Network Security

Assignment: Network Layer

This week's assignments utilize Docker to create the different lab environments. Docker is a wrapper around your system's kernel, allowing you to create lightweight virtual environments, without necessitating a full-fledged virtual machine. You can read more about Docker from their website, docker.com.

We recommend using Docker in combination with a Linux machine. If you do not have one, you can use a virtual machine. These assignments may work on other operating systems as well, though this has not been tested.

Installing Docker is quite easy. Full instructions can be found on Docker's website. For Ubuntu, the following should work from the default package repositories:

```
sudo apt install docker.io docker-compose
```

When you have successfully installed Docker, you should unpack the assignment's files somewhere on your system, navigate to those files and issue the following commands:

```
docker-compose build
docker-compose up -d
```

This will show you a list of different containers that have started. In order to connect to the container, you can execute the command below, where `attacker` is the name of the container:

```
docker exec -ti attacker bash
```

The files inside the Docker containers are lost after the container has been destroyed. We have therefore set-up file sharing between the assignment's directory on your host and the `/data` directory in the `attacker` containers. If you need to adjust this, you can simply modify the `volumes` line in the `docker-compose.yml` file, and run the `up` command again.

If you need to, you can monitor the network between the Docker containers on your host machine by attaching Wireshark or tcpdump to the `br-`device created by the assignment.

After finishing the assignment, you can destroy the environment using the command below. First, make sure to copy your developed files outside the container or in the shared data directory. Otherwise, your assignment will be lost.

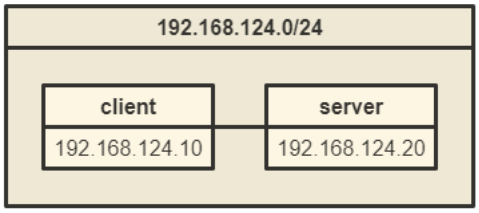
```
docker-compose down
```

Once you are completely done with these assignments, we recommend you execute a full system prune to remove the container cache from your system and free up some disk space.

```
docker system prune
```

ICMP spoofing (15 points)

Consider the simple client-server setup below:



The server hosts a secret on port 80, but this port is secured by a ICMP port knocking mechanism. The port knocking mechanism works by simply sending two ICMP ping packets with a payload length of 22 to the server:

```
ping -c 2 -s 22 192.168.124.20
```

ICMP is only accepted from the client's IP address, though the secret will be world-readable when the port knocking has succeeded.

Goal

Craft a Python3 script with the name `icmp_spoof.py` to target this ICMP port knocking mechanism from the attacker's machine, spoofing the client's IP address in the process. The script should accept the IP address we are targeting, the IP address we are spoofing the request from and the packet length as arguments, i.e.

```
python3 icmp_spoof.py 192.168.124.20 192.168.124.10 22
```

The assignment is considered successful if you are able to successfully gather the secret from the server after executing the script.

```
curl http://192.168.124.20
```

Hint: Once the port knock succeeded, you can close the port again by restarting the server:

```
docker-compose restart server
```

Submission Instructions

Your script should run on the attacker container and exit normally (exit code 0) when it is done. After completing the assignment you should only submit the resulting script. A `requirements.txt` file should be submitted if you use any modules from the Python Package Index (pip).

ICMP covert channel (25 points)

In this assignment, we will set up a covert channel between two hosts. Considering the setup below, the victim's machine has been firewalled from the attacker's machine, only allowing ICMP traffic through.

Goal

Craft two Python3 scripts that allow communication between the two hosts, i.e. using an ICMP covert channel. Your message should not be easily readable on the wire, i.e. you must implement some form of encryption. Also make sure that you do not use ICMP's Echo protocol, as that may cause the covert channel to be echoed back.

Your first script, called `covert_sender.py`, should accept two arguments: the IP address of the receiver, and the message to send. The second script, `covert_receiver.py`, does not accept any arguments, but should simply listen for the covert messages, and print them to the console.

On the sending side, we would expect to be able to do something like this:

```
python3 covert_sender.py 192.168.124.10 "Hello"
python3 covert_sender.py 192.168.124.10 "World"
```

We expect the following output on the receiving side:

```
Hello
World
```

Hint: In order to connect to the different containers, you can execute the commands below in different terminal windows.

```
docker exec -ti attacker bash
docker exec -ti victim bash
```

If you need some dependencies in the victim container, you'll discover that the firewall will block this. You can fix this by temporarily disabling the firewall and re-enabling it afterwards:

```
docker exec victim bash -c "iptables-restore < /etc/iptables/accept-rules.v4"
docker exec victim bash -c "iptables-restore < /etc/iptables/rules.v4"
```

Submission Instructions

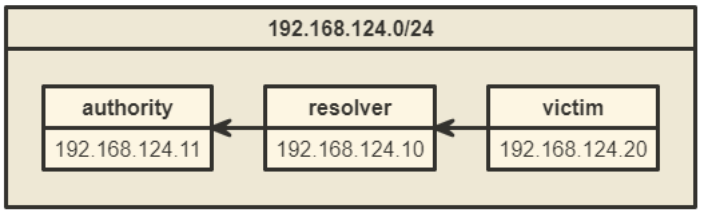
Your scripts should be able to run interchangeably on either the attacker and victim containers. The `covert_sender.py` script should exit normally (exit code 0) after it is done; the `covert_receiver.py` should run indefinitely.

