Lab Report DDOS

Dean Carmi

Table of Contents

1. Setting up the machines

In order to do recreate the conditions of the experiment I included the instructions for each machine.

1.1. Victim

The victim should include only a running *apache server*, listening on port 80 (without firewall)

1.2. Monitor

In directory `monitor_files` we have all files related to the monitor. Using the `makefile` in that directory, steps to setup the system:

`make hping`

install nessecary package.

`make python_scapy`

installing everything related to the right version of python.

1.3. Attacker

In directory `attacker_files` we have all files related to the attacker. Using the `makefile` in that directory, steps to setup the system:

`make python_setup`

install anything related to python.

2. Creating the attack

2.1. Using C

Instructions:

Victim

Run the apache server.

Attacker

In directory `attacker_files` run:

- Compile - `make c_ddos`
- Run - `sudo ./syn_flood 10.0.2.12 80` (You can replace the IP for any IP that you attack)

Monitor

In directory `monitor_files` run:

- Ping - `make start_c` . **Note: You need to stop manually this process**

2.1.1. After the attack

Attacker side

A new file named `syns_results_c` will be created containing the required info.

Monitor side

A new file named `pings_results_c_raw.txt` will be created and we need to parse it. Run `python3.8 parse_hping.py pings_results_c_raw.txt`. The script will ask you in which language the code is, you should reply **c** for C and **p** for Python. **Note: the RTT is in milliseconds**.

2.2. Using Python

Instructions:

Victim

Run the apache server.

Attacker

In directory `attacker_files` run:

- Run - `sudo python3.8 ddos.py 10.0.2.12 80 1000000` (You can replace the IP for any IP that you want to attack, the last nubmer is the amount of packet, a million)

Monitor

In directory $monitor_{files}$ run:

- Ping - `make start_py` . **Note: You need to stop manually this process**

2.2.1. After the attack

Attacker side

A new file named `syns_results_p` will be created containing the required info.

Monitor side

A new file named `pings_results_p_raw.txt` will be created and we need to parse it. Run `python3.8 parse_hping.py pings_results_p_raw.txt`. The script will ask you in which language the code is, you should reply **c** for C and **p** for Python. **Note: the RTT is in milliseconds**

