Computer Network Lab: Lab #1

塗大爲
Department of Computer Science and Information Engineering
National Taiwan University
b07902024@ntu.edu.tw
陳威翰
Department of Computer Science and Information Engineering
National Taiwan University
b07902132@ntu.edu.tw
黃于軒
Department of Computer Science and Information Engineering
National Taiwan University
b07902134@ntu.edu.tw
楊子平
Department of Computer Science and Information Engineering
National Taiwan University
b07902136@ntu.edu.tw

I. Environment & Installation

The WLAN authentication system is implemented in Rust[1] using the Rocket[2] web framework. The project should build and run comfortably on most Linux distros as follows:

1. Follow the official guide[3] to install Rust and Rust-related utilities.
2. We use `sqlite3` for the persistence of user data. On Ubuntu, for example, one should download the *library* version of the package `libsqlite3-dev`.
3. Initialize the database with `sqlite3 db.sqlite < create_users.sql`.
4. Set up the WiFi hotspot and preliminary firewall rules. We use *linux-wifi-hotspot*[4] to bind the WiFi access point onto the wireless interface[5].

```
create_ap --daemon -n wlp3s0 'MyAccessPoint' '
    '
sysctl net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -o enp2s0 -j
    MASQUERADE
iptables -t nat -A PREROUTING -i ap0 -p tcp -j
     REDIRECT --to-ports 8000
export IFNAME=ap0
```

Replace `wlp3s0` with your WiFi interface and `enp2s0` with the outgoing interface. The environment variable `IFNAME` refers to the interface name of the created WiFi access point and is used by the web server to produce firewall rules.

5. Download the dependencies and compile the project with `sh rustup set nightly cargo build --release`
6. Execute `./target/release/wlan` to start the service on port 8000.

II. Implementation

As mentioned previously, we use the Rocket web framework to handle HTTP requests. Additionally, Rocket provides us the ability to interact with our database through Diesel[6] and dynamically render web pages using Tera[7].

When users connect to the WiFi access point for the first time, the requests will be redirected to our service by the rule

```
iptables -t nat -A PREROUTING -i ap0 -p tcp -j
     REDIRECT --to-ports 8000
```

and the login/registration page will be shown. Users can register their accounts and obtain permanent login permissions afterward. The user information is maintained in our database, and the passwords are hashed and verified using bcrypt[8]. After successfully logging in, the IP address of the client will be extracted from rocket_client_addr[9] and the following rules will be inserted:

[1] https://www.rust-lang.org/
[2] https://rocket.rs/
[3] https://www.rust-lang.org/tools/install
[4] https://github.com/lakinduakash/linux-wifi-hotspot
[5] In our testing, the internal wireless interface is used, with the wired Ethernet interface as the outgoing interface.
[6] https://diesel.rs/
[7] https://tera.netlify.app/
[8] https://docs.rs/bcrypt/0.9.0/bcrypt/
[9] https://docs.rs/rocket-client-addr/0.4.3/rocket_client_addr/index.html

```
iptables -t nat -I PREROUTING 3 -i ap0 -s <IP>
     -j ACCEPT
iptables -t filter -A FORWARD -i ap0 -s <IP> -
    m conntrack  --cstate NEW -j ACCEPT
iptables -t filter -A FORWARD -s <IP> -m
    conntrack --cstate ESTABLISHED,RELATED -j
    ACCEPT
iptables -t filter -A FORWARD -d <IP> -m
    conntrack --cstate ESTABLISHED,RELATED -j
    ACCEPT
```

The first rule overwrites the aforementioned redirection rule so that traffic coming from the specific IP will be accepted at the PREROUTING stage and proceed to the FORWARD chain in the filter table. In the FORWARD chain, traffic from/to the specific IP is accepted due to the remaining three rules. Finally, the accepted packets are masqueraded and sent to the original destination via the interface with internet access. The interaction between our service and `iptables` is made possible by rust-iptables[10].

To block a specific IP address, we simply delete the four rules added above. Additionally, the rules

```
iptables -t filter -I FORWARD 1 -i ap0 -s <IP>
     -j DROP
iptables -t filter -I INPUT 1 -i ap0 -s <IP> -
    j DROP
```

are added to prevent that user from accessing the login page. Unblocking is implemented as deleting the two extra rules added when blocking the user. That being said, the user has to re-login to our website to get access to the Internet because the previous rules are removed.

Lastly, to monitor the bandwidth of each connected IP addresses, the output of the command `iptables -vnxL FORWARD -t filter` is parsed. The `-x` flag forces `iptables` to emit full numeric values of the statistics instead of human-readable ones like `60K`. Because our portal only *forwards* packets to the internet, checking the FORWARD chain suffices. Since there are three rules corresponding to each of the IPs, the sum of these values is calculated and rendered to the frontend.

III. Situational Questions

In this section, we only consider IPv4, as the settings to support IPv6 should be similar.

A. Your website is working on port 8080 and can only be accessed by 140.112.0.0/16. How to modify your iptables to block unavailable users?

To block users outside of `140.112.0.0/16` from accessing the website, adding the following two rules suffices:

```
iptables -A INPUT -p tcp -s 140.112.0.0/16 --
    dport 8080 -j ACCEPT
iptables -A INPUT -p tcp --dport 8080 -j DROP
```

The first rule accepts all TCP traffic from `140.112.0.0/16` to port 8080 and the second one drops all other traffic to that port.

[10]https://docs.rs/iptables/0.4.3/iptables/

B. Behind the machine, there is a SSH server which locates at `192.168.10.2` in the `eth1`. People who want to connect to SSH server from `eth0` need to connect the machine at port 2222 and the machine will redirect the flow to the SSH server at port 22. How to configure the iptables?

We assume that `192.168.10.2` is already routed properly through `eth1` on the machine.

```
iptables -t nat -A PREROUTING -p tcp --dport
    2222 -i eth0 -j DNAT --to-destination
    192.168.10.2:22
iptables -A FORWARD -i eth0 -p tcp -d
    192.168.10.2 --dport 22 -j ACCEPT
iptables -t nat -A POSTROUTING -p tcp -d
    192.168.10.2 --dport 22 -j MASQUERADE
```

The first rule rewrites the destination address and port to those specified. The second rule ensures that the corresponding traffic is forwarded, while the third rule rewrites the source IP address to that of the outgoing interface.

Note that one may also need to enable IPv4 forwarding with

```
sysctl net.ipv4.ip_forward=1
```