**Ubuntu Firewall**

What is a Firewall?

A firewall is a security tool on computer systems that can control all incoming and outgoing traffic. The firewall is the first line of defense when it comes to cyber security. It is important to only allow necessary and secure ports when it comes to inbound traffic. Some network engineers utilize a DMZ (demilitarized zone) as another security measure. This redirects inbound traffic towards a less critical part of the corporate network.

**Web Server**

Includes all web/internet traffic between the server and its visitors. This helps control all of the incoming traffic to the server on port 80 and redirect it to port 8080.

1. **Command – “sudo ufw allow 80/tcp” & “sudo ufw allow 8080/tcp”**

A black screen with white text

Description automatically generated

* These two commands change the firewall rules to allow traffic from port 80 to port 8080. Port 80 is the default port for web (http) traffic and port 8080 is an alternate port for web traffic.

1. **Command – “sudo ufw enable” & “sudo ufw status” & “sudo ufw reload”.**

A screenshot of a computer

Description automatically generated

Enable the firewall and show the status.

Run the first two commands to confirm the firewall is enabled, and check if the new rules for the web server traffic are in place before continuing. Once the settings are confirmed, reload the firewall to apply the new changes. Reload the firewall.

1. **Command – “sudo iptables -t nat -A PREROUTING -p tcp –dport 80 -j REDIRECT –to-port 8080” & “sudo apt install iptables-persistent”.**



A screenshot of a computer

Description automatically generated

* The first command is used to tell the server to redirect all traffic from port 80 to port 8080. To save this rule, install the package iptables-persistent, and select ‘Yes’ on the two prompts. This will ensure the rules we just created will survive reboots and updates to the server.

1. **Command – “sudo iptables -t nat -L -n”.**

A screenshot of a computer

Description automatically generated

* This is a useful command to confirm port forwarding rules, like the one we just created. Option **-t nat** specifies the NAT Table, which is the resource for all Network Address Translation (NAT) rules. **-L** option lists all rules in the firewall, and **-n** is used to disable DNS because we do not need to see hostnames in this scenario.

A screenshot of a computer

Description automatically generated

* Open a web browser, and type in the IP Address of the Ubuntu Server, if the default Apache2 page loads that confirms the configuration is correct.

A black screen with white text

Description automatically generated

* Run the command “**sudo wireshark**” to show all network traffic on the server. Here we can see the web server traffic resulting from opening the web page in the previous screenshot.

**MySQL**

MySQL is a database management system that is open source. To connect and make it functional on the Ubuntu server, port 3306 will need to be opened.

1. **Command – “sudo ufw allow 3306 /tcp”.**

A screenshot of a computer

Description automatically generated

* This command is used to open port 3306, reload the firewall like we did for the web server (**sudo ufw reload**) and then run **sudo ufw status** to confirm the change has been made to this port (see screenshot below).



Reload the firewall **sudo ufw reload.**

A screenshot of a computer

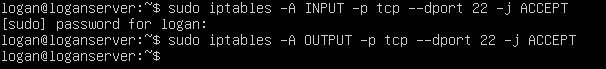
Description automatically generated

Shows the active firewall rules, including the new rule to allow traffic for MYSQL (Port 3306).

**SSH**

SSH, acronym for Secure Shell, is a network tool that allows more secure communication between two computers. SSH uses encryption as part of the ‘handshake’ between the two computers communicating with each other. We will need to configure the SSH service to allow both incoming and outgoing traffic.

1. **Command – “sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT” & “sudo iptables -A OUTPUT -p tcp –dport 22 -j ACCEPT”.**



* The commands to allow incoming traffic (**INPUT**) and outgoing traffic (**OUTPUT**). Although this was configured previously, it is good practice to confirm with the command **sudo ufw status**.

A screenshot of a computer

Description automatically generated

1. **Command – “sudo iptables -I INPUT -p tcp --dport 22 -j DROP”**

A black screen with white text

Description automatically generated

* To disallow incoming SSH traffic, use this command to tell the system to drop all packets coming into port 22. This will restrict the server from any incoming SSH access, which is a security risk for unauthorized access. Reload the firewall and attempt to ping the server on port 22. If the ping is unsuccessful that confirms the port is shutdown (*see below*).

