Deliverable D3.4
5G-PPP Security Enablers Documentation (v1.0)
Generic Collector Interface Enabler

Project name	5G Enablers for Network and System Security and Resilience	
Short name	5G-ENSURE	
Grant agreement	671562	
Call	H2020-ICT-2014-2	
Delivery date	30.09.2016	
Dissemination Level:	Public	
Lead beneficiary	NEC	Felix Klaedtke, felix.klaedtke@neclab.eu
Authors	Orange: Jean-Philippe Warry, Ghada Arfaoui	

Document Version	Date	Change(s)	Author(s)
0.1	28.06.2016	Created template	Felix Klaedtke

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.

Disclaimer

The information in this document is provided 'as is', and no guarantee or warranty is given that the information is fit for any particular purpose.

The EC flag in this deliverable is owned by the European Commission and the 5G PPP logo is owned by the 5G PPP initiative. The use of the flag and the 5G PPP logo reflects that 5G-ENSURE receives funding from the European Commission, integrated in its 5G PPP initiative. Apart from this, the European Commission or the 5G PPP initiative have no responsibility for the content.

Copyright notice

© 2015-2017 5G-ENSURE Consortium

Contents

1 Introduction

In this chapter, we present the "Generic Collector interface" enabler documentation. This enabler mainly aims at collecting data from the 5G network elements / components. The "Generic Collector Interface" will provide to authorized parties / actors large amount of data namely: logs, events and incidents related to virtualization, identity management, communication protocols / layers / stacks and some specific privileges escalation. This enabler leverages also the implementation of efficient FastData [1] inside 5G Networks, in order to detect as soon as possible security and efficiency issues of the network.

At the writing time of this document, the Generic Collector interface is still under research. In the scope of R1, we plan to develop some features (namely the report exchange), of the open specifications described in D3.2 [2], as a proof of concept. Our proof of concept mainly includes three components: a monitoring server that centralizes the collected data of a given network, the monitoring client(s) that send(s) collected data to the monitoring server and the monitoring service that uses the collected data.

2 Installation and Administration Guide

This section describes how the enabler is installed, configured, and administrated.

2.1 System Requirements

In a nutshell, we recall a simplified functional overview of the "Generic Collector interface". The Client engine sends XML reports to the server engine. Depending on the security policies, the server engine transmits these reports to the designated service provider.

As shown in Figure 1 , the "Generic Collector Interface" consists of three main software blocs namely the client engine software (cf. Open specification in D3.2), the server engine software (Open specification D3.2) and the service software. The first software, so called "monitoringClient.py", should be installed on all the network components. This software should run with the root privileges in order to be able later to collect necessary information. The second software, so called "monitoringServer.py", should run in an external server (i.e., the collect server). The third software, so called "monitoringService.py" should run on the service provider side.