3. Report

3.1. Regarding syns results

C code

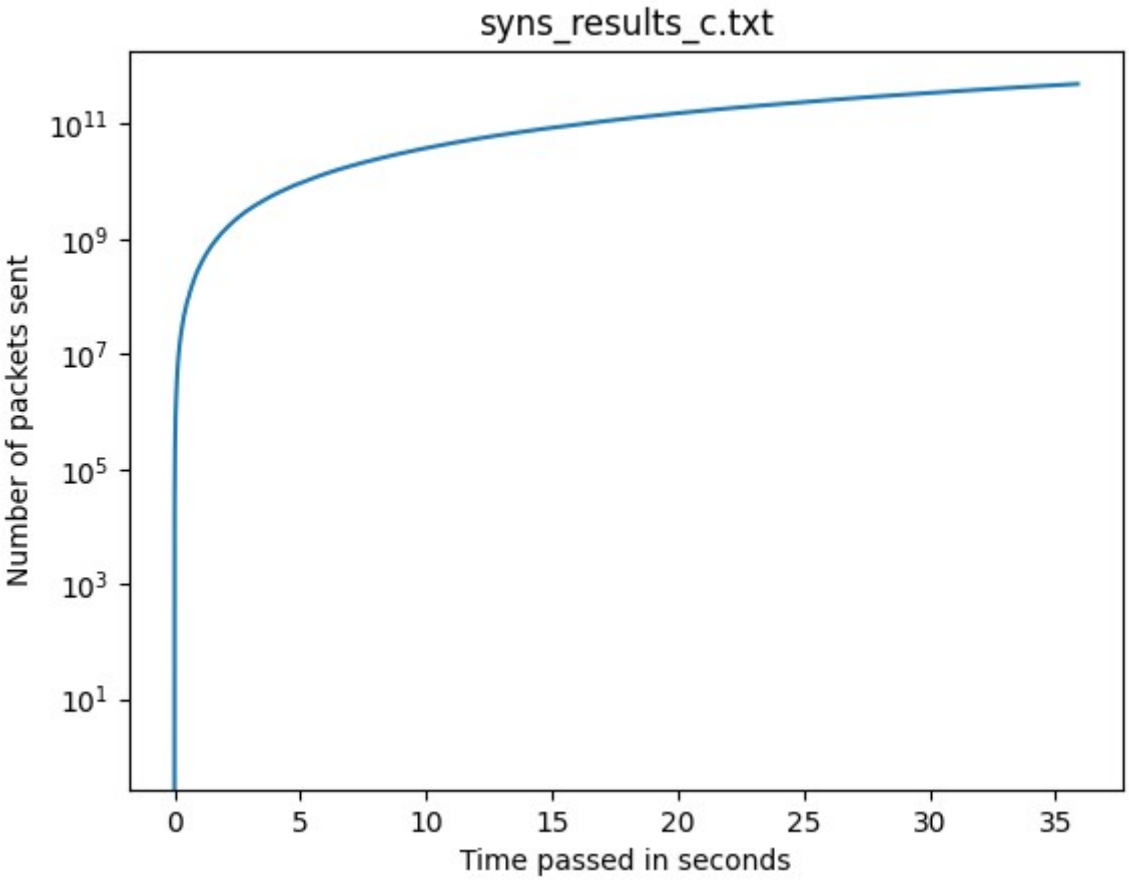

Figure 3.1.1: Results of syn packet sending with C

- Average time for a packet - 0.000036 seconds.
- Total running time - 52.894455 seconds.
- Standard deviation - 0.0000315946680146141

