Configuration Manual

MSc Research Project
MSc Cybersecurity

Forename Surname
Student ID: X21179506

School of Computing
National College of Ireland

Supervisor: Raza Ul Mustafa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Corey Gallagher……

Student ID: ……x21179506…….……

Programme: ……MSc Cybersecurity…… **Year:** ……2…….

Module: ……Research Project…….……

Lecturer: ……Raza Ul Mustafa…….……

Submission Due Date: ……31/01/2024…….……

Project Title: ……Enhancing CAN Bus Security Using Message Authentication…

Word Count:2404 **Page Count:** ……11…….……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……31/01/2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Corey Gallagher
Student ID:X21179506

1 Introduction

This configuration manual will go into detail of how the environment was set up to make the testing of the CAN bus possible. The manual will also cover in detail all the python scripts that were created and written which were used to test both the latency of the network and the enhanced security (Message Authentication) which was added to the CAN messages. The research question being investigated is **How can the implementation of message authentication mechanisms enhance the security of the CAN bus communication in vehicles?** After reading through this configuration manual, it should be clear to the reader how the environment was configured, and the reader should have a good understanding of the scripts used for the testing and for the generating of results.

2 Creation of Environment

For this investigation to take place it was found that using socketCAN API for Linux was the best approach as it did not require any additional hardware, which would have been difficult to acquire along with being costly. socketCAN allows for the creation of a virtual can bus where the CAN network can be simulated on the Linux environment. A virtual machine (VM) was created, and the Ubuntu operating system was installed on it. Ubuntu allows for high configuration in a controlled environment, the VM also keeps any configuration separate from the host machine therefore safeguarding the host from potential attacks or misconfigurations. It also allows for snapshots and clones of the VM to be created at any time so if something fails in the VM it can be rolled back to the last working state. SocketCAN API is then installed on the VM which will be used to create the virtual CAN and allow for the configuration of the scripts.

By default, when the Ubuntu machine is launched there is no running VCAN interface, to combat this a shell script was created. This shell script first checks for any existing VCAN interface. If there was no existing interface, then the script will be run in root privilege mode and the interface VCAN interface will be brought online. The shell script that was used for bringing the interface online is shown below in **Figure 1.**

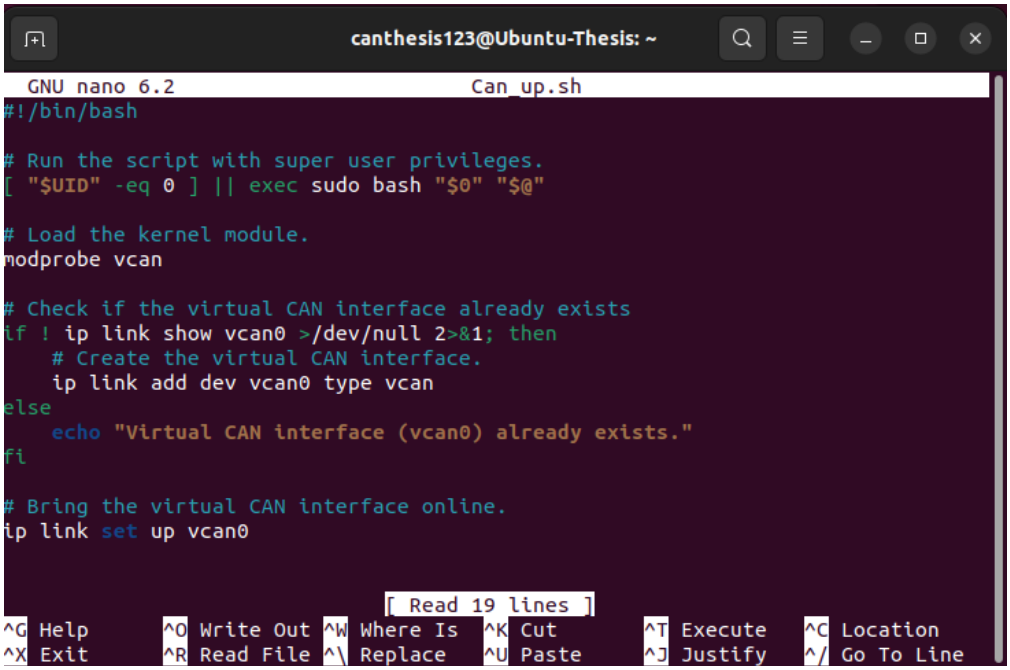

Figure 1: Can_up.sh script to bring interface online.