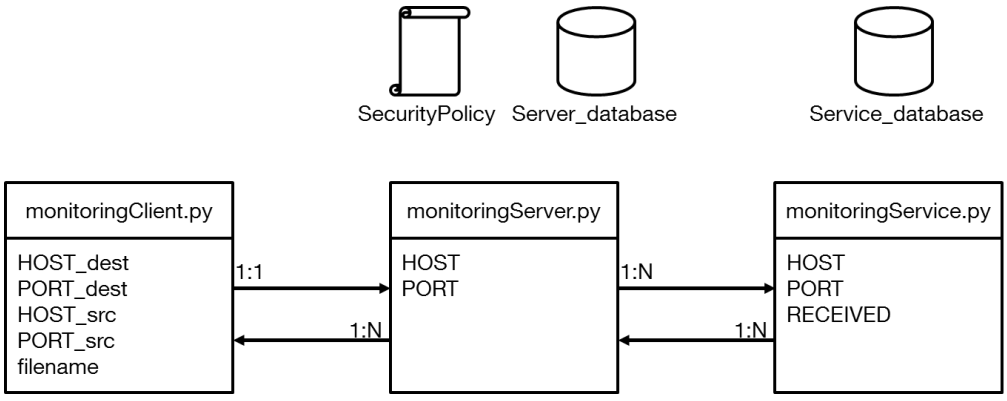

Figure 1: Generic Collector Interface diagram

2.2 Enabler Configuration

The “Generic Collector Interface” has a so called “securityPolicy” file (cf. Figure 1) and two database folders, i.e., “Server_database” and “Service_database” (cf. Figure 1). The “securityPolicy” file should be accessible only in “read” mode for this enabler, namely “monitoringServer.py”. This file contains the network configuration, .i.e., “who can get which information”. For instance, this file can contain information like: data originated from the network components *X* must be transmitted only to the service *A.* This file is organized in four fields per line, as follows: <source IP address> <source port> <target IP address> <target port>. The “Server_database” folder should be available in “read” and “write” modes only for the server engine software. This folder will temporary contain the collected data. Once the data is correctly transmitted to the designated service, it is deleted from this folder. The “Service_database” folder should be available in “read” and “write” modes only for the service software.

The current main configurations of this enabler are

- The setup of the IP addresses and ports for the different running software (Figure 2, Figure 3, Figure 4).
- The setup of the storage folders and its access rights
- The setup of the security policy file and its access rights

For “monitoringClient.py”, it is required to precise the available IP address and port of the host running “monitoringServer.py” (lines 5 and 6 in Figure 2). Lines 8 and 9 specify the IP address and port from with data will be sent. HOST_src = '' means that ‘monitoringClient.py” will send data on the available interface. To force using a given IP address, replace '' by the given IP address.

Figure 2: Configuration of "monitoringClient.py"

For “monitoringServer.py”, it also required to specify the IP address and port where to wait for inputs from the client engines (lines 6 and 7 in Figure 3). HOST = '' means that “monitoringServer.py” will wait for input on all the available interfaces. As previously mentioned, to force “monitoringServer.py” to listen only to a given IP address, replace '' by the given IP address.

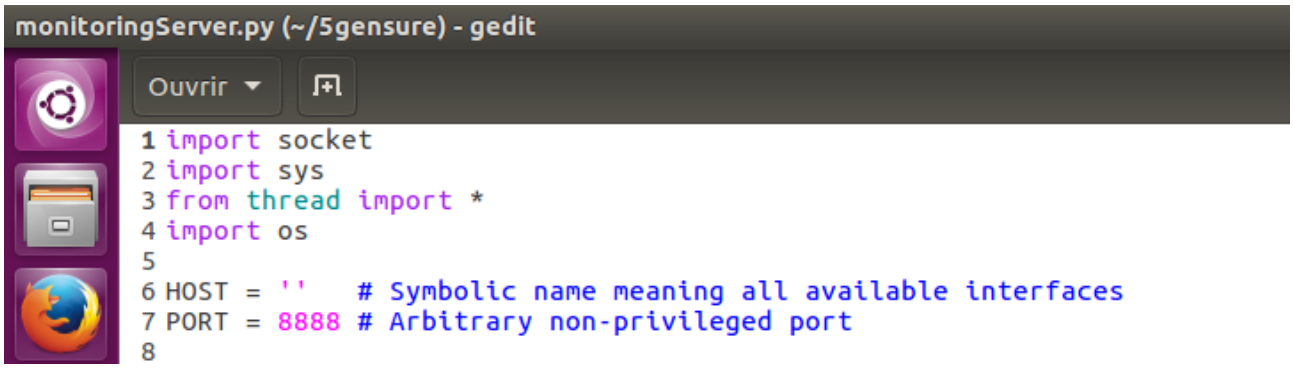

Figure 3: Configuration of "monitoringServer.py"

"monitoringService.py" requires almost the same configuration as "monitoringServer.py": specify the IP address and port where to wait for inputs from the server engines (lines 5 and 6 in Figure 4). HOST = '' means that "monitoringService.py" will wait for input on all the available interfaces. As previously mentioned, to force "monitoringService.py" to listen only to a given IP address, replace '' by the given IP address.