After completing the assignment you should only submit the resulting scripts. A `requirements.txt` file should be submitted if you use any modules from the Python Package Index (pip).

DNS Hijacking (30 points)

This assignment concerns an application server that regularly checks for updates. However, the update server is not properly secured, so if we can manage to redirect the update to our own malicious update server, we can exploit this process. In this exercise, we will target the DNS infrastructure to accomplish this.

The vulnerable application will be running on 192.168.124.20. It periodically resolves the domain name `update-server.updateserver.corp` from its DNS server at 192.168.124.10.



Goal

The goal of this exercise is to intercept the DNS request sent by the application, and craft a response faster than the local DNS server is able to, redirecting the application to another IP address. Do *not* attack other DNS requests than the DNS request for the update server from the victim to the resolver.

You should produce a Python3 script called `dns_inject.py`, accepting four arguments:

1. The interface to listen and send on
2. The IP address of the victim
3. The domain name of the DNS request
4. The IP address we want the answer to be

For instance:

```
python3 dns_inject.py "br-dns" "192.168.124.20" \
          "update-server.updateserver.corp" "1.2.3.4"
```

You can monitor the application by following the logs of the victim:

```
docker-compose logs -f victim
```

The assignment is considered successful if the victim logs show something like this:

```
Querying update-server.updateserver.corp
Got answer {'1.2.3.4'} for update-server.updateserver.corp
Exploit succeeded!
```

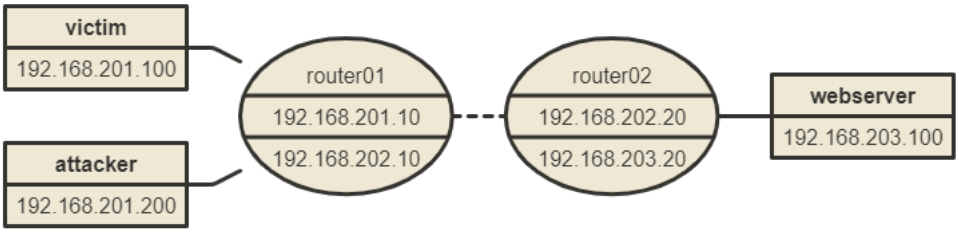
Note: We have delayed the DNS server in this assignment's environment. Otherwise, you would never be able to monitor for the DNS request, craft the DNS response and inject it faster than the DNS server is able to. In reality, the DNS server would be somewhere else on the network and/or the internet, allowing for this type of attack.

Submission Instructions

Your script should run on the attacker container and exit normally (exit code 0) when it is done. After completing the assignment you should only submit the resulting script. A `requirements.txt` file should be submitted if you use any modules from the Python Package Index (pip).

RIP Hijacking (30 points)

Consider the following network diagram:



The attacker and victim are both connected to the same router (router01). The webserver is connected to a different router (router02). The victim is attempting to connect to the webserver over port 80. Between router01 and router02, RIP version 1 is used to announce routes.

Goal

You should craft a Python3 script that sends RIP messages from the attacker machine to router01 to make sure that all traffic destined towards the webserver, is instead routed to you. The script should be named `rip_hijack.py` and take three arguments: the target IP (i.e. the router), the network address to reroute, and the IP address to route to. For instance:

```
python3 rip_hijack.py "192.168.201.10" "192.168.203.0/24" "192.168.201.200"
```

The assignment is considered successful when the victim performs a HTTP request against the attacker machine.

Hint: It may be useful to connect to one of the routers to see the state of the RIP routing table. You can do this as follows:

```
docker exec -ti router01 vtysh
show ip rip
# or
show ip rip status
```

Submission Instructions

Your script should run on the attacker container and exit normally (exit code 0) when it is done. After completing the assignment you should only submit the resulting script. A `requirements.txt` file should be submitted if you use any modules from the Python Package Index (pip).

Make sure your assignment conforms to the in- and output described in the Goal subsection. Your assignment is partly graded through an automated system. Any deviations from the described in- and output can affect your grade.