Dissertation on

"CAN FUZZING USING SAVVY CAN ON ICSim "

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

UE20CS461SC – Automotive Cyber Security

Submitted by:

1	DIVYA SRUJANA	PES1UG22CS832
2	MANASI TAWADE	PES1UG22CS815
3	VANUSHREE DM	PES1UG22CS846
4	VAIBHAVI	PES1UG21EC320

Under the guidance of

Dr. Roopa Ravish
Associate Professor
CSE department

June - Aug 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY

Introduction:-

In the rapidly evolving field of automotive technology, ensuring the security and robustness of Controller Area Network (CAN) systems is paramount. CAN networks, which facilitate communication between various electronic components within a vehicle, are critical for the vehicle's operation and safety. However, these networks are vulnerable to various types of cyber-attacks, including the injection of malformed or unexpected CAN frames. To address this, we propose a fuzzing strategy using SavvyCAN on ICSim, a virtualized environment for simulating in-vehicle networks. This approach leverages frame sender, sniffer, and custom frame-sending techniques to test the resilience of CAN systems thoroughly. By systematically injecting and analyzing a range of abnormal CAN frames, we aim to uncover potential vulnerabilities and propose enhancements to fortify the security and reliability of automotive CAN networks.

Problem Statement:-

Developing a fuzzing strategy using SavvyCAN on ICSim to evaluate the robustness of automotive CAN (Controller Area Network) systems. Utilize frame sender, sniffer, and custom frame-sending techniques to inject malformed or unexpected CAN frames. Analyze the impact on ICSim's virtualized vehicle components to identify vulnerabilities, potential crashes, or unexpected behaviors. Document the findings and propose mitigations to enhance the security and reliability of automotive CAN networks.

Objective:-

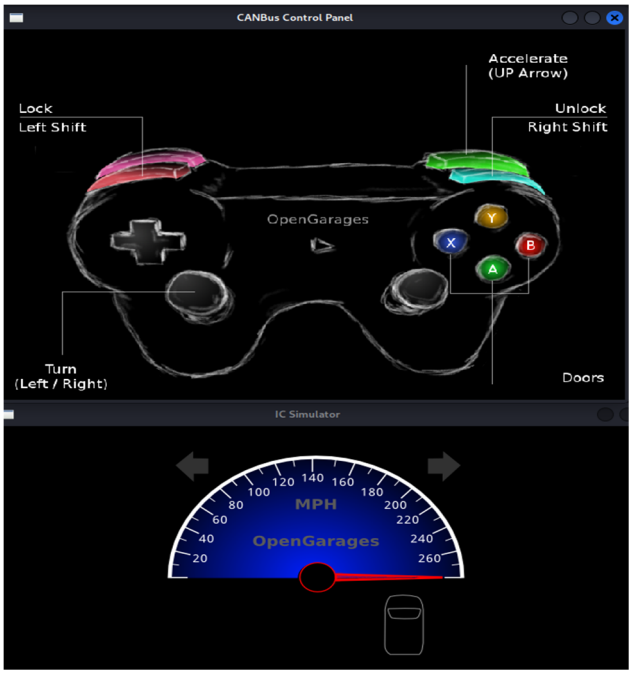
SavvyCAN on ICSim will be used to test and improve the security of automotive CAN systems. This involves:

1. Sending unusual CAN frames.
2. Monitoring vehicle responses.
3. Identifying weaknesses.
4. Suggesting improvements.
5. Enhancing testing methods for better automotive cybersecurity.

