Chair of IT Security
Brandenburg University of Technology

Asya Mitseva, M.Sc.
Fabian Mildner, B.Sc.
Prof. Dr.-Ing. Andriy Panchenko

Summer Term 2025

**Part II: Network Security**

**Deadline: 14th May 2025**

# Cyber Security Lab (Ethical Hacking)
# ICMP Redirect Attacks

## Introduction

Routing belongs to the fundamentals of large scale networking. While in the scope of local area networks (LAN), all clients are typically connected by a single switch, in large scale networks multiple different sub-nets are connected using routers. The task of these routers is to forward IP packets between different sub-nets. However, sometimes, a router receives packets outside the scope of its responsibility, or it simply reaches the limit of its capacity. The ICMP redirect message is a special control packet, which allows the router to tell the client to use a different router for sending the subsequent packets to the intended destination. So the router can redirect the traffic if it believes that the transmission stream is not optimally or even incorrectly routed.

While at first glance, this feature seems to provide natural functionality, it comes with a certain drawback. A malicious actor can simply force clients to take certain routes by sending them spoofed ICMP redirect messages. Thus, the attacker is able to change the route in such a way that they become a man-in-the-middle, eavesdropping on the connection. As a result of a man-in-the-middle attack, the adversary may obtain sensitive communication data. The obtained secrets could be partial information of a Diffie-Hellman key exchange, or even a whole symmetric key. This makes the attack under consideration extremely dangerous if there are no countermeasures installed.

In this laboratory exercise, you should mount a basic ICMP redirect attack to establish yourself as the man in the middle. Once you have obtained a position between the client and a server, you should start to listen and modify sent packets. Basic knowledge of the IP and ICMP protocol as well as routing is recommended for this practical sheet. We recommend that you use a Debian-based Linux operating system to solve this laboratory exercise.

# Preparation of the Experimental Setup

For this laboratory exercise, you need to simulate a small routed network infrastructure. The bare minimum to study the attack must include a client machine, a service provider (server), and an intermediate router responsible for the traffic between the client and the server. Additionally, the attacker needs to have a malicious router and a client in the same sub-network as the victim. To get you started quickly, we provide you with a configuration that allows you to deploy a minimal experiment infrastructure by means of virtual devices. In the following, we describe how to deploy the provided virtual environment and how to set up the network.

## An overview of the network topology

Before discussing any technical details, we give a brief overview over the used network infrastructure. You will have two sub-nets, namely the local network `10.9.0.0/24` and the remote network `192.168.60.0/24`. We assume that the victim user and the adversary are members of the local network `10.9.0.0/24`. In the same network, the attacker operates his own router. The second network `192.168.60.0/24` consists of two service providers represented by two simple machines. The router connecting both networks possesses two interfaces with IP addresses `10.9.0.11` and `192.168.60.11`, respectively, thus having access to both subnets.

Overall, you should obtain a network topology as shown in Figure 1.