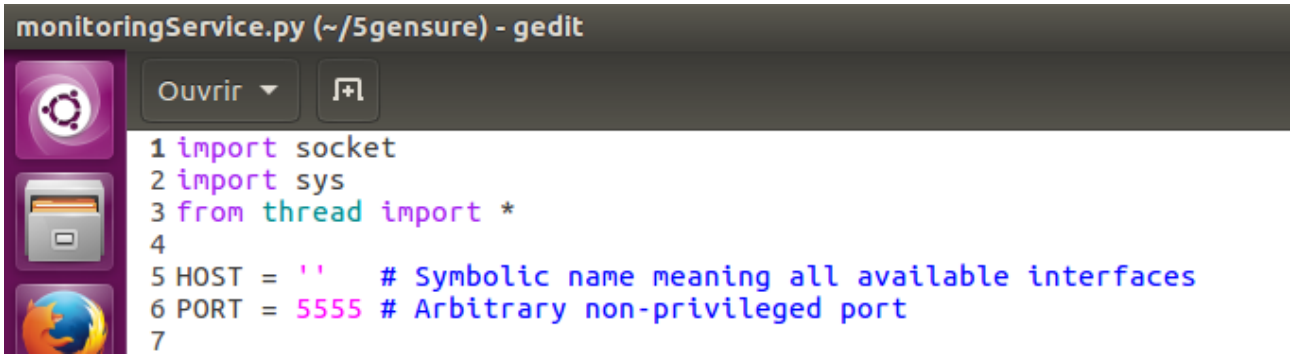

Figure 4: Configuration of "monitoringService.py"

2.3 Enabler Installation

In this link [3], you can find compressed folder "GenericCollectorInterface.zip" containing three python scripts, i.e., "monitoringClient.py", "monitoringServer.py" and "moitoringService.py", an example of a security policy file "securityPolicy.txt", and an example of an XML report "report.xml".

The python script "monitoringClient.py" should be installed in all the network components that are supposed to collect and send reports to the Server engine. The python script "monitoringSrever.py" should be installed in a host that is accessible by the entire network component and that can access the host where installed the "moitoringService.py" script. "securityPolicy.txt" should be in the same host as the "monitoringSever.py". "report.xml" can be duplicated in all the network components (This example can be used to facilitate the tests). Afterwards, "Server_database" should be created in the machine hosting "monitoringServer.py". Similarly, "service_database" should be created in the machine hosting "monitoringService.py".

2.4 Troubleshooting

The message error in Figure 5, can occur in the server engine side (“monitoringServer.py”). It is due to socket issues. This problem does not require any solution. It will be solved by the following report transmission process.

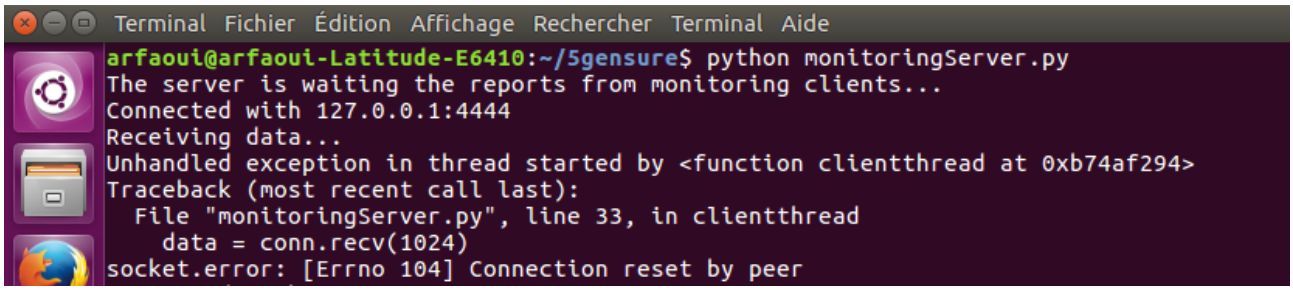

Figure 5: Socket error

3 User and Programmer Guide

This section describes how to use and “program” the enabler.