Once the interface is brought online then a simple test can be done to ensure everything is up and running correctly. The command **"cangen vcan0"** can be used to send randomly generated CAN messages across the created CAN network. By opening a second terminal and then running the command **"candump vcan0"** the randomly generated messages that are being transferred across the CAN interface will be displayed in real time. If the second terminal, then displays a growing list of randomly generated CAN messages the virtual interface has been set up and configured correctly.

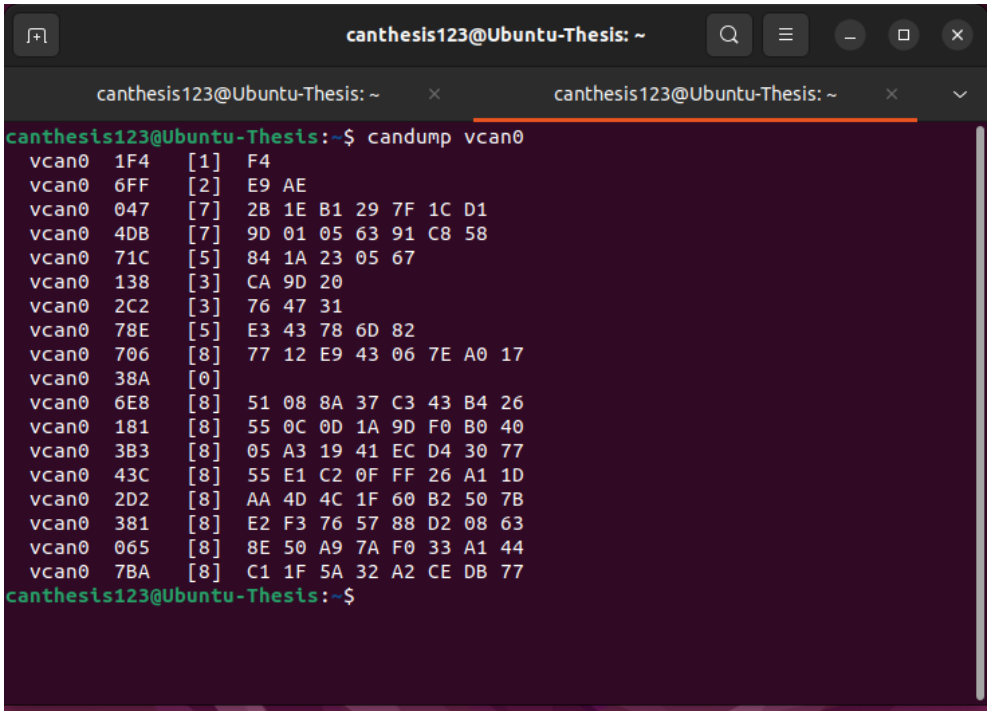

Figure 2: Candump command running sending random CAN messages.

Once these messages are being displayed and seen by the user then the configuration of the environment has been completed successfully. The next step is to create the python scripts that

will be used for the testing of the environment, and this will be the main backbone of the investigation.

3 Python Scripts for Testing

To carry out the examination and the evaluation of adding the message authentication to the CAN messages it was necessary to create four Python scripts. These four Python scripts were the following:

1. **attack_script.py**
2. **no_security_receiver.py**
3. **can_sender_with_auth.py**
4. **can_receiver_with_auth.py**