Python code

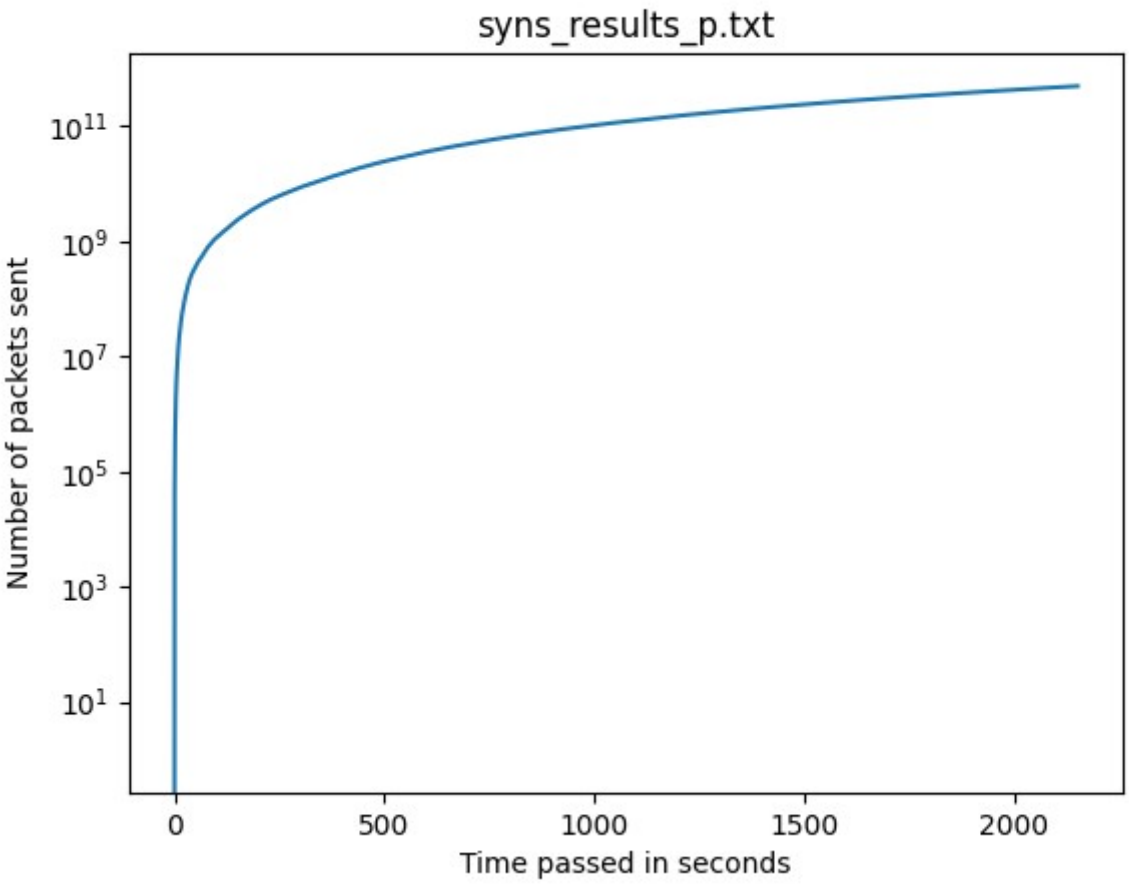

Figure 3.1.2: Results of syn packet sending with Python

Average time for a packet

0.002151517 seconds.

Total running time

2515.619506395 seconds.

Standard deviation

0.0004762834996620782

3.1.1. Explaination

We can observe two differences:

- Attack Time - C took 52 seconds vs Python that took 2515 seconds. C was faster by a factor of 48.
- Average time for a packet - C took 0.000036 seconds vs Python that took 0.00215 seconds. Python was slower by a factor of 59.

We know that because Python is an interpreted language, it amplifies the number of actual CPU instructions required to perform the code given (compared to code in C). In addition, In my case the

Python code didn’t run as machine code (like the C code), but it ran in a virtual machine (the bytecode interpreter).