PROCEDURE:-

- ICSim
- SavvyCAN
- Qt5
- Qtserialbus

1.Setting up the Instrument Cluster Simulating



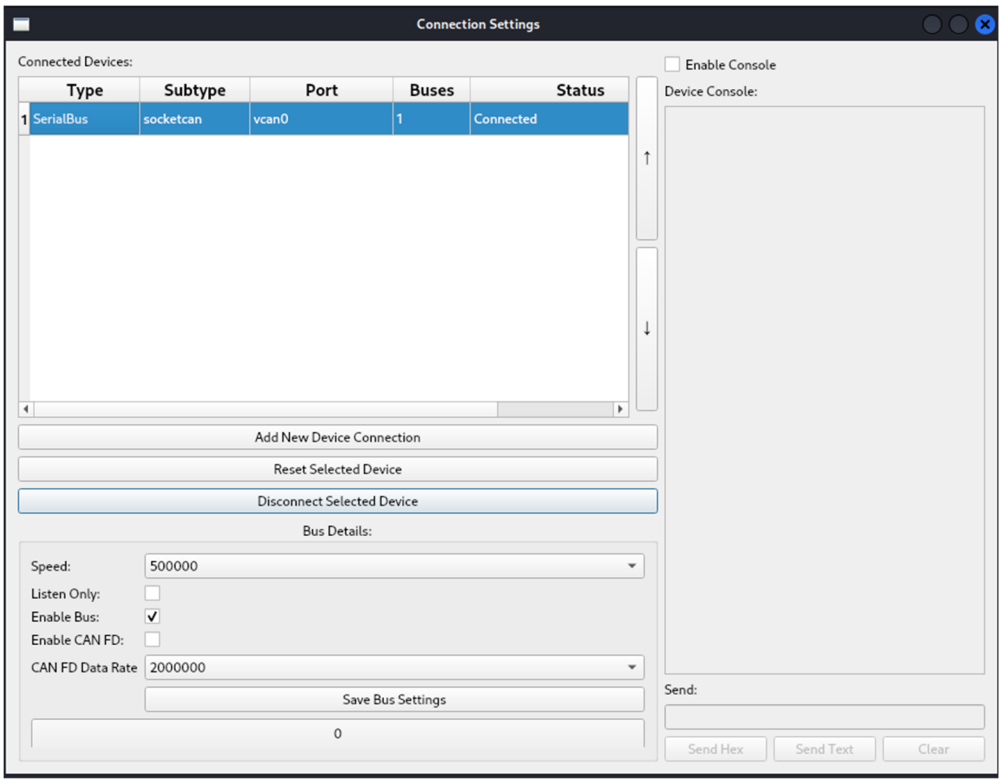
2. Installation Of Savvy Can

SavvyCAN: SavvyCAN is an open-source tool used for analyzing Controller Area Network (CAN) bus traffic in vehicles. It allows users to monitor, log, and interpret CAN bus data.

