

Brooklyn Dressel
Assignment 4

Abstract

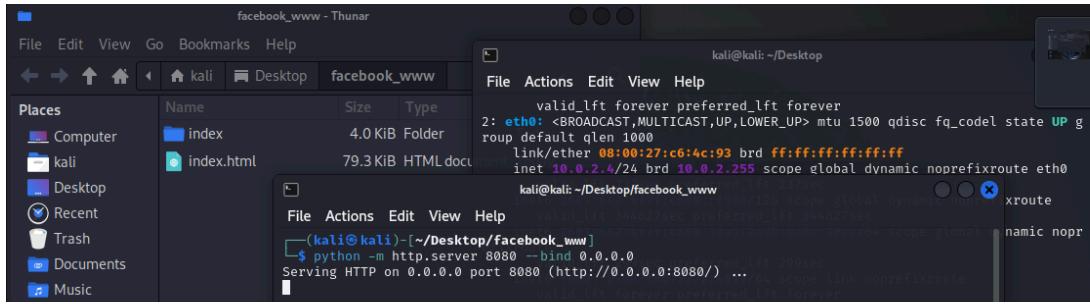
This project explores the use of iptables firewalls to control and manipulate network traffic. Focusing on creating, blocking, and redirecting traffic to and from two virtual machines. Additionally, the limitations of mac address filtering as a security measure are demonstrated and analyzed. Traffic redirection to a local web server is also tested, and used to see the impact of blocking specific devices. It becomes apparent that mac address filtering is not a secure method of access control as mac address spoofing is incredibly easy to perform. However, filtering traffic by IP addresses is proven to be fairly secure and even allows the ability to block all IP addresses under the same subnet mask. Packet analysis using Wireshark highlights the availability of communications along SSH servers, HTTP pages, and network traffic with firewall interference. The findings reveal the strengths and limitations of firewalls as it applies to the five main security service categories.

Introduction

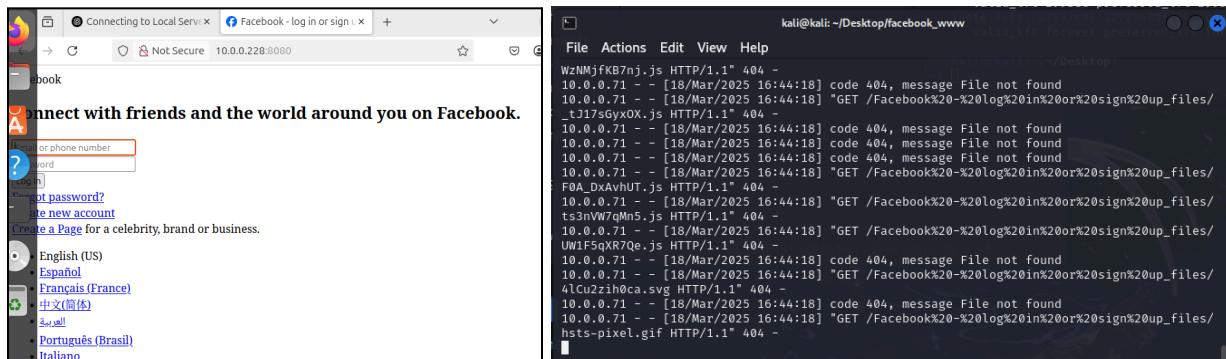
In this project I used a Kali Linux virtual machine to host a local website as well as a SSH server. I used the iptables firewall to block inbound and outbound traffic by blocking port number, ip address, as well as mac address. Additionally, I used an Ubuntu virtual machine to connect to the locally hosted website and the SSH server, the Ubuntu was also used to demonstrate the Kali's firewall capabilities. All traffic packets were captured using Wireshark on the Kali system. Both virtual machines were run on a Bridged Network to allow them internet access while also allowing the two virtual machines to communicate with each other. Additionally, all commands used during this project are provided separately.

