MAMI summer school:
Path tracing with tracebox

K. Edeline J. Iurman
korian.edeline@uliege.be justin.iurman@uliege.be

Université de Liège

16/04/18

1 Overview

Middleboxes such as firewalls, NAT, proxies, or Deep Packet Inspection play an increasingly important role in various types of IP networks, including enterprise and cellular networks. Recent studies have shed the light on their impact on real traffic and the complexity of managing them. Network operators and researchers have few tools to understand the impact of those boxes on any path. To address this problem, researchers from UCL and ULiège, proposed a new tool called `tracebox` [1], which is an extension to the widely used `traceroute` [3] tool that is capable of detecting middlebox interference and of locating them on a given path.

In this exercise session, you will be confronted with different network topologies. Each of the topologies contains different middleboxes which will affect the network traffic in different ways. Your goal is to describe the topology, to detect and localize the middleboxes, and to characterize their action. To do so, you will be invited to code a *simplified version* of `tracebox` in Python using Scapy.

2 tracebox

`tracebox` mechanism is illustrated in Fig. 1. It relies on RFC1812 and RFC792 stating that ICMP `time-exceeded` message should quote the IP header of the original packet and respectively the complete payload or the first 64 bits. `tracebox` uses the same incremental approach as `traceroute`, i.e., it sends packets with different IP, ICMP, UDP, or TCP fields and options with increasing TTL values. By comparing the quoted packet to the one sent, one can highlight the modifications applied by a middlebox and the initial TTL value allows us to localize the two or more hops between which the change took place.

In Fig. 1, the source sends a TCP packet, `a`, with an initial TTL value of 3. The middlebox, located between hop 2 and hop 3, modifies (for instance) the TCP Initial Sequence Number (ISN) and forwards the rewritten packet, `b` to the next hop. When the router located at hop 3 receives the packet, the TTL has expired and it sends back to the packet source an ICMP time-exceeded packet `c` containing packet `b` as a payload. When the source receives it, it is able to compare packet `a` and the payload of packet `c` to detect any changes and the initial TTL value, i.e., 3, allows tracebox to bound the middlebox location. The router that reveals packet modifications in the ICMP time-exceeded message is called the informant router.

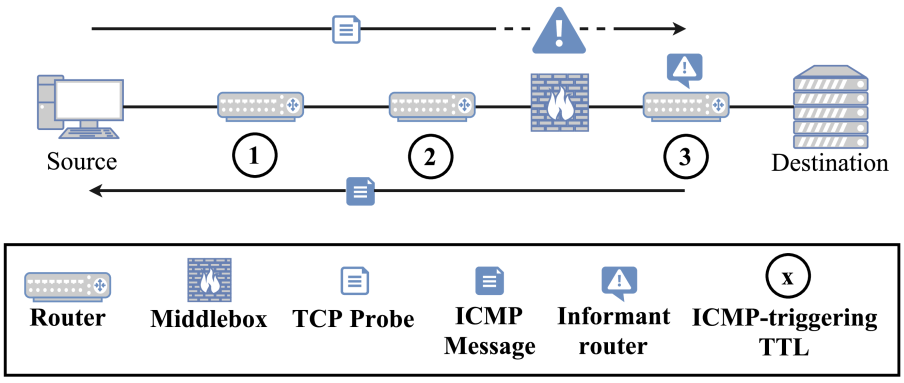

Figure 1: tracebox [2]

3 Practical Details

3.1 Requirements

- VirtualBox[1] installed on your machine,
- Vagrant[2] installed on your machine,
- An ability to code in Python 3.x.

3.2 Virtual machine

The installation, boot and execution of the VM is entirely handled by *Vagrant*, through *VirtualBox*. Many useful *Vagrant* commands exist, such as *status*, *up*, *ssh*, *destroy*, etc. You can find more information about them in the *Vagrant* CLI documentation[3].

To start the VM, go to the *vagrant* folder and execute this command:

```
vagrant up
```

Once the VM is running, you can login with ssh, thanks to this command:

```
vagrant ssh
```

From now on, any command you type will be executed on the VM. To close the ssh connection and get back to your host machine (in */vagrant/* folder), enter *exit*. At the end of the lab, you can delete the VM by executing this command:

```
vagrant destroy -f
```

3.3 Materials

All the material needed for the laboratory is already installed and configured on the virtual machine (see section 3.2). Once logged in, you'll see a simple traceroute core written in Python with Scapy that you can use as a base for a tracebox (*∼/simple_tracebox.py*). For the sake of simplicity, any modification applied to this file in the VM will also be applied to your original file (host machine), and vice-versa.

You'll also find an executable called *setup_topology*. It takes either 1, 2 or 3 as an argument (there are 3 different topologies in total, from simple to more complex) and is used to switch to any topology you want to use for your tests. As an example, if you enter the command *./setup_topology 2* then you will switch to topology 2. For each topology, the destination address is **10.0.0.10**.

[1]https://www.virtualbox.org/wiki/Downloads
[2]https://www.vagrantup.com/downloads.html
[3]https://www.vagrantup.com/docs/cli/

4 Questions

For each of the available topologies, we ask you to use to observe the path towards destination **10.0.0.10** and to:

1. Identify links and hops. Try to draw the network topologies.
2. Identify and characterize middleboxes (load balancer, tcp enhancing proxy, firewall ?).
3. Find out if they may cause any problem with TCP, UDP or ICMP.

To achieve this, we recommend to use regular network measurement tools (ping, variants of traceroute) from the utility software `scamper` (`$man scamper`), and to use and extend the provided Python script.

5 Documentation

- *Python*: `https://docs.python.org/3/`
- *Scapy*: `http://scapy.readthedocs.io/en/latest/`
- *dpkt*: `http://www.commercialventvac.com/dpkt.html`

6 Feedback

For those who whish to get feedback about your work, send a PDF file with your name, your answers and your Python code to korian.edeline@uliege.be.

References

[1] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. Revealing middlebox interference with tracebox. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 1–8. ACM, 2013.

[2] K. Edeline and B. Donnet. A first look at the prevalence and persistence of middleboxes in the wild. In *Teletraffic Congress (ITC 29), 2017 29th International*, volume 1, pages 161–168. IEEE, 2017.

[3] V. Jacobson. Pathchar: A tool to infer characteristics of internet paths, 1997.