3.1 attack_script.py

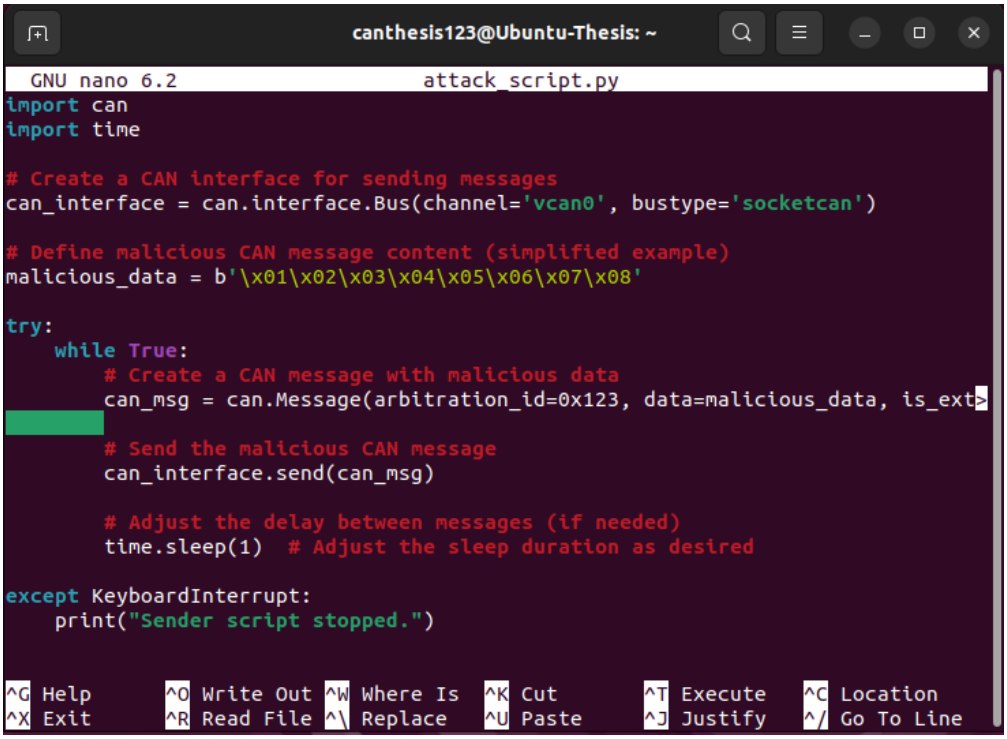

Figure 3: attack_script.py

The above script is used for the attacking testing of the CAN network. The script creates a CAN interface and then sets the malicious data. In this case the malicious data is a simplified form, but it can include data that would take advantage of the lack of message authentication of the CAN bus. The script enters an infinite loop where it continuously sends the malicious CAN messages until the script is cancelled by keyboard input from the user. Once the script is cancelled the message “Sender script stopped” is printed.

