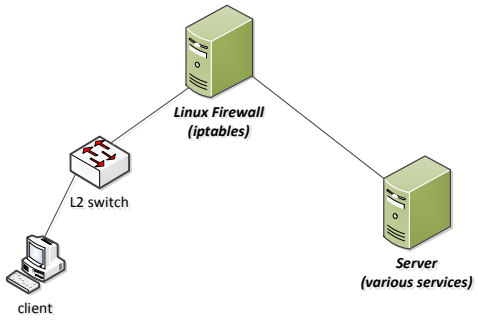
We need to complete some preliminary steps before we can test out iptables ability to prevent an ICMP flood attack. TABLE 3 lists the hardware/software requirements we are using for this scenario.

TABLE 3

CPU Arch.	AMD Athlon (tm) Dual Core Processor 4450e
MEMORY	1879.52 MB
NIC 2	100baseT/Full
Layer 2 Switch	Procurve 408 switch
Layer 2 Switch	Cisco WS-C2960-24-S
O/S	Ubuntu 10.10
Iptables Version	v1.4.4
Web Server	apache2 - 2.2.16-1ubuntu3.1

All of these items are required components before we start our testing scenario. You can see the topology we will be using for testing in FIGURE 2.

FIGURE 2



The client will be connected to a simple Layer 2 switch. This area will represent a network outside the firewall. The Linux firewall will be configured with two network interfaces (100Mbs). The Linux firewall will be configured to forward packets destined to the webserver in eth0 and out eth1 which is the interface that connects to the webserver.

V. PERFORMANCE ANALYSIS

TABLE 1 (T)

Application: httping **Throughput Avg. (Mbps)** **Webpage:** http://lynx-site.lynx.com **50 HTTP GET Requests**
T= 257.0625 Mbps

TABLE 2 (PROTCOL LATENCY)

Application: Wireshark **50 HTTP Requests**	
Connection Time (Avg. sec)	0.0361
Time-to-first-byte (Avg. sec)	0.0145
Time-to-last-byte (Avg. sec)	0.0226

TABLE 3 (CONNECTION RATE)

Application: Wireshark **100 -200 Connection Requests**	
Connection Rate:	4,066.00

You can see in FIGURES 3 and 4 that the CPU utilization did not dramatically rise during all our performance tests. You can also view the amount of memory being utilized by each application using the command listed in FIGURE 5. This command shows you that the Ubuntu systems resources are not over allocated.

FIGURE 3 (CPU 1)

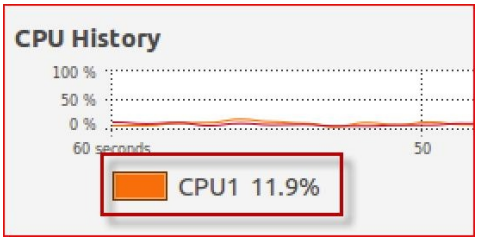


FIGURE 4 (CPU 2)

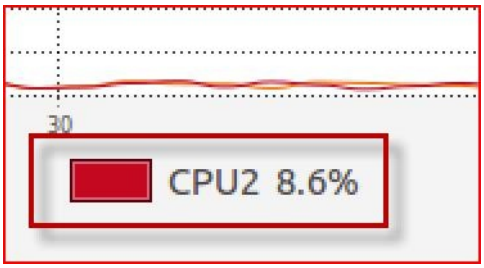


FIGURE 5 (% MEM)

#ps aux | awk '{print $4"\t"$11}' | sort | uniq -c | awk '{print $2" "$1" "$3}' | sort -nr | head
6.2 1 /opt/google/chrome/chrome
4.1 1 /usr/bin/X
3.1 1 /usr/bin/python2.6
3.1 1 /opt/google/chrome/chrome
2.2 1 mono
2.0 1 /usr/lib/nspluginwrapper/i386/linux/npviewer.bin
2.0 1 nautilus
1.8 1 mono
1.0 1 /usr/bin/python
1.0 1 /opt/google/chrome/chrome

- **Note:** This command lists the current % of memory consumption per application running. The *"awk/sort/uniq/"* commands are just to format the output to make it clearly readable. You can see that we do not have any abnormal rise in memory usage even when the IMCP DoS is occurring. The highest memory allocation is for the Google Chrome web browser which is using 6.2%

VI. CONCLUSION

You can see from measured data that the Linux Firewall performs very well in all our tests. Some of the limitations of Linux firewalls that use iptables are that they don't provide for redundancy like the appliances do. This is usually not a huge issue since the low cost is perfect for small businesses. Large enterprises usually use a combination of High-End firewalls and servers that use iptables to provide layered security for servers. I certainly believe this architecture will fit the needs for most small businesses.

REFERENCES

[1] Michael Rask, "Linux Firewalls", 1st ed, Library of Congress Cataloging-in-Publication Data, 2007