1. Installation and running in Ubuntu

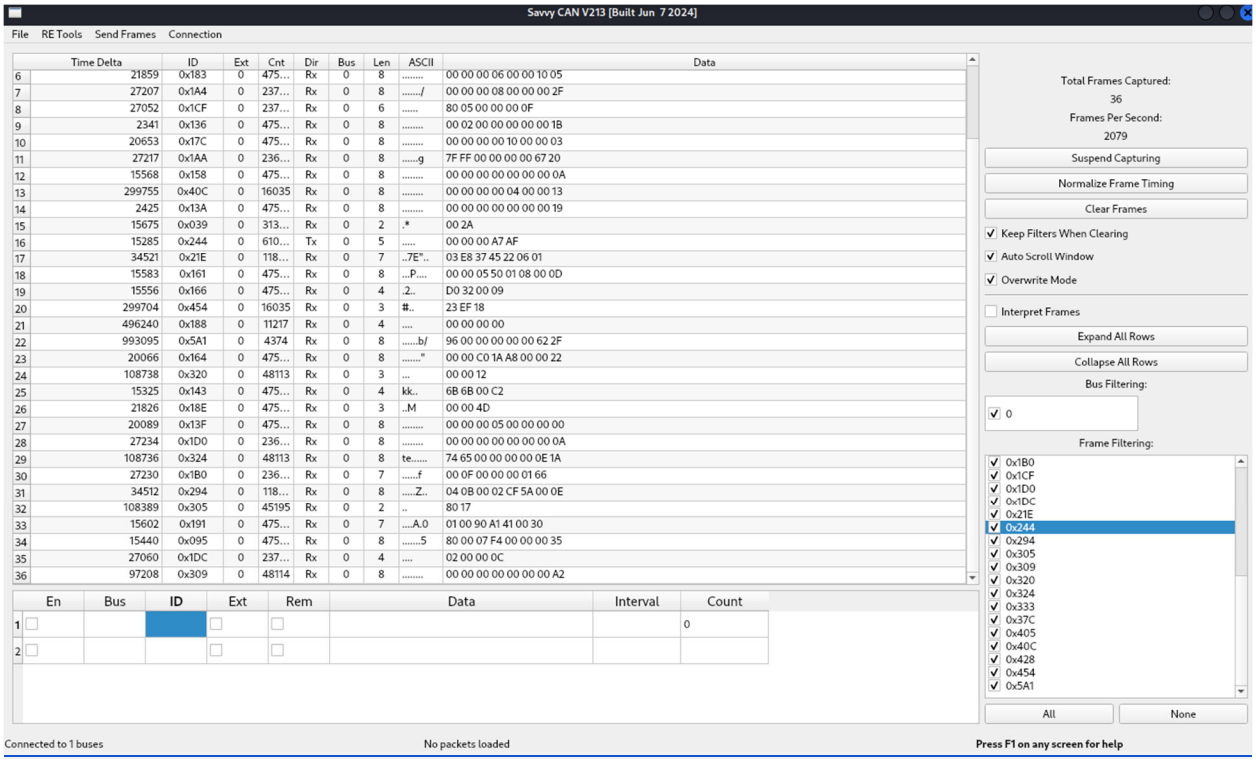
wget

https://github.com/collin80/SavvyCAN/releases/download/V199.1/SavvyCAN-305dafd-x86_64.AppImage



chmod744SavvyCAN-305dafd-x86_64.AppImage#and

./SavvyCAN-305dafd-x86_64.AppImage



3. Installing Qt5