3.2 no_security_receiver.py

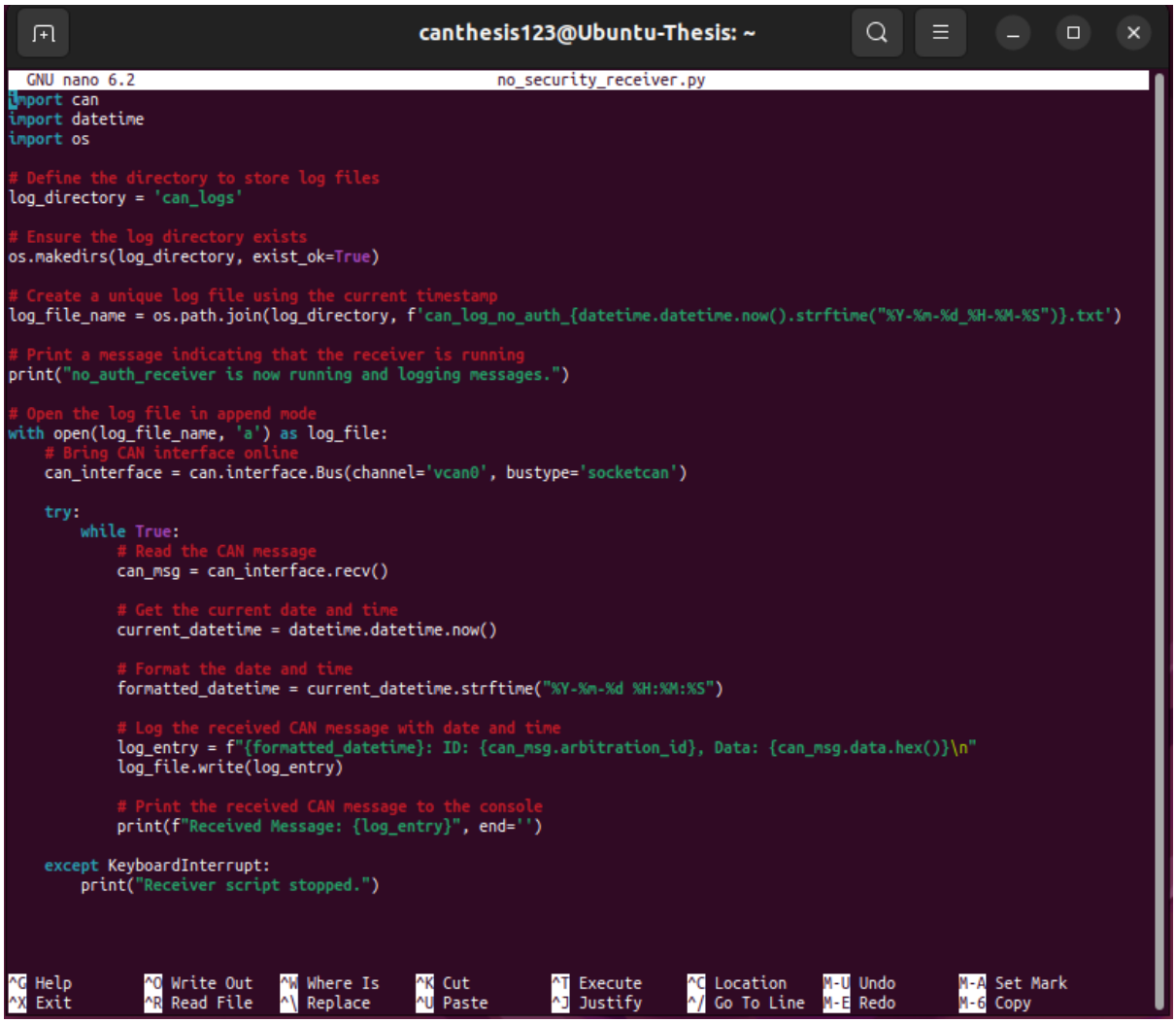

Figure 4: no_security_reciever.py

This script was created as a receiver node on the CAN bus. Firstly, the script will create a directory called "**can_logs**". This directory will be used to store the log files from the messages that are being received. The script then creates a logfile and gives it a unique name based on the time stamp when the log file was created. The script then enters a loop where it reads and stores all the messages that are being received and gives them a time stamp. While doing this it also outputs on screen the messages that are being logged to give the user visual feedback. Once the user inputs Ctrl + C to cancel the running of the script the log file is stored and saved and can be used for analysis in future.

3.3 can_sender_with_auth.py

Figure 5: can_sender_with_auth.py

This is the script that sends CAN messages with added message authentication using HMAC (Hash-based Message Authentication Code). The script defines a secret key which in this case is "**MyKey**" in byte format. Like the other scripts it then enters an infinite loop which will continuously send authenticated CAN messages. The HMAC calculation then takes place which uses the HMAC library and SHA-256 as the hash function. The message is then sliced so that it is 4 bytes in length. The HMAC value is then calculated by using the secret key and the data to be sent. The script then checks the message length to ensure that it does not exceed 8 bytes which is the maximum size for the standard CAN messages. The script then appends the HMAC value which was calculated to the original CAN message to create the authenticated CAN message. The authenticated CAN message will then be sent one time every second until the user interrupts the script.