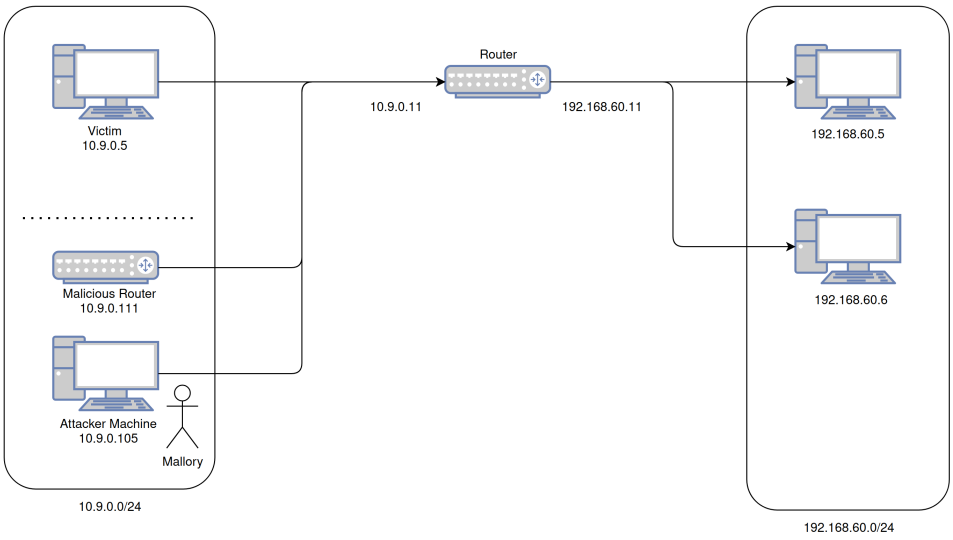



Figure 1: Overview of the laboratory network.

## Setting up the network devices

Each of the devices shown in Figure 1 is provided through a docker container [2]. In the following, we describe how to set up these containers.

For the purpose of this laboratory exercise, you will host the entire network on the same system. You will use docker to host the network infrastructure on your virtual machine or a comparable linux environment with docker and docker-compose installed. `docker-compose` [3] allows you to easily build, start, and stop the collection of containers as defined in the `docker-compose.yml` configuration file provided by us.

1. **Creating and starting the infrastructure:** To setup the infrastructure, navigate to the folder in which you saved the `docker-compose.yml` file provided by us. In addition, you should ensure that a `volumes` folder exists within the same directory. From there, you can execute the command:

   ```
   $ sudo docker compose up
   ```

   The docker-compose tool will automatically download the needed image files for the containers, create the required networks and connect and configure containers as specified in the `docker-compose.yml` file. The containers will run as long as you do not stop the command. Afterwards, you should be able to see running containers using `docker ps` (e.g. run in a different terminal).

   **Please Note:** Some of you may be familiar with the command `sudo docker-compose up` instead. This is also a possible option, however, `docker compose up` is the more modern solution, which is why we suggest it. It should be noted that there are slight behavioral differences between these commands.

2. **Stopping the infrastructure:** There are two main ways to stop the running containers. On the one hand, you can simply stop the command issued before using `CTRL + C` from the terminal window from which the network was launched. On the other hand, you can also issue a `sudo docker compose stop` command from within the same folder as the `docker-compose.yml` file to achieve the same effect. This procedure has the advantage of also being usable when you started the containers in detached mode, i.e. without keeping the previous command in the foreground of your terminal window.

3. **Cleaning up:** If you are running the infrastructure on something other than a disposable virtual machine, you may want to clean up the containers created by docker compose after you are done. You can do this with the `sudo docker compose down` command. Note that this command may miss some containers if you performed changes to the configuration file over time. You can also use `docker container prune`, `docker network prune` and `docker image prune`. Please be aware that this will delete all unused docker containers, networks and images, which may play a role if you already use docker for other purposes than just this lab.

## Working with the devices.

In this laboratory exercise, you need to get access to the machines directly to work with them. Docker allows us to execute commands inside containers using the `exec` command:

```
$ docker exec -it <container-id> <command>
```

where `<container-id>` refers to the name or ID of the container and `<command>` to the command to be executed. You can only execute commands that are also installed within the container. For example, since Firefox is not installed inside of the containers, the command `docker exec -it <container-id> firefox` will fail. To find out the container name or ID, you can use:

```
$ docker ps
```

which provides a list of all docker containers that are currently *running* on your host machine.

In order not to execute single commands inside the container all the time, it is best practice to launch a shell session. Since all containers used in this lab come with a pre-installed `bash` shell, you can run the command:

```
$ docker exec -it <container-id> bash
```

Thus, you can work with the container as with any other (non-graphical) UNIX/Linux installation.