3.2. Regarding the pings results

C code

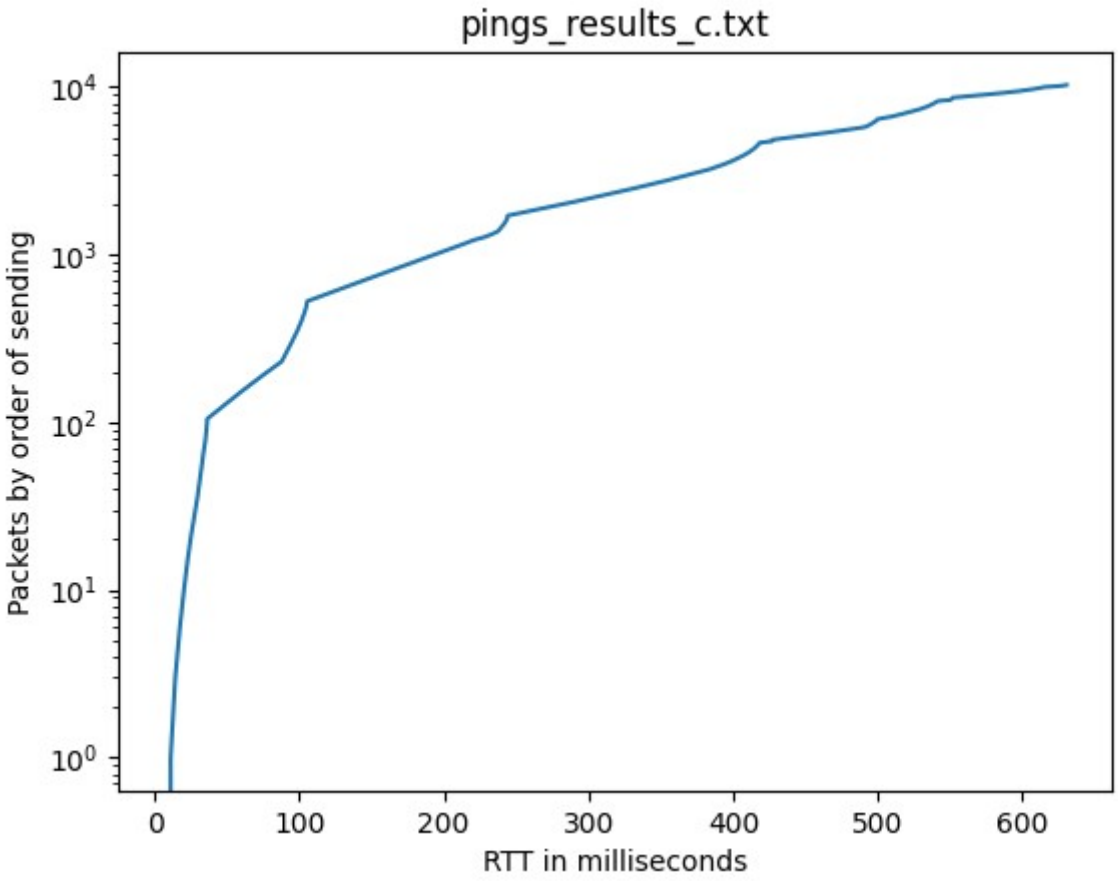

Figure 3.2.1: Results of the pings sending when the C attack took place

Average RTT	4.378472 milliseconds
Standart deviation	2.6498208

Python code

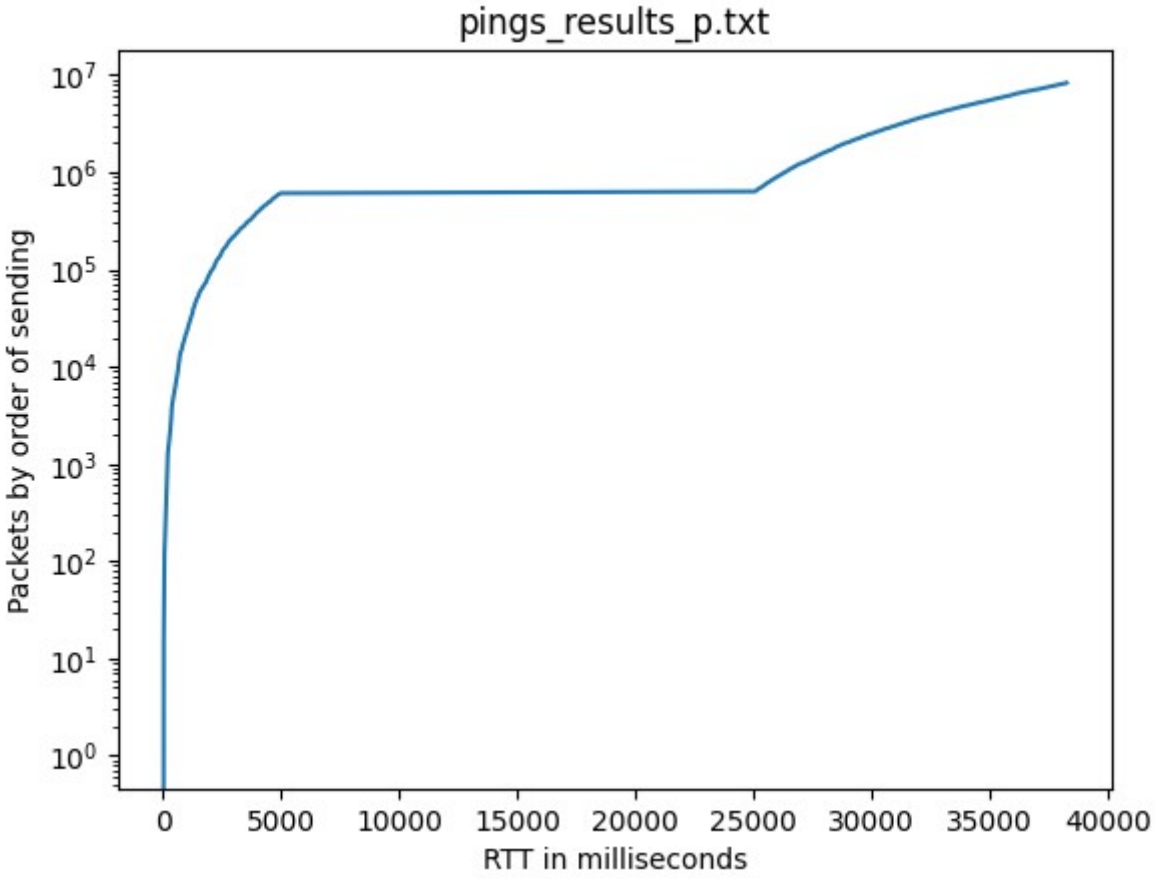

Figure 3.2.2: Results of the pings sending when the Python attack took place

Average RTT	9.39337423 milliseconds
Standard deviation	69.80985083

3.2.1. Explaination

- Average RTT