3.4 can_receiver_with_auth.py

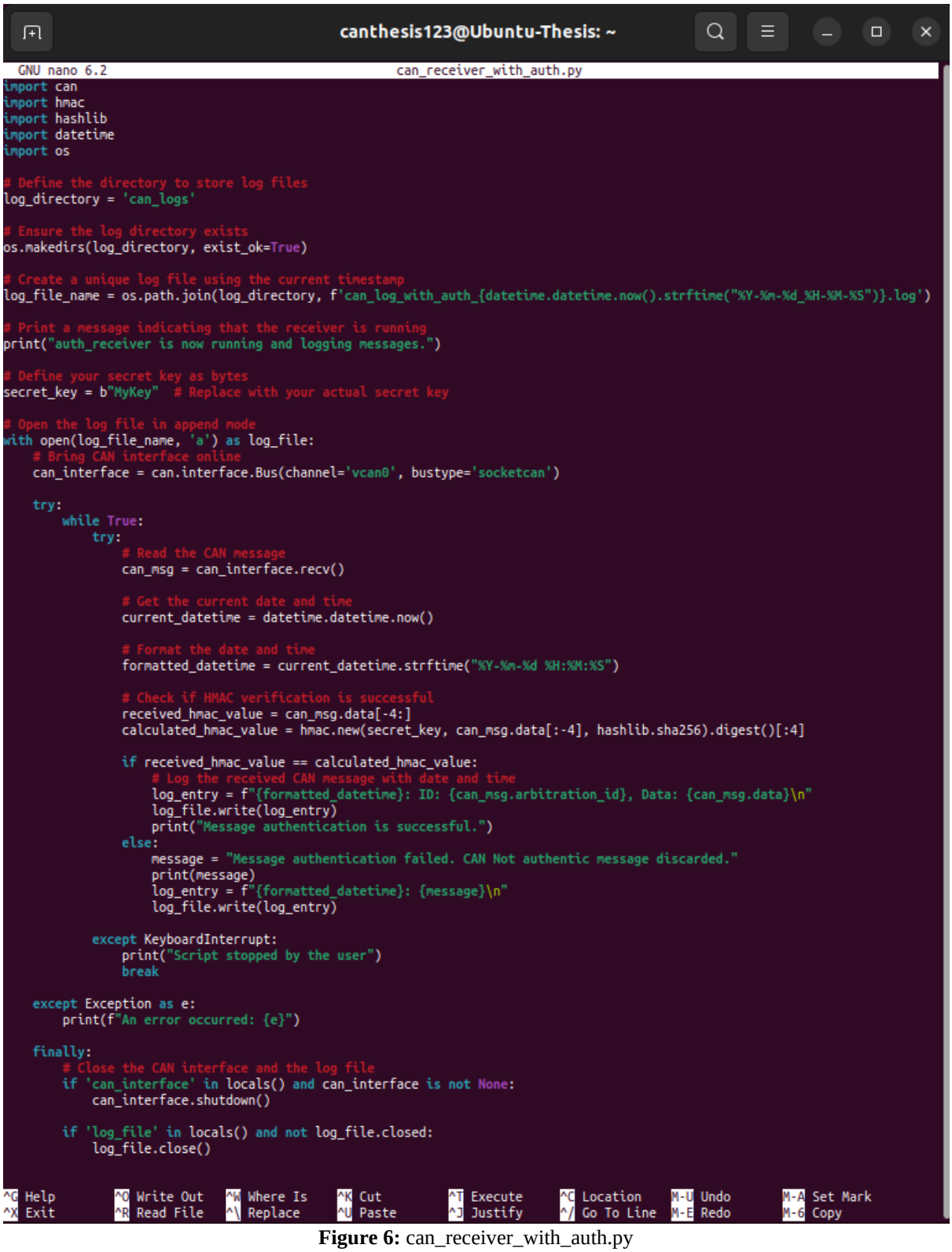

Figure 6: can_receiver_with_auth.py

can_receiver_with_auth.py is a script that is used to receive the authenticated CAN messages. Like the other receiver script that did not have any authentication built in the script defines the "**can_logs**" directory if it has not been so already. Then as before the log file is created based on the timestamp and is stored in this directory. The secret key that was used for the sender script is shared with the receiver script. The script then enters the "**while true**" infinite loop

where it can continuously receive and process the CAN messages. HMAC verification then takes place. The script reads the message received and retrieves the date and time which the message was received at. A comparison then takes place where the script compares the received HMAC value in the CAN message with the calculated HMAC value. If the verification is successful and the values match, then the message is logged with the date and time attached. If the verification fails a message is printed on the console and the details are logged in the log file. The script will continue to run until the user interrupts the process using keyboard input.

These four scripts which have been detailed and written will be the main scripts that will be used for the testing of the message authentication. Although these scripts are the main focus as they include all the authentication and security techniques that are the foundation of the study an additional two scripts also need to be created for the measuring of the effectiveness of the security mechanisms. These two scripts will be detailed in the next section.

