VeriDas

/Phygital

/Integration PHACEX protocol

Index

/Introduction

The Veridas biometric terminal is a system capable of providing authentication and authorization services for different use cases. In addition, it allows the integration of the authentication service with an **External Access Control System (EACS)**.

This document aims to serve as a guide for the integration of the Veridas biometric terminal authentication service with an EACS through the **PHACEX (Physical Access Control Exchange)** communication protocol. The PHACEX protocol is intended for the communication between components via the standard network protocol **Mosquitto (MQTT)**.

/Pre-requirements

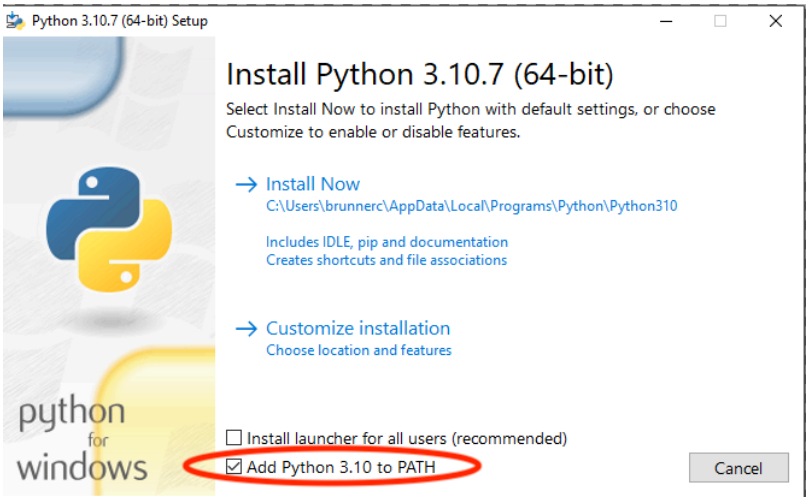
To start the integration, it is necessary to install the following components on the computer on which the EACS is to be run:

- **MQTT**: https://mosquitto.org/download/
  After installing MQTT, the service must be started in case it does not start automatically.

- **Python**:
  On **Linux**, Python comes pre-installed, so there's no need to download it. However, it's important to note that the executable requires either Python 3.8 or Python 3.10 to function properly. To check if Python is installed and if it's one of these two versions, open a command prompt window and enter the following command: *$ python3 -V*
  If the output does not show a Python version or shows a Python version other than 3.8 or 3.10, you will need to install it. To do this, run the following command: *$ sudo apt install python3.10*

  In the case of **Windows**, as on Linux, the executable also requires Python version 3.8 or 3.10. To check if it is installed and if it corresponds to one of these two versions, open a command prompt window and run the following command: *$ python -V*
  If the output does not show one of the required Python versions, you can install it from this link:
  https://www.python.org/downloads/release/python-31011/

In the installation, It is necessary to add the binary to the PATH (when opening the installer a clickable field is shown that performs the action automatically, see image below). If this field is not checked, the path where python is installed must be added manually to the system or user PATH following these instructions.



/Attachments

Alongside this integration guide, a set of files essential for executing and validating the integration is included:

main.py

File that creates and runs the EACS Mosquitto client (class *Client* found in the file *eacs.py*) based on the *settings.yml* file.

settings.yml

Configuration file that determines the EACS behavior.

pyphacex/eacs.py

In this file, an **EACS** is **already implemented** to:

- **Receive *dasgate_alive*** events. A *dasgate_alive* event is published by a Veridas terminal to notify its presence to the EACS.

- **Publish** an ***eacs_alive*** when it receives a *dasgate_alive*. Both events are used to verify communication between both ends (terminal and EACS).

- **Receive** ***authentication_succeeded*** events. An *authentication_succeeded* event is published by a Veridas terminal when it successfully authenticates a user.

- **Process** the ***authentication_succeeded*** event. An *authentication_succeeded* event has the ***user_id*** field, which is the identifier of the user for whom the authorization query will be made. The *main.py* file contains the *get_authorization_response* function, responsible for checking if this *user_id* has access rights. Currently, it:

    - Returns *AuthorizationResponse* with *result="granted"* if the *user_id* is equal to the value of the variable *users/authorized_user_id* present in the *settings.yml* file.