Summary of Results

Upon opening and logging into the Kali Linux virtual machine, I opened Firefox and navigated to the Facebook website. I saved the site by right clicking on the page and created a folder on my desktop for easy access. Inside the folder I renamed the html file index, opened a python terminal inside the folder and ran command 1 to begin hosting the copy of the facebook website as a locally hosted website. With a response from the terminal, shown below, I knew that the website was successfully hosted.



After setting up the webpage on Kali, I used command 2 in order to get the ip address of my Kali machine, additionally I opened Wireshark to prepare to capture the upcoming network traffic. Then I opened the Ubuntu virtual machine and started Firefox, using the url `http://10.0.0.228` I was able to successfully connect to the Kali hosted webpage. It is interesting to note that the url automatically updated to `http://10.0.0.228/8080`, showing that this was in fact a locally hosted website. Since this was an http site, network traffic is not encrypted and plain to see in the packet captures. I was able to confirm this was the Kali hosted webpage as lots of feedback was sent in my Kali terminal, notifying me of the http requests.



Next, in order to test the limits of Kali's firewall capabilities, I moved to block `www.facebook.com` in its entirety. So, I first used command 3 to find the host, Facebook, information, specifically Facebook's ip address. Using the ip address, I ran command 4 in order to find the extent of `www.facebook.com`'s possible ip addresses, from this I found the network address with subnet mask to be `157.240.0.0/16`. Since Facebook could use any of the ip addresses within the range of the subnet mask. Next I ran command 5 to drop all traffic to that network. I ran command 6 to confirm that this rule had been created, the result below shows that any outgoing traffic to those ip addresses would be dropped.

```

File Actions Edit View Help
[(kali㉿kali)-~/Desktop]
$ host -t a www.facebook.com
www.facebook.com is an alias for star-mini.c10r.facebook.com.
star-mini.c10r.facebook.com has address 157.240.22.35

[(kali㉿kali)-~/Desktop]
$ whois 157.240.22.35 | grep CIDR
CIDR: 157.240.0/16

[(kali㉿kali)-~/Desktop]
$ iptables -A OUTPUT -p tcp -d 157.240.0.0/16 -j DROP
iptables v1.8.10 (nf_tables): Could not fetch rule set generation id: Permission denied (you must be root)

[(kali㉿kali)-~/Desktop]
$ sudo iptables -A OUTPUT -p tcp -d 157.240.0.0/16 -j DROP
[sudo] password for kali:
[2/Mar/2025 17:00:11] "GET /Facebook%20-%20log%20in%20or%20sign%20up" code 0/0, message File not found

[(kali㉿kali)-~/Desktop]
$ sudo iptables -L -v -n
[sudo] password for kali:
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out    source         destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out    source         destination
Chain OUTPUT (policy ACCEPT 103 packets, 10544 bytes)
 pkts bytes target     prot opt in     out    source         destination
 245 14700 DROP      tcp   --  *       *        0.0.0.0/0          157.240.0.0/16

```

From there, I ran command 7 to block any inbound traffic headed to port 8080. This blocks all server requests, and I ran command 8 to ensure that the rule was implemented. This command is similar to command 6 but it only returns rules that apply to port 8080. Then I attempted to connect to the locally hosted website once more from my Ubuntu machine using the same url before. However, this time the connection timed out and I was unable to connect to the server.

```

File Actions Edit View Help
[(kali㉿kali)-~/Desktop]
$ sudo iptables -A INPUT -p tcp --dport 8080 -j DROP

[(kali㉿kali)-~/Desktop]
$ sudo iptables -L -v -n | grep 8080
 0  0 DROP      tcp  --  *       *        0.0.0.0/0          0.0.0.0/0

```

The connection has timed out

The server at 10.0.0.228 is taking too long to respond.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

