Task 5 - Firewalls Walkthrough

What is a Firewall

Firewalls are devices, or programs (as part of a system) that is used to monitor and filter network traffic. There are two broad classes of firewalls that you will probably run into. These classes are **Stateful** and **Stateless** firewalls. To briefly cover the two types, a **Stateless** firewall does not take the connection state into account as long as the packet does not satisfy any of the rules that would reject the packet it will be accepted (Even if the packet makes no sense - such as a FIN packet for a connection that has never been established). A **Stateful** firewall operates in a similar manner but tracks and can consider the state of the connection the packet is a part of as an additional variable.

Firewalls can be a standalone system often used at the edge of a network, a system integrated into a router (as you likely have in your homes!), and it can also be a program or system integrated into an end-system (This is a host-based firewall).

Why do we want to utilize Firewalls

As with the previous task, this is hopefully quite intuitive. Firewalls allow us to create systems and rules so we can control the types of connections we allow, not only to our host systems - but also to the network as a whole.

Solutions

The following are three methods of solving this problem, as we are using containers using the host-based firewall is a bit more difficult as Docker will create and edit the IPTables configurations on its own. We can implement this isolation using just the objects and options provided by Docker

Docker

1. Create a docker network (We can just do this on the default network too!) **This may have been done for you**
    - docker network create --internal --driver bridge web-container
2. Stop the website container
    - docker ps
    - docker rm -f <Container-Name/ID>
3. Stop the API container
    - docker ps
    - docker rm -f <Container-Name/ID>
4. Stop the SQL container
    - docker ps
    - docker rm -f <Container-Name/ID>
5. Stop the proxy container
    - docker ps
    - docker rm -f <Container-Name/ID>
6. Stop Nginx **ONLY if there is NO Nginx container**
    - sudo systemctl stop nginx

7. Start the website container. The main difference now is there will be **no** port mappings, and it will be attached to our new network
    - docker run -d --name <web-container-name> --network website_proxy_network --hostname web-container <Container-Image-Name>
8. Start the API container. The main difference now is there will be **no** port mappings, and it will be attached to our new network
    - docker run -d --name api-server --network website_proxy_network --hostname api-server <Container-Image-Name>
9. Start the API container. The main difference now is there will be **no** port mappings, and it will be attached to our new network
    - docker run -d --name database --network website_proxy_network --hostname database <Container-Image-Name>
10. Start the proxy container. The main difference now is it will be the only container with port mappings and will be attached to our new network
    - docker run -d -p 80:80 -p 443:443 --name proxy-container --network website_proxy_network -v /etc/nginx/nginx.conf:/etc/nginx/nginx.conf --hostname proxy-container nginx

If the proxy is local we can do the following (It binds to 127.0.0.1)

1. Stop the website container
2. Start the website container
    - docker run -d --name web-container --hostname web-container -p 127.0.0.1:8080:80 <Container-Image-Name>

Uncomplicated FireWall (UFW)

This method is unreliable when using a Docker container. This is due to the changes Docker makes to the IPTables which may override the changes UFW makes.

1. Install ufw
    - sudo apt install ufw
    - **Note the package manager may change between distros**
2. Create a rule to allow from localhost
    - ufw allow from 127.0.0.1 proto tcp to 127.0.0.1 port 8080
3. Create the rule to block port 8080 **Port may not be 8080**
    - ufw deny 8080
4. Enable UFW
    - ufw enable

IPTables

1. Add rule to allow loopback
    - iptables -A INPUT --src 127.0.0.1 -j ACCEPT
2. Add rule to block port 8080
    - iptables -A INPUT -p tcp --dport 8080 -j DROP

If the web-server is a container it become a bit complicated, since we would need to add rules to the DOCKER-USER chain to block communications to the container. However since the server listens on port 80, all outbound http communications will also be blocked.