4 Metrics Scripts

4.1 Metrics_no_auth.py

The **metrics_no_auth.py** script was the script that was used in the testing phase of the investigation for the receiver node that did not contain any message authentication. This script was used to record the latency on the messages being sent and received in the non-authenticated sender and receiver scripts. The function defined **monitor_latency_throughput_no_auth** takes in the CAN channel where the CAN message is being sent and then also takes in a duration in seconds for the metrics script to run, in this case it will be 60 seconds. The variables **total_latency**, **start_time**, **end_time** are then all defined before entering the infinite loop. Once inside the infinite loop the script receives the CAN message using **can_interface.recv()** the latency on this message is calculated by using the difference in the timestamps. The script then prints to screen the latency for each CAN message being received. After the 60 seconds running of the script the average latency and throughput of all the sent messages are calculated and the results are printed to screen and stored in a log file inside the **Metrics_Logs** directory. The script can be seen below **Figure 7.**

4.2 Metrics_with_auth.py

The **metrics_with_auth.py** script is like the **metrics_no_auth.py** script except it now considers the HMAC (Hash-based Message Authentication Code) verification for CAN messages. The function **monitor_latency_throughput** takes the HMAC and the duration in seconds as parameters. The variables are then set up to track the messages received and the total latency. As before the script then enters an infinite loop. Inside the loop the script receives the CAN messages and then performs the HMAC verification on the message. The last 4 bytes of the received message (**received_hmac_value)** is extracted and the HMAC is then calculated on the rest of the received message using the defined secret key. If the received and the calculated HMAC values are matching, then the message is authentic. The latency and throughput are calculated like the previous script and the results are also logged in the log file. The script for the **metrics_with_auth.py** can be seen in **Figure 8.**

