MPTCP Security Analysis

Polytechnic of Turin - Stockholm's KTH

Fabrizio Demaria

Fall 2015

Abstract

Abstract goes here

Acknowledgements

I want to thank...

Contents

Chapter 1

Multipath TCP introduction

1.1 Overview

1.2 Benefits

Chapter 2

Deployment analysis

2.1 Performance, security and reliability

2.2 Middleboxes compatibility

2.3 Supported platforms

2.4 Deployment status

Chapter 3

The protocol design

3.1 Transmission Control Protocol (TCP)

3.2 Extending TCP to MultiPath

3.2.1 Control plane

3.2.2 Data plane

Chapter 4

Security

4.1 Networking security overview

4.2 MPTCP residual threats

4.3 The ADD_ADDR attack

4.3.1 Concept

4.3.2 Execution

4.3.3 Requirements

Chapter 5

Executing ADD_ADDR attack

5.1 User Mode Linux

In order to achieve a reliable reproduction of a real world scenario, the simulation included the setup of two User Mode Linux (UML) virtual machines running a Linux kernel with enabled support for MPTCP. These two machines will act as client and server, carrying on a MPTCP connection that will act upon. Using UML to proceed with the experiments seemed to be the best way to go, allowing very fast setup and boot-up time, with good emulation of real machines and giving the possibility to work a on a single hosting machine with no risk to damage or crash its underlying kernel via our tests.

A very good resource in terms of tool, configuration files and actual kernel images to be used out of the box is the official mptcp website: *http://www.multipath-tcp.org/*. In particular, the website offers a python script that would download all the necessary files to run the two virtual machines. As for this specific purpose of verifying the ADD_ADDR attack feasibility we won't need to act on the kernel source code, and we are able to use the above mentioned components out of the box. Actually, this is the best way to go, performing the attack on the official distribution currently offered by the official resources.

Here it follows the content of the *client.sh* script downloaded through the setup.py script retrieved from the official Web page (similar shell script can be found for the server counterpart, including a single interface configuration according to testing network definition):

```
1 #!/bin/bash
2
3 USER=`whoami`
```

```
4
5  sudo tunctl -u $USER -t tap0
6  sudo tunctl -u $USER -t tap1
7
8  sudo ifconfig tap0 10.1.1.1 netmask 255.255.255.0 up
9  sudo ifconfig tap1 10.1.2.1 netmask 255.255.255.0 up
10
11 sudo sysctl net.ipv4.ip_forward=1
12 sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/8 ! -d 10.0.0.0/8 -j MASQUERADE
13
14 sudo chmod 666 /dev/net/tun
15
16 ./vmlinux ubda=fs_client mem=256M umid=umlA eth0=tuntap,tap0 eth1=tuntap,tap1
17
18 sudo tunctl -d tap0
19 sudo tunctl -d tap1
20
21 sudo iptables -t nat -D POSTROUTING -s 10.0.0.0/8 ! -d 10.0.0.0/8 -j MASQUERADE
```

Listing 5.1: *client.sh*

With this code, it is possible to run the two UML machines and use the local *tap* interfaces to sniff and inject packets (acting, in this specific case, as a physical man in the middle).

The resulting network scenario will result as follows:

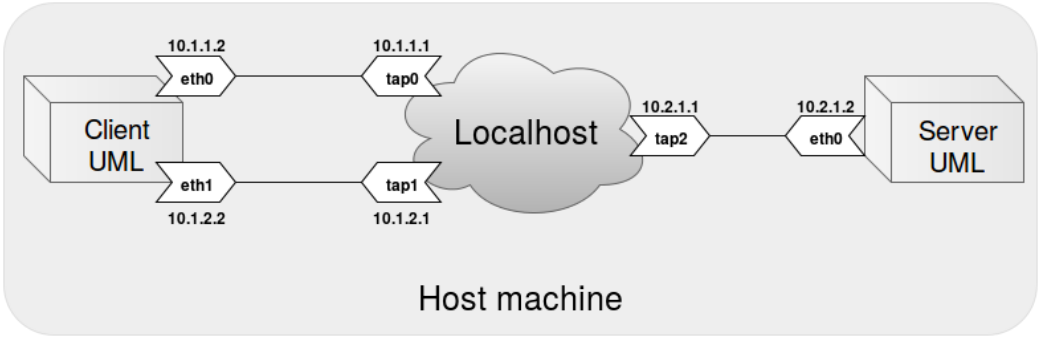

Figure 5.1: Network scenario

5.2 Scapy tool

5.3 Results analysis

5.4 Limitations and future work

Chapter 6

Implementing MPTCP v1.0

6.1 The Linux Kernel implementation

6.2 ADD_ADDR2 and MPTCP version 1 principles

6.2.1 ADD_ADDR2 implementation

6.2.2 Retro-compatibility with MPTCP version 0

6.2.3 Port advertisement

6.2.4 IPv6 considerations

6.3 Final patch analysis

6.4 ADD_ADDR2 further considerations

Chapter 7

Conclusions