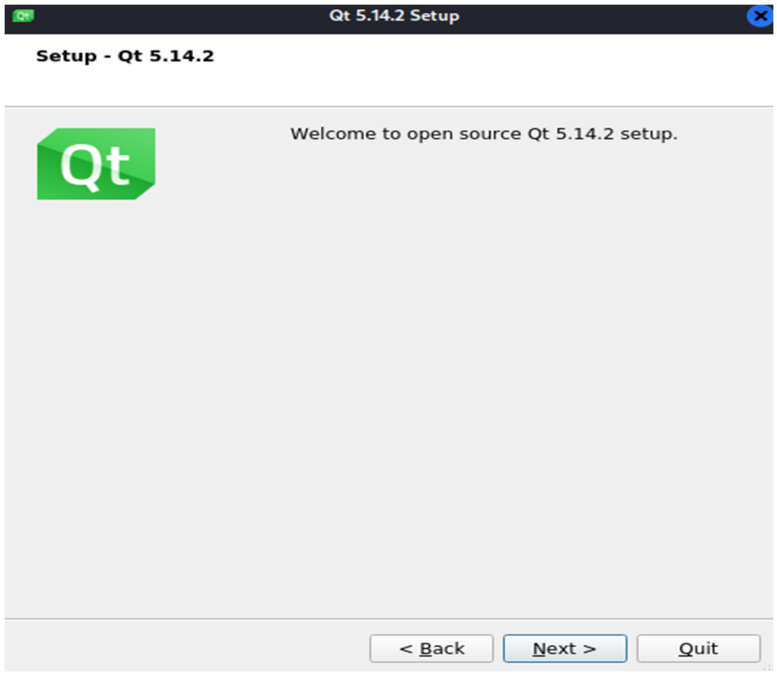
Qt5 is a versatile, cross-platform application framework that is widely used in developing software with a graphical user interface (GUI)

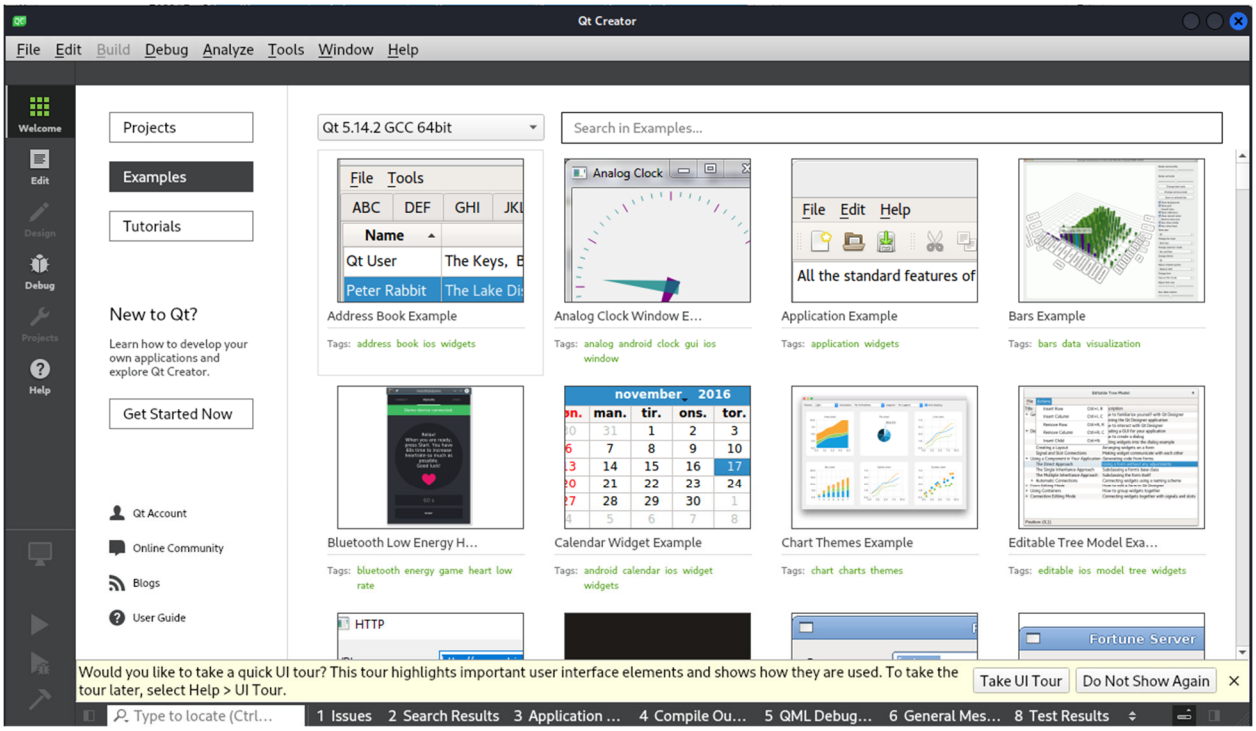
$ wget https://download.qt.io/official_releases/qt/5.14/5.14.4/qt-opensource-linux-x64-5.14.2.run