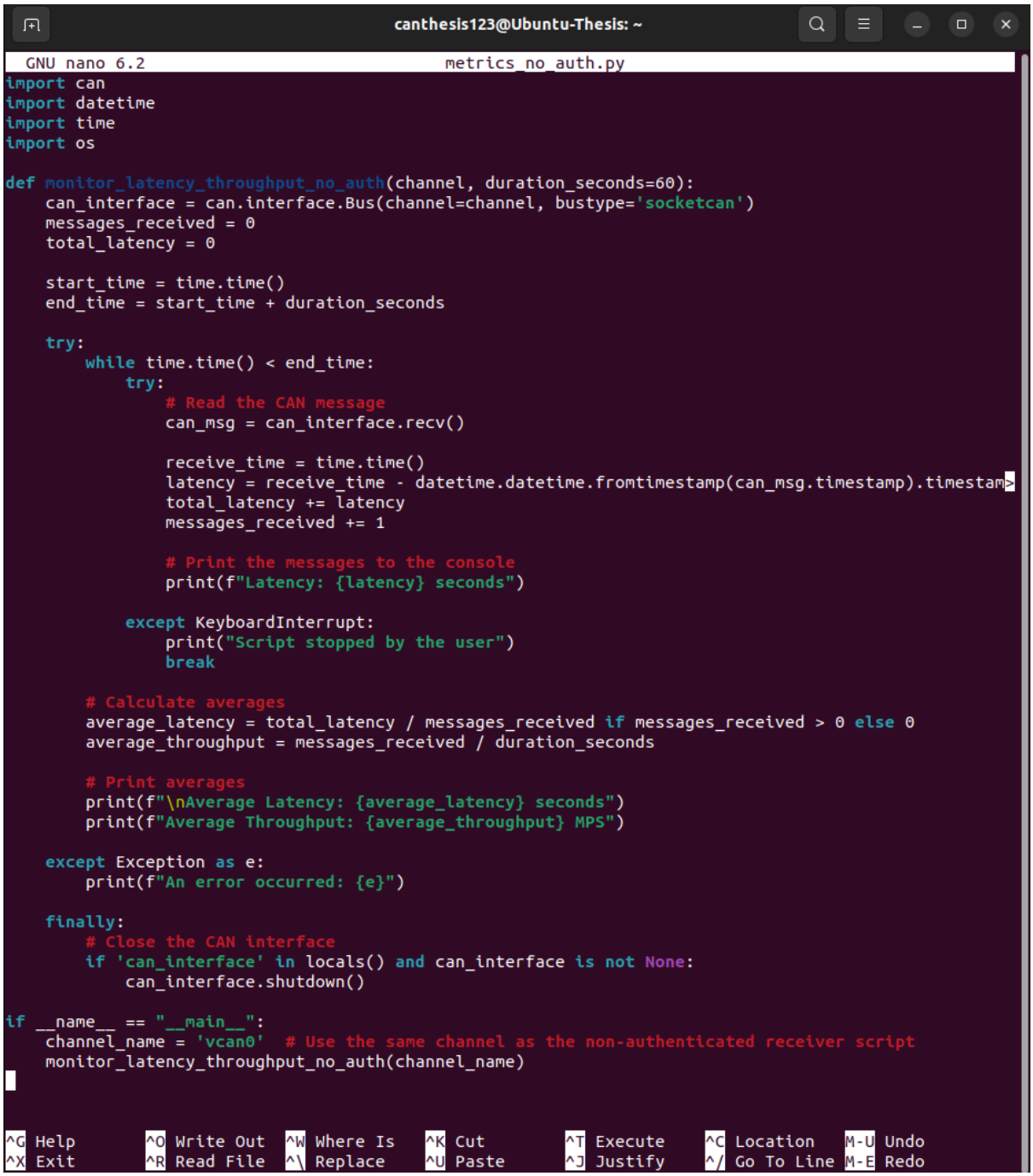

Figure 7: metrics_no_auth.py

Figure 8: metrics_with_auth.py

5 Logs

The logs for the scripts with authentication and without authentication can be found in the directory **can_logs**. These files are created automatically after the user interrupts the running of the script. The file is clearly named and states if it contains authentication or no authentication and is made unique by its timestamp. Similarly, the metrics logs can be found in the directory **Metrics_Logs.** After the metrics scripts are run to completion then the results are stored here in files that are made unique also by their timestamp. Inside these files the results will contain the average latency and throughput of the CAN messages over 1 minute. These logs will be used to gather the results for the analysis of the test.

6 Steps to Conduct Test

1. **Ensure that the VCAN is online:** Before we can begin the testing of message authentication using the scripts, we first need to ensure that the VCAN environment is online. To do this we can open a terminal and use the command **"ifconfig"** in the output of this if we see the interface VCAN0 and see that it is online then we can test it further by opening two separate terminals then running the commands **"cangen VCAN0"** and **"candump VCAN0"**. If the second terminal is displaying a growing number of randomly generated CAN messages, then the VCAN is online, and all configuration is working as expected.
2. **Conduct baseline test:** The baseline test is conducted by running a baseline test where the scripts with no authentication are run and their metrics are recorded. To do this we will use the scripts **attack_script.py** and **no_security_receiver.py**.
    - Start the receiver node by running the **"python3 no_security_receiver.py"** command which will start a receiver CAN node which has no security and will accept all CAN messages.
    - Start the attack script, which is a CAN sender script that has no message authentication and will continuously send CAN messages to the CAN receiver until the script is interrupted by the user. The command to do this is **"python3 attack_script.py"**
    - The Metrics script is then needed to record the latency and get the average time per message received by the node. This is done by using the metrics script that was created for no authentication using the command **"python3 Metrics_no_auth.py".** This script will run and record results for one minute before storing the results.

    These tests are run for a total of 10 times and the average is recorded for use in the baseline test.
3. **Test the message authenticated scripts:** After conducting the baseline tests we need to test the effectiveness and of the scripts which include the message authentication. To do this we will do the following:
    - Start the message authentication receiver. The message authentication receiver should be started which will be used to receive the messages with authentication and receive messages without authentication from the attack script. The message authentication receiver is started using the command **"pyton3 can_receiver_with_auth.py.** The receiver script should only receive messages that have the correct HMAC value.
    - Start the message authenticated sender script by running the command **"pyton3 can_sender_with_auth.py".** This script will continuously send an authenticated CAN message to the receiver node every one second until the user interrupts the script.
    - The metrics script which will record the result of the test for the authenticated messages is needed and this is started by using the command **"python3 Metrics_with_auth.py"** this will record the average latency over one minute and store the results.
    - To test the message authentication is working as expected we will open a third terminal window and ensure that only the CAN receiver with auth is running. Then we will start the attack script as before. The CAN receiver with auth

should not accept any of these messages as they do not have any HMAC authentication in the script.

Like the baseline tests these will be conducted 10 times in total. The results from both these tests can then be analysed and the conclusion can be developed based on the results as to whether the message authentication in the CAN is worth implementing or not in vehicle security measures.

7 Conclusion

This document outlines the key scripts that were used along with how the Vcan0 interface was brought online. The document gives a clear review of each script and details the main lines of code in each script. The screenshots should provide a clear and obvious reference to the reader which line of code and script is being referenced. By reading this configuration manual and understanding each of the scripts then it should be clear the inner workings of each and how they can be used to answer the research question, **how can the implementation of message authentication mechanisms enhance the security of the CAN bus communication in vehicles?**