    - Otherwise, it returns *AuthorizationResponse* with *result="denied"* and *reason="unauthorized".*

- **Publish** an **access_granted** if the *get_authorization_response* function returns *AuthorizationResponse* with *result="granted"*, indicating that the user has access rights.

- **Publish** an **access_denied** if the *get_authorization_response* function returns *AuthorizationResponse* with *result="denied"*, indicating that the user does not have access rights.

For a more detailed description of the message contents check the documentation of the keeper API, additionally, to check the type of messages that are to be received check the inspector API.

phacex-0.5.0-py2.py3-none-any.whl

Installable wheel file that contains the **validator**. The validator **simulates** the events that would be generated by a Veridas terminal and generates tests to verify that the communication with the EACS is correct.

run-validator.sh / run-validator.bat

Execution scripts (*run-validator.sh* to Linux and *run-validator.bat* to Windows) to run the EACS, the validator and check if the tests pass.

run_validator.py

Module present in *phacex-0.5.0-py2.py3-none-any.whl* containing the validation tests.

run-eacs.sh / run-eacs.bat

Execution scripts (*run-eacs.sh* to Linux and *run-eacs.bat* to Windows) to run EACS only.

/Pre-validation

To validate that the pre-requirements have been installed correctly and the attachments work as expected, run the *run-validator* file.

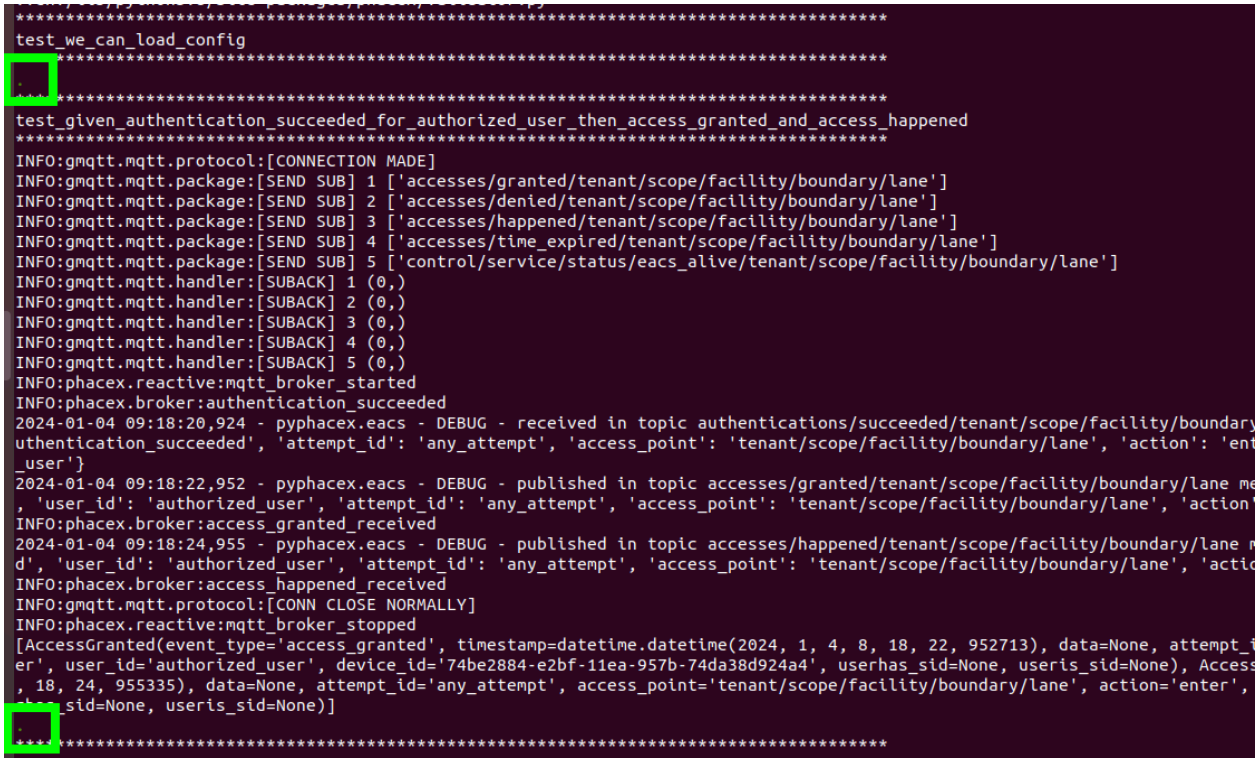
To do that, the first step is to open a command prompt window. Once the window is opened and located in the directory containing the files, run the following commands:

Windows	Linux
$.\run-validator.bat	$ chmod +x ./run-validator.sh # just the first time to give execution rights to the file $ ./run-validator.sh

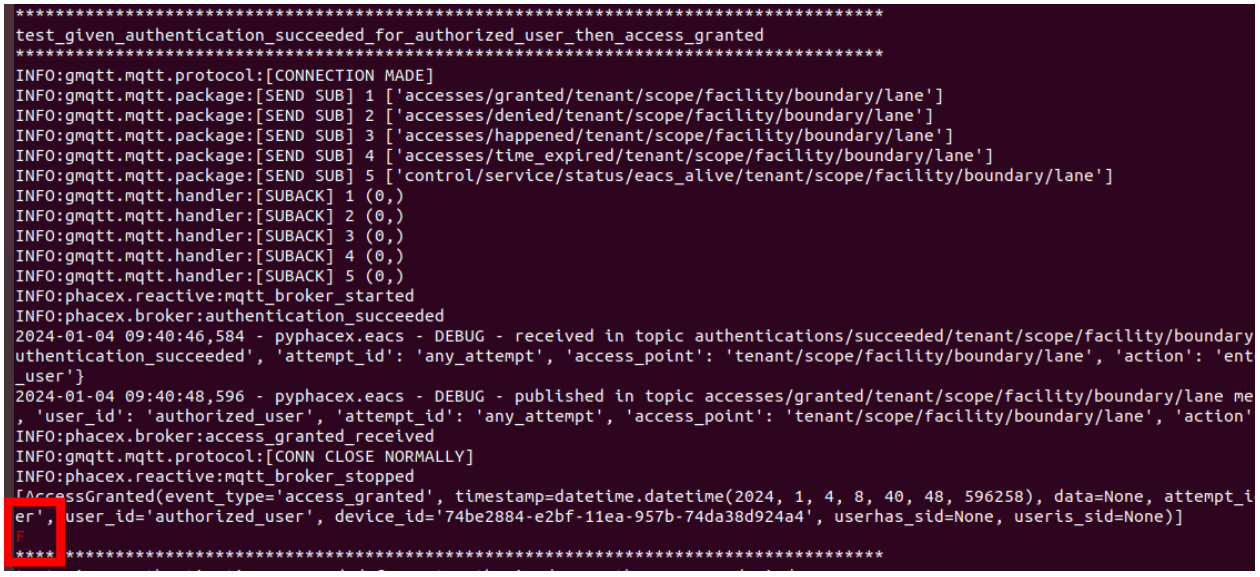
The script will prompt you to select the Python version you want to use. To ensure proper execution, please enter the version available on your computer: “1” for python3.8 or “2” for python3.10.

Once the script is launched, various messages will be observed on the screen (in the case of Windows, a secondary window opens). These messages represent the tests that validate the behavior of the EACS and the validator described in pyphacex/eacs.py section. The results are as follows:

- The window shows that as the tests pass successfully, green dots appear:



- Otherwise, a red F will appear, indicating that the executed test has not been completed with the expected result, tests passed correctly will show a green dot. At the end of the test execution, additional details are shown for those failed tests so that we can fix the created program:



/Verification with Veridas terminal

When you have **obtained** a **Veridas terminal**, you should power it on and connect it to the same local network as the computer where you are carrying out the execution.

In addition, it is necessary to consult the Veridas team for the ***access_point***, that is, the unique identifier of the terminal, and replace the example *access_point* in the ***settings.yml*** file with the specified *access_point*.