Login using Email and Password .

$ chmod a+x ./qt-opensource-linux-x64-5.14.2.run

$ sudo ./qt-opensource-linux-x64-5.14.2.run


Qt 5.14.2 Setup
Setup - Qt 5.14.2
Welcome to open source Qt 5.14.2 setup.
< Back
Next >
Quit


Qt Creator
File Edit Build Debug Analyze Tools Window Help
Welcome
Edit
Design
Debug
Projects
Help
Projects
Examples
Tutorials
New to Qt?
Learn how to develop your own applications and explore Qt Creator.
Get Started Now
Qt Account
Online Community
Blogs
User Guide
Qt 5.14.2 GCC 64bit
Search in Examples...
Address Book Example
Analog Clock Window E...
Application Example
Bars Example
Bluetooth Low Energy H...
Calendar Widget Example
Chart Themes Example
Editable Tree Model Exa...
Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.
Take UI Tour
Do Not Show Again
Type to locate (Ctrl...
1 Issues
2 Search Results
3 Application ...
4 Compile Ou...
5 QML Debug...
6 General Mes...
8 Test Results

4. Install qt serial bus

$ sudo apt install qtdeclarative5-dev qttools5-dev g++

$ git clone https://github.com/qt/qtserialbus

$ cd qtserialbus

$ make

$ sudo make install

5.Build SavvyCAN

In order to use qtserialbus, the SavvyCAN's AppImage file that we downloaded

earlier will not work. SavvyCAN has to be built with qmake**.**

$ git clone https://github.com/collin80/SavvyCAN

$ cd SavvyCAN

$ make

6. Starting the SavvyCAN

$ cd SavvyCAN

$ ./SavvyCAN

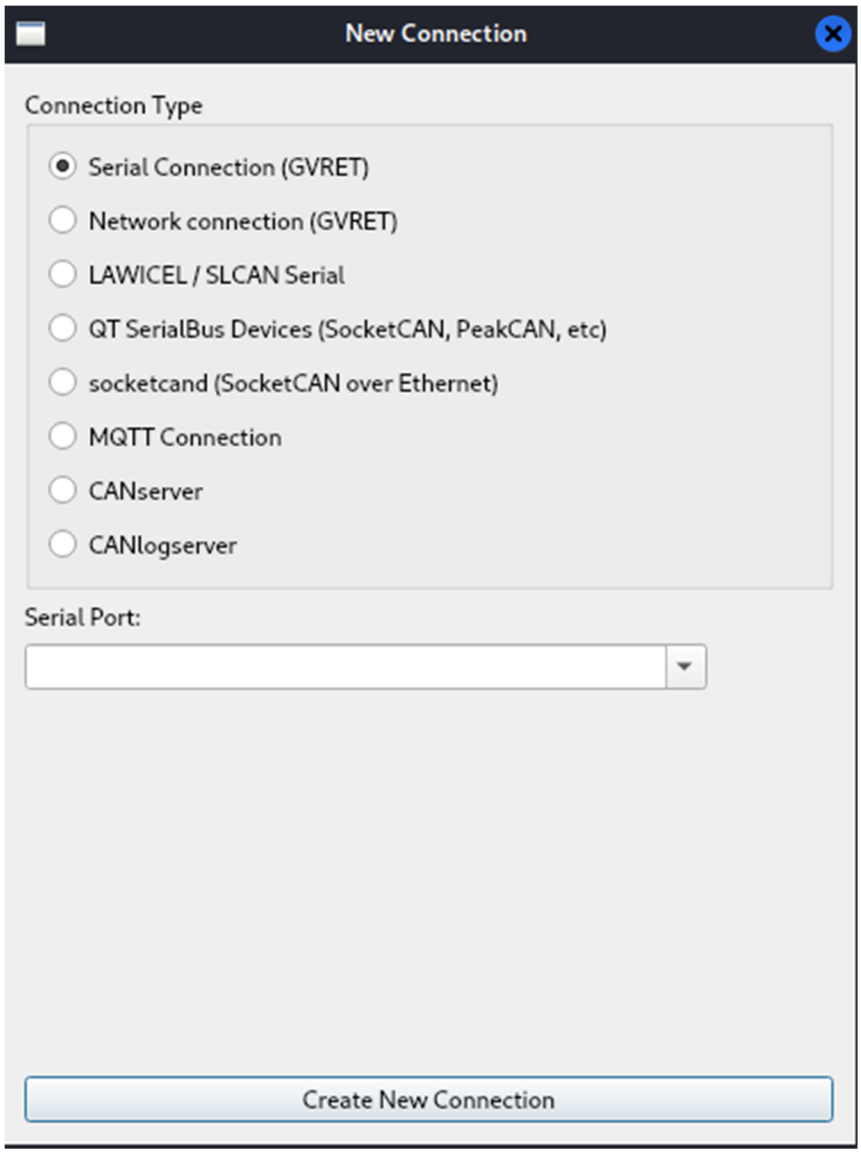
7.Adding vcan0 to SocketCAN

To make a new connection in SavvyCAN,

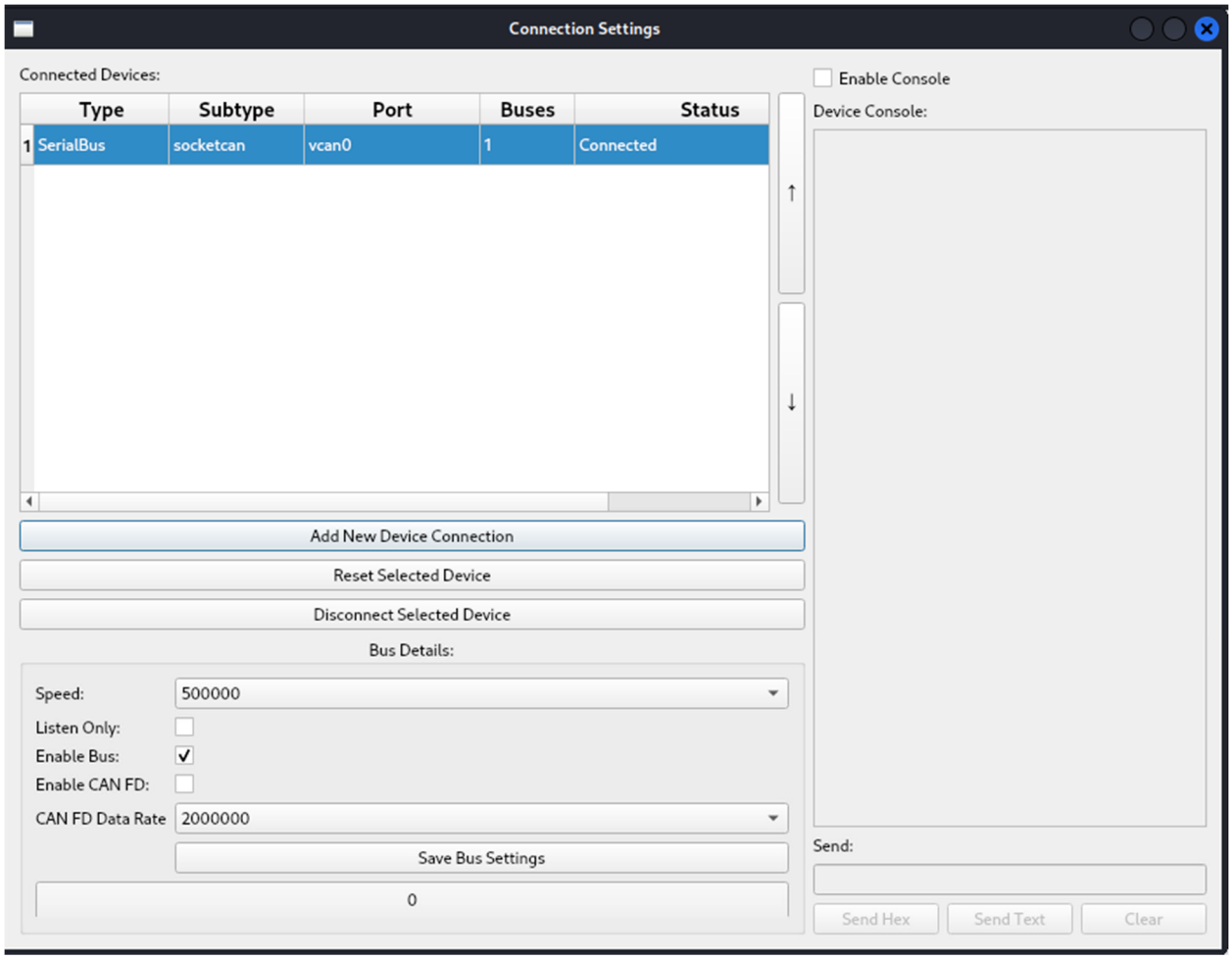
1.Open the SavvyCAN

2. Goto Connection menu -> Open Connection Window -> Add New Device Connection
3. Choose the connection with the following setting

Connection Type as QT SerialBus Devices



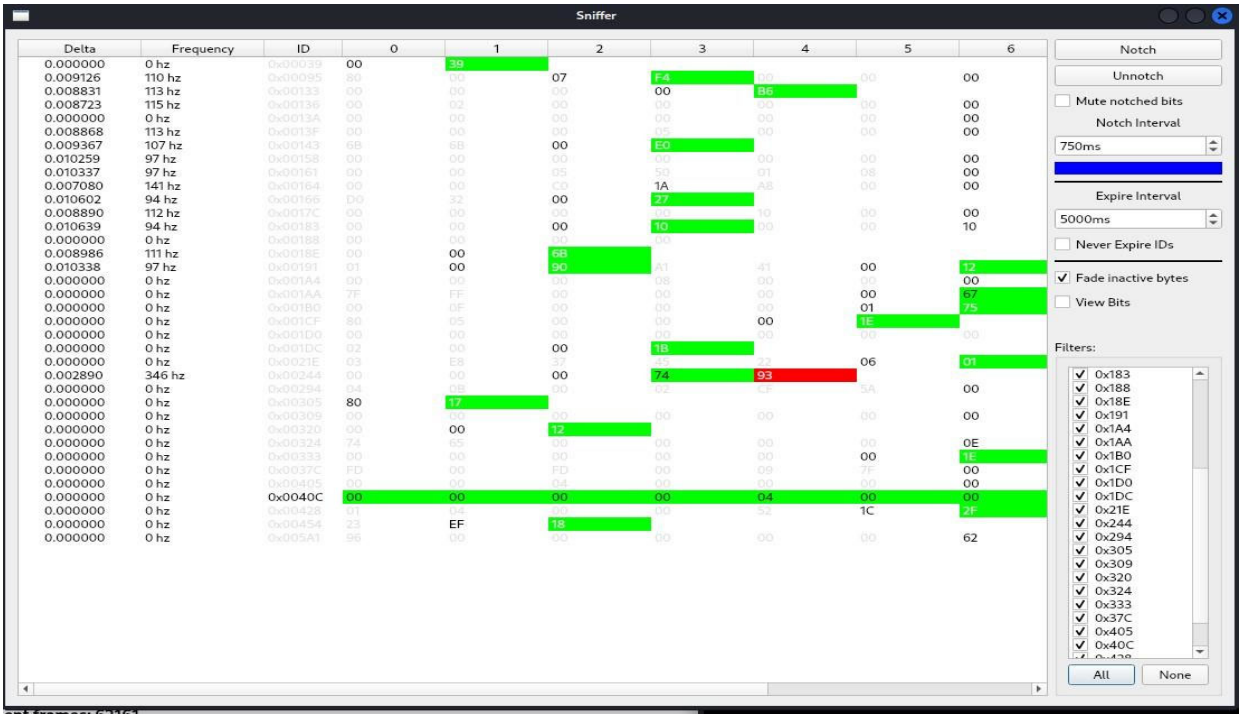
SerialBus Device Type as socketcan Port as vcan0

