Enabler Manual
Malicious Traffic Generator (MTG)

Project name	5G Enablers for Network and System Security and Resilience
Short name	5G-ENSURE
Grant agreement	671562
Call	H2020-ICT-2014-2
Authors	NIXU: Tommi Pernilä, Markku Suominen, Aleksi Dahl, Anita Kasari

Foreword

5G-ENSURE belongs to the first group of EU-funded projects which collaboratively develop 5G under the umbrella of the 5G Infrastructure Public Private Partnership (5G PPP) in the Horizon 2020 Programme. The overall goal of 5G-ENSURE is to deliver strategic impact across technology and business enablement, standardisation and vision for a secure, resilient and viable 5G network. The project covers research & innovation - from technical solutions (5G security architecture and testbed with 5G security enablers) to market validation and stakeholders engagement - spanning various application domains.

This manual is part of the project's deliverable D3.8. It describes how one of the security enablers that are developed within the work package WP3 of the 5G-ENSURE project is installed and administrated. Furthermore, this manual contains a user guide of the respective security enabler.

Disclaimer

Copyright notice

Contents

1 Introduction

In this chapter, we present the "Malicious Traffic Generator" enabler documentation. This enabler is one of Release 2 enablers. It mainly aims to generate a malicious traffic to the 5G network to cause misbehaviour and Denial of Service on the network functions. The "Malicious Traffic Generator" will produce both predefined DoS-attacks and random (fuzzed) traffic to be send to the network.

2 Installation and Administration Guide

This section describes how the enabler is installed, configured and how to use it.

2.1 System Requirements

The enabler is to be executed in a Linux host. The enabler expects Python 2.7, Scapy (Python library) and Radamsa to be available. (See the instructions in chapter 2.3.2). The enabler must be executed with root-privileges on the host machine. The minimum system requirements are listed below.

OS: Debian 8 (should work on any Debian based distribution)
Minimum requirements for CPU: single x86 core
Minimum requirements for disk space: 512MB
Minimum requirements for RAM: 1GB
Minimum requirements for installed applications: Python 2.7, Scapy 2.3.3 Python library, Radamsa 0.6a
Minimum requirement: Installation needs an active internet connection

2.2 Enabler Configuration

This enabler is configured by command line switches, see section 3.1. Modifications can also be done alternatively directly to source code if wanted.

2.3 Enabler Installation

2.3.1 System Installation

Follow normal installation instructions of Debian.

2.3.2 Dependencies Installation

If Python is not included on the used Linux distribution it can be installed by issuing following command:

```
$ apt-get install python
```

Scapy can be installed from Github or by using Debian apt tools. Installing from Github by using following command:

```
$ git clone https://github.com/secdev/scapy
```

and from Debian repository:

```
$ apt-get install python-scapy
```

Radamsa is installed from Github with following command:

```
$ git clone https://github.com/aoh/radamsa.git && cd radamsa && make && sudo make install
```

2.4 Troubleshooting

Make sure that internet connection is working and make sure that the user is executing the commands with root privileges.

3 User and Programmer Guide

This section describes how to use and program the enabler.

3.1 User Guide

After necessary packages are installed the enabler is called from command line by using command:

```
$ python maliciousTrafficGenerator.py
```

The Python script has a help file included which can be shown by using –h switch as shown in Figure 1.

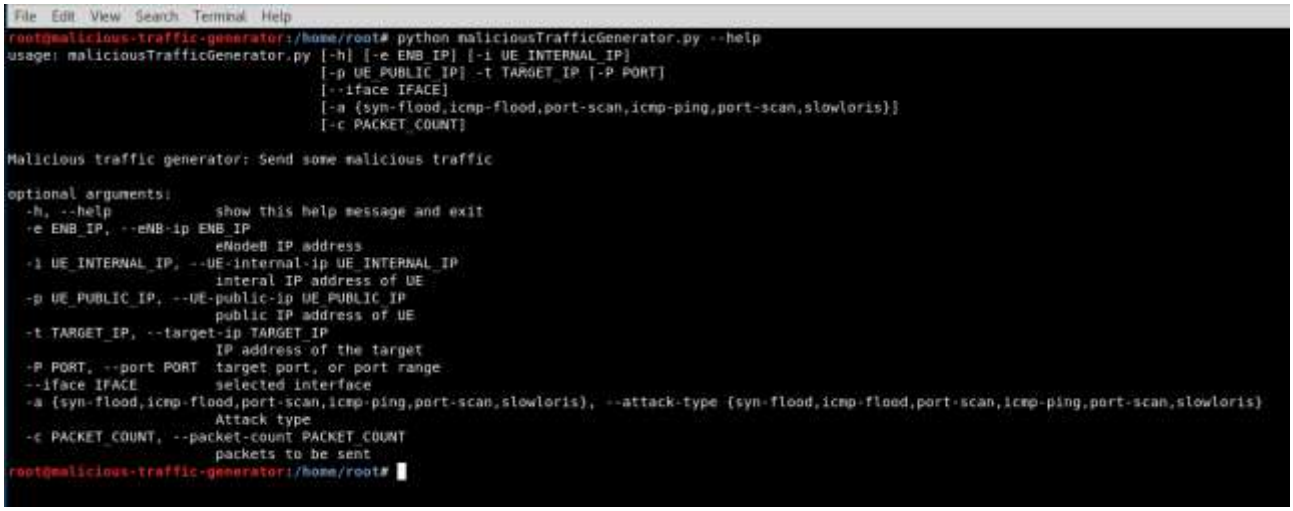

Figure 1 Help file

User must provide values for the parameters and choose one of the attack types to launch enabler. For example, the port-scan attack can be launched by issuing following command:

```
$ python maliciousTrafficGenerator.py --eNB-ip 127.0.0.1 --UE-internal-ip 127.0.0.50 --UE-public-ip 10.0.2.15 --target-ip 192.168.56.101 --port 1-100 --iface lo -a port-scan -c 1
```