On the other hand, the validator described in the phacex-0.5.0-py2.py3-none-any.whl section is no longer required, as this function is now performed by the Veridas terminal itself. Therefore, it is now only necessary to execute the EACS. To do this, the ***run-eacs.bat*** file (for Windows) and ***run-eacs.sh*** file (for Linux) are provided. To execute it, open a command prompt window and execute:

Windows	Linux
$.\run-eacs.bat	$ chmod +x ./run-eacs.sh # just the first time to give execution rights to the file $ ./run-eacs.sh

Once launched, like the validator file, the script will ask for the Python version you want to use. Then in the command prompt window, you will be able to observe the numerous events exchanged between the Veridas terminal and the EACS (including all ***dasgate_alive*** and ***eacs_alive*** messages):

When a **registered user** authenticates on the Veridas terminal, an ***authentication_succeeded*** event is expected to be published by the terminal, and an ***access_denied*** event is expected to be published by the EACS. This triggers a red screen on the terminal. This is because in the ***settings.yml*** file, the *user_id* of the authenticated user does not match the value of the ***authorized_user_id*** variable. If the *user_id* in the settings.yml file is replaced with the authenticated user's *user_id*, subsequent authentication attempts on the Veridas terminal will result in the EACS

publishing an ***access_granted*** event, triggering a green screen display on the terminal.

NOTES

- **To stop the execution of EACS, type "*exit*" in the command prompt window where the 'run-eacs' file is running**
- **To apply changes to any file, it is necessary to stop the execution of 'run-eacs' and restart it**

/Adapt files to complete the integration

settings.yml

The initial step to complete the integration is to properly define the ***settings.yml*** file, specifically the ***events*** variable. Veridas and the integration system must **collaboratively define** the behavior of the entire system (terminal and EACS):

- If access control is to be handled by the EACS, the variables *access_granted* y *access_denied* should be *True* → Standard behavior.
- Additionally, if the external server will manage the user's passage through a physical gate and notify the Veridas terminal, the variables *access_happened* and *access_time_expired* should also be set to *True*.
- On the other hand, if the server is not responsible for notifying the user's passage through a physical door and only handles the authorization process, the variables *access_happened* and *access_time_expired* should be set to *False*.

main.py

In the main.py file, the following classes are defined:

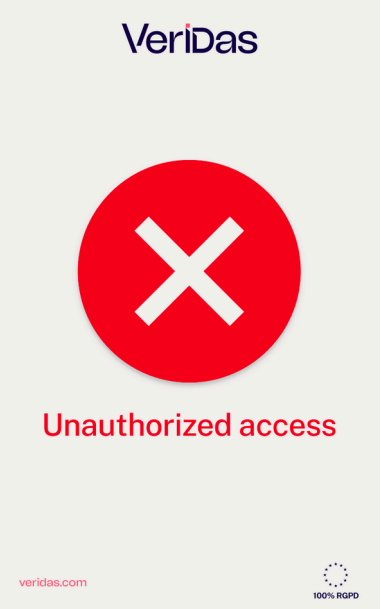
- ***UserAuthorizerCustom***: This class contains the ***check_user_is_authorized*** function, responsible for verifying, based on the *user_id*, whether an authenticated user has access rights or not. By default, it contains content to

perform the Pre-validation and Verification with Veridas terminal sections. However, this function is meant to be modified to complete the integration.

- ***AccessControllerCustom***: This class contains the ***access_happened*** function, which reports the user's passage through a physical door. Therefore, this function only needs modification if the *access_happened* and *access_time_expired* variables in *settings.yml* have been defined as *True*.

/Annex Customizing the access denied images on the terminal

The terminal will display an "Access Denied" screen based on the specific reason published by the EACS (External Authorization and Consent Service). For the default "unauthorized" reason, the terminal displays this generic image:



To customize the access denied screen for different scenarios, you must ensure the EACS publishes the appropriate **reason** code. This requires modifying the reason variable within the *AuthorizationResponse* object returned by the *get_authorization_response* method (located in the *main.py* module). Use ***snake case*** nomenclature convention for these custom reasons, such as *wrong_location*, *canceled_user*, or *capacity_exceeded*.

For instance, if the published *access_denied* reason is *capacity_exceeded*, the terminal will display a specific image for that scenario:

Important Note:

To enable this access denied image customization, please contact the Veridas Customer Success team and **provide a list of the different reason codes** you intend to use. This will allow them to create the corresponding images for your integration.