3.1 User Guide

After installing and configuring the applications

1- Start the server engine (Figure 6):

python monitoringServer.py

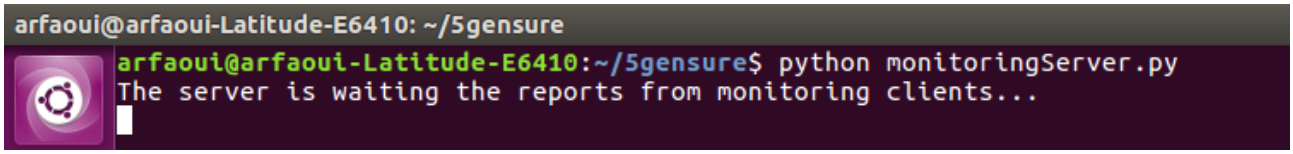

Figure 6: Start the server engine

At this moment, the server engine is ready to get any report from any client and then transmit it to the corresponding Service.

2- Start the service (Figure 7):

python monitoringService.py

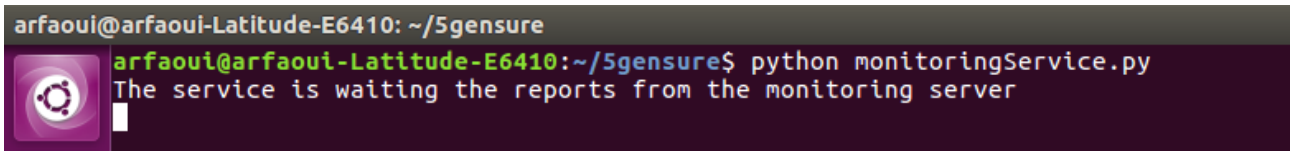

Figure 7: Start the service

At this moment, the service is ready to receive any report from any client.

3- Start a report transmission by the client engine (Figure 8):

python monitoringClient.py

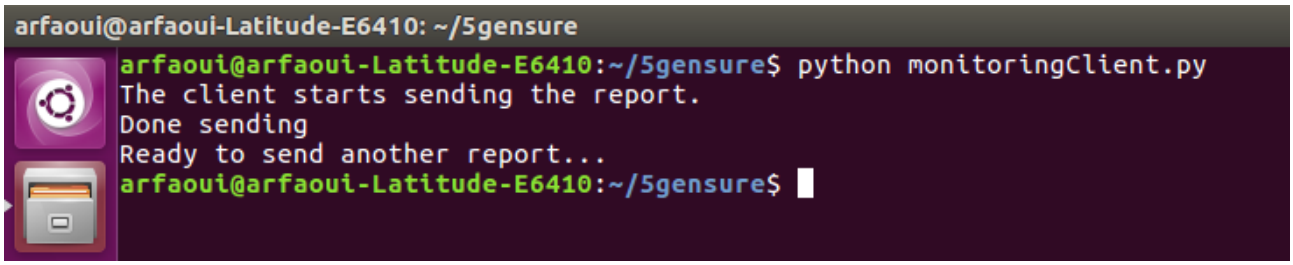

Figure 8: Start a report transmission

This command triggers the transmission of a report from the client engine to the server engine (Figure 9 and Figure 10).