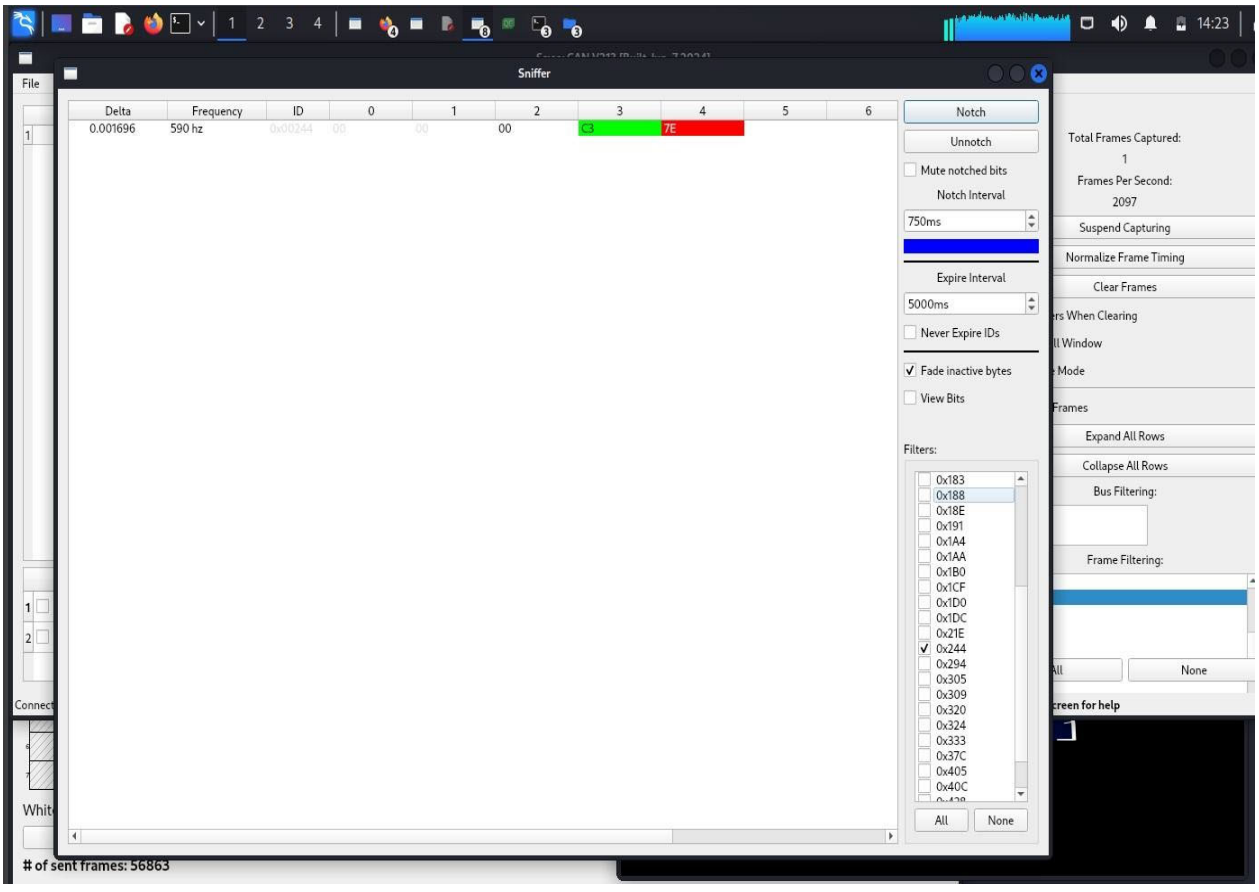


8. Sniffer

A CAN bus sniffer is a tool or software application that listens to the CAN bus network and captures all the messages being transmitted. It does not interfere with the communication but simply monitors the data for analysis purposes. This is analogous to a network packet sniffer used in computer networks.

We find the packet for performing a specific operation , this can be done by splitting packets and here in savvy can we can perform this by selecting the option -> Fade inactive bytes . The green color in the screenshot indicates , the active operation being performed .