## A short description of the devices.

Each of the involved machines is provided by means of a basic Linux container. By default, the routing capability is enabled by the `ipv4_forward` option, set in both the router and the malicious routing host from within the `docker-compose.yml` file. You are free to adjust this setting if you feel it is required for your solutions. For the other containers, IP forwarding is not necessarily enabled, but can be enabled by adjusting the `docker-compose.yml` file.

Beside the basic tools like `ping` [7], `arp` [1], and `ip` [4], each system has *MyTraceroute* (mtr) [5] pre-installed. You can directly call it from the terminal of a given machine using

```
$ mtr -n <IP>
```

where `<IP>` have to be replaced with the IP address of interest. On each machine, you can use *netcat* [6], which is available by the command `nc`. The machine of the attacker and the malicious router additionally have Python3 [9] and the Scypy library [11] installed.

It is important to note that docker containers may lose the data you save on them if you stop and delete them. Therefore, the machines of the attacker are configured by an additional *host share*, named `volumes`. You may want to use this share to permanently store files required for the attack and to share them with your containers. The storage is automatically mounted to `/volumes` on the attacker and the malicious-router container. This means that files that you move into your local `volumes` folder will show up within the containers at this location.

# Tasks

## Task 1: Setup the Network

Before you can mount the actual attack, you will have to setup the network and ensure that it is working as expected. Configure and start the network devices as described in the preparation section. Afterwards, check that each machine is available in terms of network connectivity $(i)$ within the sub-net and $(ii)$ from one network to the other one. You may need to use the tools `ping`, `traceroute`, and/or `mtr` to achieve this. Be prepared to show the correct functioning of your laboratory setup during the lab defense.

## Task 2: Understanding ICMP Redirect Messages

Once the network is set up, you should check whether the victim machine supports ICMP redirect messages. Many modern operating systems have already disabled this feature if it is not required, which is commonly the case. For this purpose, you are requested to write a short program that crafts and sends spoofed ICMP messages. Think about a way how you can prove that your script is correct, i.e., that the ICMP redirection messages are accepted by the victim and the redirection takes place? Analyze the following scenarios using your developed tool:

1. The attacker, its malicious router, and the victim are in the same sub-network.
2. The attacker and the victim are in the same network. However, the attacker tries to redirect the victim's communication over a router placed on a different sub-network.
3. The attacker and the victim are in the same network. However, the attacker tries to redirect the victim's communication to a non-existing router.
4. The attacker is in a different sub-network than his victim.

Elaborate on each of the scenarios above. Verify whether the ICMP redirect attack becomes effective or it is not possible. Provide an appropriate explanation for your observations in either case.

**Hints:**

1. What are the differences between the routing table and the routing cache? What is the purpose of the cache and by which operations does it get affected? Basic knowledge of the routing cache management is also required.
2. While developing your attack, you might need to take a look into the `scapy` library. The library is already installed on the attacker machine, as well as on the malicious router.
3. When mounting your attack, make sure that, from a technical point of view, all provided machines (containers) are equal.
4. The RFC 1122 [10] (Sec 3.3.1 and Appendix A) is a helpful source to understand why sometimes the attack may be successful and other times it renders useless.

## Task 3: Establish Yourself as the Man in the Middle (Eavesdropper)

Use your knowledge about ICMP redirect messages gained in the previous task to establish the malicious router as the man in the middle to eavesdrop on communication between the victim and their destination. Once you achieved this goal, use `tcpdump` [12] to eavesdrop on the communication.

**Hints**:

1. While mounting and/or debugging the attack, you should make sure that there is ongoing traffic between the victim and the destination. To do so, you should start a netcat server on each of the endpoints.
2. With the right kernel settings and network settings configured, this task does not require you to write any additional pieces of code.