Explanation of above command: The malicious traffic generator will generate TCP port scan on ports 1-100 of target host (192.168.56.101) via GTP tunnel. Other pre-defined attacks can be used by changing the –attack-type switch value. Figure 2 shows the printed output when the port-scan attack is executed.Figure 2 Port scan example

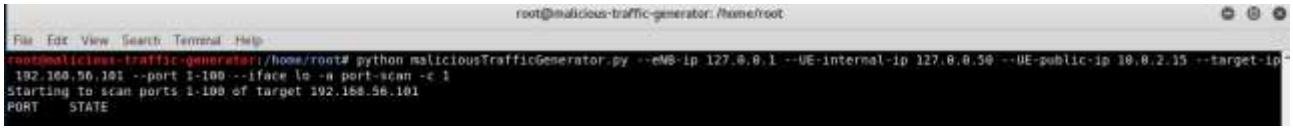

Figure 2 Port scan example

Current version of the enabler has the following attack types included:

- **syn-flood**: This attack sends TCP packets on selected ports with SYN flag set on the header.
- **icmp-flood:** This attack sends icmp packets with type 3 –code a.k.a. "BlackNurse" attack.
- **icmp-ping:** This attack sends icmp ping packets with huge payload in fragmented IP packets.
- **port-scan:** This attack scans for open ports on the target with given port or port range.
- **slowloris:** This attack sends uncompleted HTTP GET request to target.

3.2 Using the fuzzing engine

This enabler uses Radamsa as fuzzing engine. Radamsa works by reading sample files of valid data and generates randomized output based on them. Current version of the enabler supports fuzzing with Radamsa on two different modes, HTTP-request and GTP-tunnel. Various templates for malicious patterns can be made and used with fuzzing engine. Using the pattern library is described in chapter 3.3.

3.3 Using the malicious pattern library

Pattern libraries are used to feed sample data to Radamsa, which then generates different outputs based on the sample data. Current version of the enabler delivers templates for HTTP-request (e.g. in file get.http). Following lines are the context of get.http-file:

```
GET / HTTP/1.1
Host: 192.168.56.101
Connection: keep-Alive
Keep-Alive: 300
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Malicious Fuzzer
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
----------------------------------
```

Note: Ending line marked with "-"-characters is used to indicate the end of the HTTP request.

Fuzzing feature is provided in maliciousFuzzer.py –file, which reads data from standard input and sends it to the target host with TCP-datagrams inside of the GTP-tunnel (when http mode is used).

Example command:

```
$ radamsa -n inf get.http | ./maliciousFuzzer.py --target 192.168.56.101
--mode http
```

3.4 Programmer Guide

Enabler has been written with Python programming language and its source code is included in the delivered package. This means that it is very easy to customize and modify it if needed.

The enabler has been designed to be modular. One example of this is the template structure, which makes it very easy to add additional protocols.

4 Unit tests

4.1 Unit Test 1

This unit test 1 verifies that Python, Scapy and Radamsa are installed correctly.

First test that network connectivity is available (ping the target).

One can check that dependencies are installed correctly by running following commands.

1. Python installation

```
$ python -V

Python 2.7.12
```

2. Radamsa installation

```
$ radamsa -V

Radamsa 0.6a
```

3. Scapy installation:

```
$ scapy -h

Usage: scapy.py [-s sessionfile] [-c new_startup_file] [-p
new_prestart_file] [-C] [-P]

    -C: do not read startup file

    -P: do not read pre-startup file
```

4. MTG:

```
$ python maliciousTrafficGenerator.py –h

usage: maliciousTrafficGenerator.py [-h] [-e ENB_IP] [-i UE_INTERNAL_IP]
                                    [-p UE_PUBLIC_IP] -t TARGET_IP [-P PORT]
                                    [--iface IFACE]
                                    [-a {syn-flood,icmp-flood,port-scan,icmp-ping,port-scan,slowloris}]
                                    [-c PACKET_COUNT]

Malicious traffic generator: Send some malicious traffic
optional arguments:
  -h, --help            show this help message and exit
  -e ENB_IP, --eNB-ip ENB_IP
                        eNodeB IP address
  -i UE_INTERNAL_IP, --UE-internal-ip UE_INTERNAL_IP
                        interal IP address of UE
  -p UE_PUBLIC_IP, --UE-public-ip UE_PUBLIC_IP
```

```
                public IP address of UE
  -t TARGET_IP, --target-ip TARGET_IP
                IP address of the target
  -P PORT, --port PORT  target port, or port range
  --iface IFACE         selected interface
  -a {syn-flood,icmp-flood,port-scan,icmp-ping,port-scan,slowloris}, --attack-type {syn-flood,icmp-
flood,port-scan,icmp-ping,port-scan,slowloris}
                Attack type
  -c PACKET_COUNT, --packet-count PACKET_COUNT
                packets to be sent
```