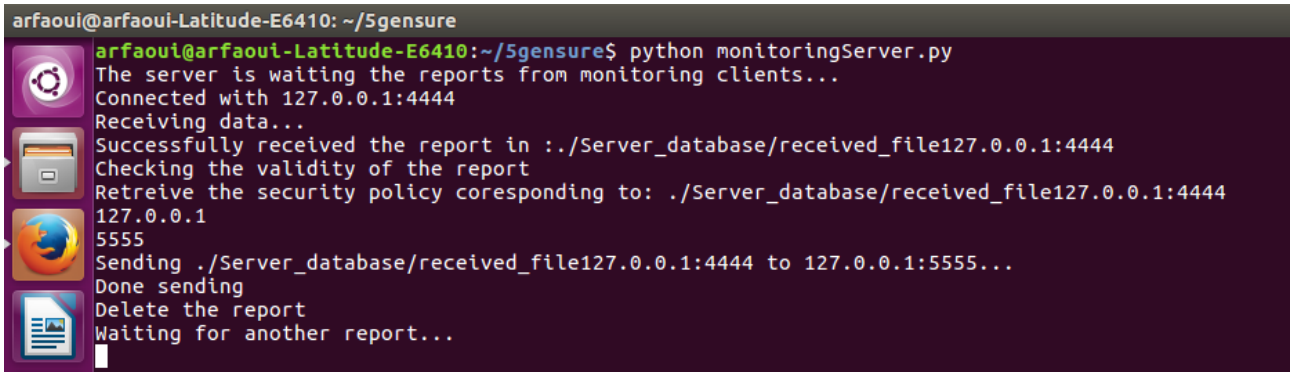

Figure 9: Server engine when receiving a report

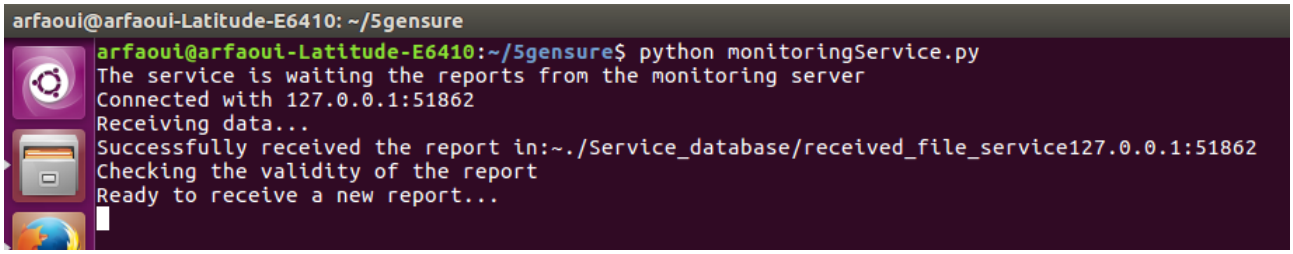

Figure 10: Service when receiving a report

3.2 Programmer Guide

This section is not applicable for this enabler.

4 Unit Tests

4.1 Unit Test 1

The objective of the first test is to separately check that the scripts “monitoringServer.py”, “monitoringClient.py” and “monitoringService.py” work properly. This means that the test will check that every script listens and sends on the designated IP address / port, and have the necessary access rights to the databases (i.e. “Server_database” and “service_database”) and to the security policy file. To perform this test, first, ping every script on the designated IP addresses / ports and monitor the exchanges by Wireshark. Then, check the access rights of the relevant databases and security policy file.

4.2 Unit Test 2

The goal of this test is to check that all the software components correctly communicate with each other, and hence, the XML report is correctly transmitted from a client engine to the corresponding service. To do so, after following the three steps described in section 3.1. Check manually that the file “report.xml” has been transmitted only to the authorized service. Using Wireshark to trace the various exchanges between the different scripts / machines can be helpful.

5 References

[1] D. C. Hansen, "Fast Data: Go Big. Go Fast.," *Data Integration - Inside Oracle's Data Integration community,* 2012.

[2] "Deliverable D3.2 5G-PPP security enablers open specifications (v1.0)," 2016.

[3] 5G-ENSURE, "Software Delivery information for R1," [Online]. Available: https://workspace.vtt.fi/sites/5g-ensure/SitePages/5G%20ENSURE%20-%20Software%20Delivery%20information%20for%20R1.aspx.