The next step to testing the Kali Linux machine's firewall was to block Ubuntu by its mac address. Naively, I simply attempted to block all incoming traffic from my Ubuntu's mac address, which I had found using command 2, so I ran command 9 and confirmed the rule was in place using command 10. In order to test if this worked I ran command 11 in order to start hosting an SSH server for my Ubuntu to attempt to connect to.

```

[(kali㉿kali)-~]
$ sudo service ssh start
[sudo] password for kali:

[(kali㉿kali)-~]
$ sudo service ssh stop

```

Then I reopened Ubuntu to connect to the SSH, and was able to. So I began looking into what the reason for this was. I double checked the Ubuntu's mac address and ensured the rule was in place again. Finally, I came to the realization that both virtual machines shared the same mac

address, and that this would interfere with the firewall rule. The screenshot below shows my attempts at problem solving this issue before I realized the problem.

```
(kali㉿kali)-[~]
$ sudo iptables -L -v -n | grep 08:00:27:c6:4c:93
  0      0 DROP      all  --  *      *      0.0.0.0/0      0.0.0.0/0
    MAC 08:00:27:c6:4c:93

(kali㉿kali)-[~]
$ sudo systemctl restart netfilter-persistent
sudo: systemctl: command not found

(kali㉿kali)-[~]
$ sudo systemctl restart netfilter-persistent
Failed to restart netfilter-persistent.service: Unit netfilter-persistent.service
is not found.

(kali㉿kali)-[~]
$ sudo apt update
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.7 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [49.3 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [115 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [267 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [203 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [884 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [0.8 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [24.3 kB]
95% [3 Contents-amd64 store 0 B]||

(kali㉿kali)-[~]
$ sudo macchanger eth0 -r
Current MAC: 08:00:27:c6:4c:93 (CADMUS COMPUTER SYSTEMS)
Permanent MAC: 08:00:27:c6:4c:93 (CADMUS COMPUTER SYSTEMS)
New MAC: 52:4d:9e:d9:8d:43 (unknown)

(kali㉿kali)-[~]
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
            inet6 ::1/128 scope host noprefixroute
                valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
    qlen 1000
        link/ether 52:4d:9e:d9:8d:43 brd ff:ff:ff:ff:ff:ff permaddr 08:00:27:c6:4c:93
        inet 10.0.0.228/24 brd 10.0.0.255 scope global dynamic noprefixroute eth0
            valid_lft 166807sec preferred_lft 166807sec
            inet6 2601:1642:4f7f:c880::f926:128 scope global dynamic noprefixroute
                valid_lft 344035sec preferred_lft 344035sec
            inet6 2601:1642:4f7f:c880:39a5:2c95:b2a8:10ee/64 scope global dynamic noprefixroute
                valid_lft 298sec preferred_lft 298sec
            inet6 fe80::b769:c00:3272:481f/64 scope link noprefixroute
                valid_lft forever preferred_lft forever
```

To solve this, I ran command 12 to change the mac address of my Kali virtual machine, confirmed this by running command 2. Now that my virtual machines had distinct mac addresses, I attempted to connect to the Kali SSH server from Ubuntu using command 13. The screenshot below shows that this was unsuccessful and the connection timed out. Additionally, Ubuntu was unable to connect to the locally hosted website. Since I had confirmed that this method was successful I ran command 14 to remove this rule and command 10 to confirm the change so I could attempt another rule for the Kali firewall.

```
(kali㉿kali)-[~]
$ sudo iptables -A INPUT -m mac --mac-source 08:00:27:c6:4c:93 -j DROP
[sudo] password for kali:

(kali㉿kali)-[~]
$ sudo iptables -L -v -n | grep 08:00:27:c6:4c:93
  0      0 DROP      all  --  *      *      0.0.0.0/0      0.0.0.0/0
    MAC 08:00:27:c6:4c:93

(kali㉿kali)-[~]
$ sudo iptables -D INPUT -m mac --mac-source 08:00:27:c6:4c:93 -j DROP

(kali㉿kali)-[~]
$ sudo iptables -L -v -n | grep 08:00:27:c6:4c:93

brooklyn@brooklyn-VirtualBox:~
brooklyn@brooklyn-VirtualBox:~$ ssh kali@10.0.0.228
ssh: connect to host 10.0.0.228 port 22: Connection timed out
brooklyn@brooklyn-VirtualBox:~$ ssh kali@10.0.0.228
ssh: connect to host 10.0.0.228 port 22: Connection timed out
brooklyn@brooklyn-VirtualBox:~$
```