## Task 4: Modifying Messages

Although the basic attack considered until here allows to break confidentiality, the integrity remains intact during basic eavesdropping. In this task, we want you to extend your basic man-in-the-middle scenario by modifying the traffic routed over your malicious router. To this end, you should develop a basic program that allows you to intercept and modify the content sent over the malicious router. The final script should provide a command-line interface with the following three parameters:

| Command | Description |
|---|---|
| `-s/--src` | The source address of the data-stream to modify (*mandatory*) |
| `-f/--find` | The pattern to replace |
| `-r/--replace` | String used to replace the found pattern |

It is recommended that you take a look at the Python `argparse` library for this. For example, the call to your program may look like:

```
$ modify.py --src 10.9.0.5 -f "I failed!" -r "I passed"
```

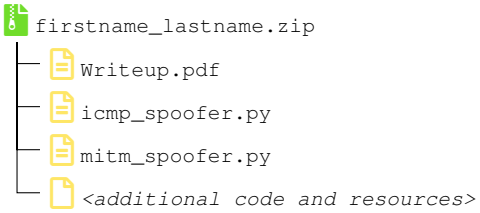
**Hints**:

1. Consider the cases where the pattern to replace and the new content have different length. Think about a reasonable condition to handle this.
2. Think about why there is no `--dst` parameter required by the script.
3. Check the settings of your operating system. Normally, routers are not designed to provide traffic streams to upper layers. Which settings may you need to change and how do these changes effect other clients who are using the malicious router as well?
4. The `--src` parameter is intended to be a network address. However, you are free to decide whether you use IPv4 addresses, IPv6 addresses or even MAC addresses. In any case you need to justify your decision.
5. To avoid any network delay, which may be noticed by the communicating peers, your program should run as fast as possible. Think carefully about the options available to improve the performance of your attack.

## Preparation of Your Submission

Your submission for this lab should contain any and all source code you have written and files you have generated, along with a detailed writeup about the tasks you solved. Your writeup should be given in an appropriate format, containing detailed descriptions of how you solved each task, what theoretical insights you gained, what observations you have made as well as images / screenshots / listings to aid the understanding of your approach. Your writeup should be submitted in PDF format.

Please submit your solutions in a packed zip file named after your full name, with the following structure and minimum required contents. In particular, ensure you submit the programs you implemented to spoof ICMP packets (Task 2) and to perform the spoofing man in the middle attack (Task 4).




## Preparation for the Q&A Session

Prepare yourself for a Q&A session of up to 40 minutes. During the Q&A session, you should be prepared to *explain and demonstrate* how you solved the tasks. Take special care to explain how your solution works and why you took the approach you took. Note that simply presenting a piece of code without any explanation will not yield any points! Last but not least, you should be ready to answer spontaneous (theoretical) questions asked by the reviewer in relation to the topics of this lab.

# References


[1] *arp(8) Linux man page*. https://man7.org/linux/man-pages/man8/arp.8.html (cit. on p. 4).

[2] *Docker*. https://www.docker.com/ (cit. on p. 3).

[3] *Docker Compose*. https://docs.docker.com/compose/ (cit. on p. 3).

[4] *ip(8) - Linux man page*. https://linux.die.net/man/8/ip (cit. on p. 4).

[5] *My traceroute*. https://www.bitwizard.nl/mtr/ (cit. on p. 4).

[6] *nc(1) - Linux man page*. https://linux.die.net/man/1/nc (cit. on p. 4).

[7] *ping(8) Linux man page*. https://linux.die.net/man/8/ping (cit. on p. 4).

[8] *pip - package installer for python*. https://pypi.org/project/pip/.

[9] *Python*. https://www.python.org/ (cit. on p. 4).

[10] *Requirements for Internet Hosts - Communication Layers*. https://datatracker.ietf.org/doc/html/rfc1122 (cit. on p. 7).

[11] *Scapy: Packet crafting for Python2 and Python3*. https://scapy.net/ (cit. on p. 4).

[12] *TCPdump*. https://www.tcpdump.org/manpages/tcpdump.1.html (cit. on p. 8).

[13] *traceroute(8) - Linux man page*. https://linux.die.net/man/8/traceroute.