A screen shot of a computer

Description automatically generated

**Allow or Block Specific Hosts / MAC Addresses**

1. **Command – “sudo iptables -A INPUT -s *IPaddress* -j DROP”**

A black screen with white text

Description automatically generated

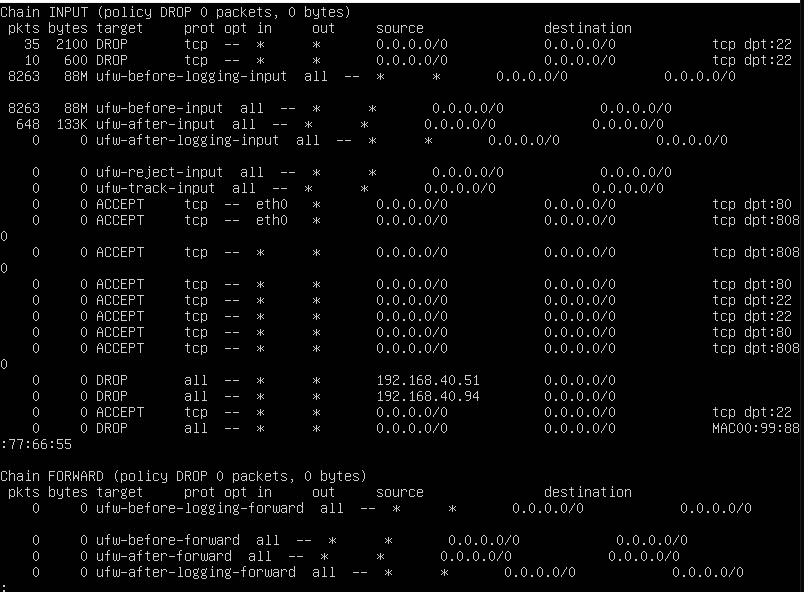
* To block a specific IP Address, we need to utilize a command like what we just used to block SSH traffic. **Iptables -A** **INPUT** tells the system to append (**-A**) a new rule to the iptables incoming traffic (**INPUT**). The **-s** option specifies an address to block incoming traffic from (**-j DROP**). I used my CentOS server IP Address as an example. To allow traffic from the address, simply use ‘**ACCEPT**’ at the end of the command instead of ‘**DROP’**.

1. **Command – “sudo iptables -A INPUT -m mac –mac-source *MACaddress* -j DROP”**



* Specifying a MAC Address to block is a little more complicated than blocking the IP Address. The start of the command is similar, but we use (**-m**) telling the server to match a MAC Address, and (**--mac-source *MACaddress***) to state the specific address to block. Just like before, **-j DROP** finishes the rule to drop all packets from this source. Alternatively, we can use **ACCEPT** at the end to allow the incoming traffic from the MAC Address.

The screenshot below shows the rules we have put into place with the firewall. To see this, run the command **sudo iptables -L -v -n | less** to view all the incoming and outgoing firewall rules. I like to pipe **| less** to have the option to arrow up and down through the output of this command.



**Block Telnet and Ping**

1. **Command – “sudo iptables -A INPUT -p –dport 23 -j DROP”**

A screen shot of a computer

Description automatically generated

* Telnet is a network tool that allows remote connections between computers on the same network. By default, Telnet uses port 23, so to block Telnet we will need to drop all incoming traffic to port 23. Just like before with SSH, use the command **Iptables -A** **INPUT** telling the system to append (**-A**) a new rule to the iptables incoming traffic (**INPUT**). Then specify TCP protocol and port 23 **(-p tcp --dport 23**) and use the option to drop the connections (**-j** DROP). To allow Telnet, simply use ‘**ACCEPT**’ at the end of the command instead of ‘**DROP’**.
* To test Telnet, run the command **telnet**, then run **open** and enter the IP Address of the server. The connection will be refused because of the firewall rule. A black screen with white text

  Description automatically generated

1. **Command – “sudo iptables -A INPUT -p icmp –icmp-type echo-request -j REJECT”**



* Ping is a network tool that is normally used to confirm connections to other computers. It works by using Internet Control Message Protocol (ICMP), which returns a reply if the device can be reached with ping. By default, ping is enabled on most devices; it can be a strategic security measure to have ping disabled in the event of a security breach. This command is telling the server to append (**-A**) a new rule to the iptables incoming traffic (**INPUT**). Then specifies ICMP echo requests (pings) and rejects them. Change **REJECT** to **ALLOW** to reenable ping in the future if desired. Reload the firewall and try to ping the server. If the ping fails like the example below, it means it worked!

A black screen with white text

Description automatically generated

Reload the firewall and try to ping the server. If the ping fails like this example, it means the rule is working as intended.

**Sources**

* <https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-firewall.html>
* <https://www.aholdengouveia.name/LinuxAdmin/iptables.html>
* <https://santoshk.dev/posts/2023/iptables-explained-an-introduction-to-linux-firewalling/>
* <https://unix.stackexchange.com/questions/412446/how-to-disable-ping-response-icmp-echo-in-linux-all-the-time>
* https://blog.rtsp.us/ufw-uncomplicated-firewall-cheat-sheet-a9fe61933330