The next attempt at checking the capabilities of the Kali iptables firewall was to block incoming traffic to port 22, which would be to the SSH server specifically. In order to do this I ran command 15 to create this new iptables rule, confirmed it was enabled using command 16. To test the firewall I ran command 13 in Ubuntu, and once again could not connect to the Kali SSH server. Since this was successful I ran command 17 to disable this rule and command 16 to confirm it.

```
(kali㉿kali)-[~]
$ sudo iptables -A INPUT -p tcp --dport 22 -j DROP
[sudo] password for kali:

(kali㉿kali)-[~]
$ sudo iptables -L -v -n | grep 22
  0      0 DROP      tcp  --  *      *      0.0.0.0/0      0.0.0.0/0
    tcp dpt:22

(kali㉿kali)-[~]
$ sudo iptables -D INPUT -p tcp --dport 22 -j DROP

(kali㉿kali)-[~]
$ sudo iptables -L -v -n | grep 22

brooklyn@brooklyn-VirtualBox:~
brooklyn@brooklyn-VirtualBox:~$ ssh kali@10.0.0.228
ssh: connect to host 10.0.0.228 port 22: Connection timed out
brooklyn@brooklyn-VirtualBox:~$
```

The final test for Kali's firewall was port forwarding. First I ensured that port forwarding, I checked to see if it was by running command 18, the output of 0 meant that it was not and I had to enable it using command 19. Then I ran command 20 to forward all incoming traffic headed to port 80 and send it to port 8080 instead.

```
kali㉿kali:[~]
File Actions Edit View Help
(kali㉿kali)[~]
$ cat /proc/sys/net/ipv4/ip_forward
0

(kali㉿kali)[~]
$ echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
[sudo] password for kali:
1

(kali㉿kali)[~]
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080
```

Now that the rule was set up, I next went back to Ubuntu in order to test it. In order to do this I ran command 21 from the terminal, the output is shown below, saying I was able to successfully connect. Note that the ip address in this command changed as I forgot to take a screenshot and redid this section of the project in order to properly capture the response in Ubuntu. The Ubuntu screenshot below show the response saying a request was sent to port 80, and the Kali screenshot shows that it served an http request on port 8080.

```
brooklyn@brooklyn-VirtualBox:~$ wget -o- http://134.154.79.231
--2025-03-20 18:00:22-- http://134.154.79.231/
Connecting to 134.154.79.231:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 81229 (79K) [text/html]
Saving to: 'index.html.4'

    OK ..... 63% 265M 0s
    50K ..... 100% 137M=0s

2025-03-20 18:00:22 (197 MB/s) - 'index.html.4' saved [81229/81229]
```

```
kali㉿kali:[~]/Desktop/facebook_www
File Actions Edit View Help
(kali㉿kali)[~]/Desktop/facebook_www
$ python -m http.server 8080 --bind 0.0.0.0
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
134.154.79.139 - - [20/Mar/2025 21:00:20] "GET / HTTP/1.1" 200
```

As extra confirmation I ran command 21 again to ensure the response was consistent. And as added confirmation I ran command 22 to see the iptable rules, the screenshot below displays that 2 packets have been redirected from port 80 to port 8080.

```
(kali㉿kali)[~]
$ sudo iptables -t nat -nvL PREROUTING
[sudo] password for kali:
Chain PREROUTING (policy ACCEPT 40 packets, 6751 bytes)
  pkts bytes target  prot opt in   out   book source           destination
      2   120  REDIRECT  tcp  --  *     *       0.0.0.0/0        0.0.0.0/0          tcp dpt:80
  redirect ports 8080
      0     0  REDIRECT  tcp  --  *     *       0.0.0.0/0        0.0.0.0/0          tcp dpt:80
  redirect ports 8080
```

Conclusion

Analysis

Review all of the packets captured by Wireshark. Comment on the different types of traffic seen, including SSH and HTTP. What type of encryption does SSH use? Find the key exchange packets.

Filter and save only several relevant packets. Include this filtered capture file.

Conclusion

In addition to your conclusion, add the following:

- What additional steps are required to turn this into an attack?
- How can IPTables be used to improve security?
- What is mac address filtering?
- How does macchanger do? What does this mean about mac address filtering?
- Describe how firewall software does or does not provide:
 - Authentication
 - Access control
- Data confidentiality
- Data integrity
- Non-repudiation

Analysis of the packets reveals that all SSH traffic appears as encrypted TCP packets sent and received on port 22. This is why when we used iptables to block incoming packets on port 22, we were unable to connect to the SSH server from Ubuntu. Additionally, SSH uses a Diffie-Hellman key exchange that is clear to see in the packet captures, after that all packets exchanged are encrypted. For the HTTP traffic, since its non encrypted web traffic (HTTPS is encrypted web traffic), all get requests, responses, and urls are plain to see.

Overall, this project demonstrates the capabilities and variety of techniques you can use to limit traffic to and from your network. There is also the possibility that these limitations can be used maliciously and with a few added steps can be turned into an attack. For instance, if an attacker were to perform a Man-in-the-Middle attack, the first step would be to partition the network. This could be done by using firewalls to stop traffic from heading from one user to another, the attacker could intercept this traffic in the middle by using the firewall or perhaps use the port forwarding technique to have all traffic sent somewhere other than the user's intended destination. If a Man-in-the-Middle attack is used to facilitate the key exchange between two users (Diffie-Hellman key exchange is particularly vulnerable to this style of attack), then the attacker could gain access to all further sent encrypted packets (thus making SSH traffic also susceptible to this style of attack).

The use of iptables can be used to increase and improve security simply by blocking unwanted or possibly malicious traffic from coming onto the network. The catch to this is that the user and network must know some information about the attacker before they are attacked. To solve this issue the user could block all traffic on the network and only allow trusted users and network access, while all others are blocked.

One of the ways that other devices can be blocked using iptables is by mac address filtering. Using this method, all incoming traffic from a particular device (as known by its mac address) can be blocked. However, as we saw during this project using macchanger, it is *incredibly* easy to change your device's mac address and bypass this firewall. However, because of the ease of use of this method, it is oftentimes used to block user's when there are low-security risks, such as at an internet cafe's wifi.

Finally, this project has given insight into how firewall technologies help users in the five service categories. Firstly, we have seen that firewall technologies do not provide authentication outright. There is a level of authentication required for users to bypass firewalls, meaning that authenticated users can access networks, however the firewall itself does not provide the means to authenticate users. Next, access control, yes, this is the primary use of firewalls. They are designed to control access to only specific users while denying access to all other users or simply blocking specific users. Similar to authentication, firewalls do not provide data confidentiality as they do not encrypt anything. However, by limiting the users who have access to data, it does help prevent attackers from getting access and thus seeing the data. Firewalls do not provide data integrity. There is no method in firewalls that can tell if data has been modified or tampered with, thus there are no data integrity checks. However, similar to above, if an attacker does not have access to the data, it helps to ensure that it cannot be altered in the first place. Finally, firewalls can provide non-repudiation to a small degree. The most common way to ensure non-repudiation is with digital signatures, and firewalls do not have the ability to do this. However, since firewalls are capable of logging network activity, there is a certain degree of being able to tell which users have done certain actions. However, this is not fool-proof and not all-encompassing.