Sniffer
Delta
Frequency
ID
0
1
2
3
4
5
6
Notch
Unnotch
Mute notched bits
Notch Interval
750ms
Expire Interval
5000ms
Never Expire IDs
Fade inactive bytes
View Bits
Filters:
0x183
0x188
0x18E
0x191
0x1A4
0x1AA
0x1B0
0x1CF
0x1D0
0x1DC
0x21E
0x244
0x294
0x305
0x309
0x320
0x324
0x333
0x37C
0x405
0x40C
All
None


Sniffer
Delta
Frequency
ID
0
1
2
3
4
5
6
0.001696
590 hz
0x0244
00
00
00
C3
7E
Notch
Unnotch
Mute notched bits
Notch Interval
750ms
Expire Interval
5000ms
Never Expire IDs
Fade inactive bytes
View Bits
Filters:
0x183
0x188
0x18E
0x191
0x1A4
0x1AA
0x1B0
0x1CF
0x1D0
0x1DC
0x21E
0x244
0x294
0x305
0x309
0x320
0x324
0x333
0x37C
0x405
0x40C
All
None
Total Frames Captured:
1
Frames Per Second:
2097
Suspend Capturing
Normalize Frame Timing
Clear Frames
Expand All Rows
Collapse All Rows
Bus Filtering:
Frame Filtering:
# of sent frames: 56863

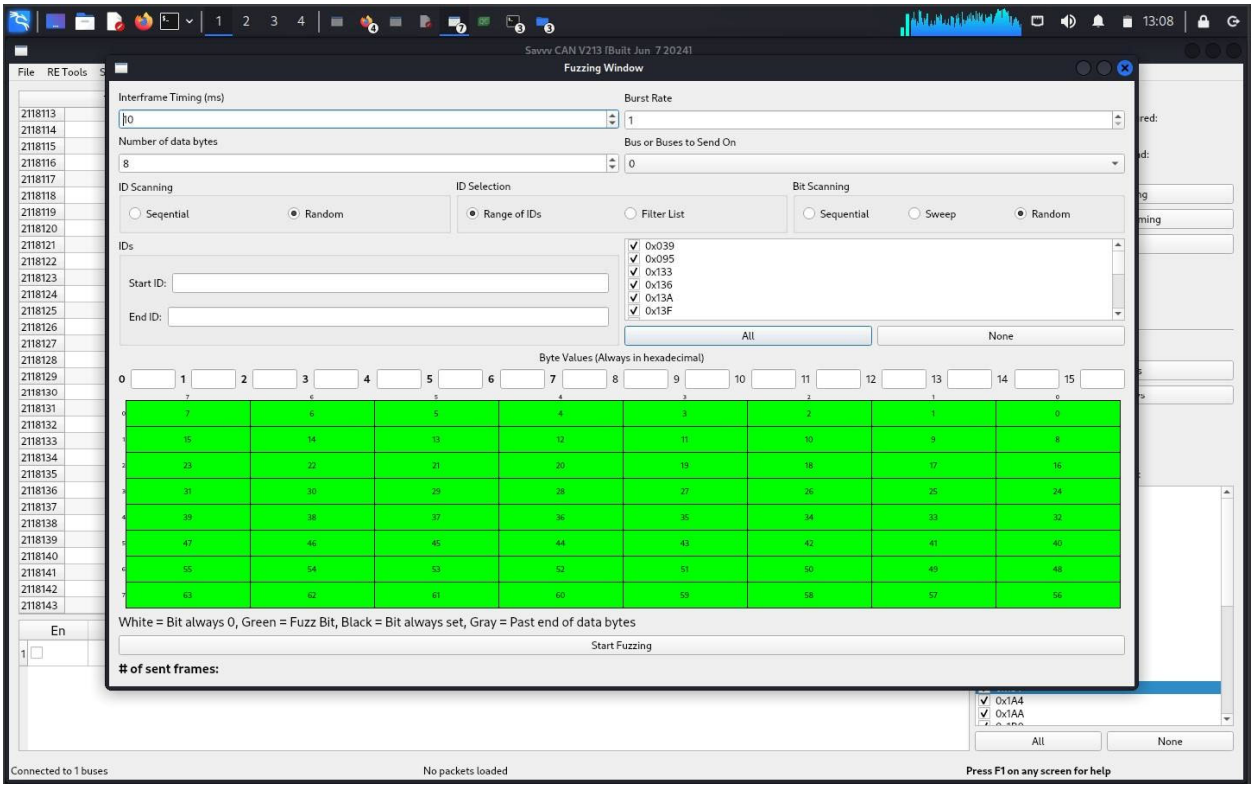
9.Fuzzing

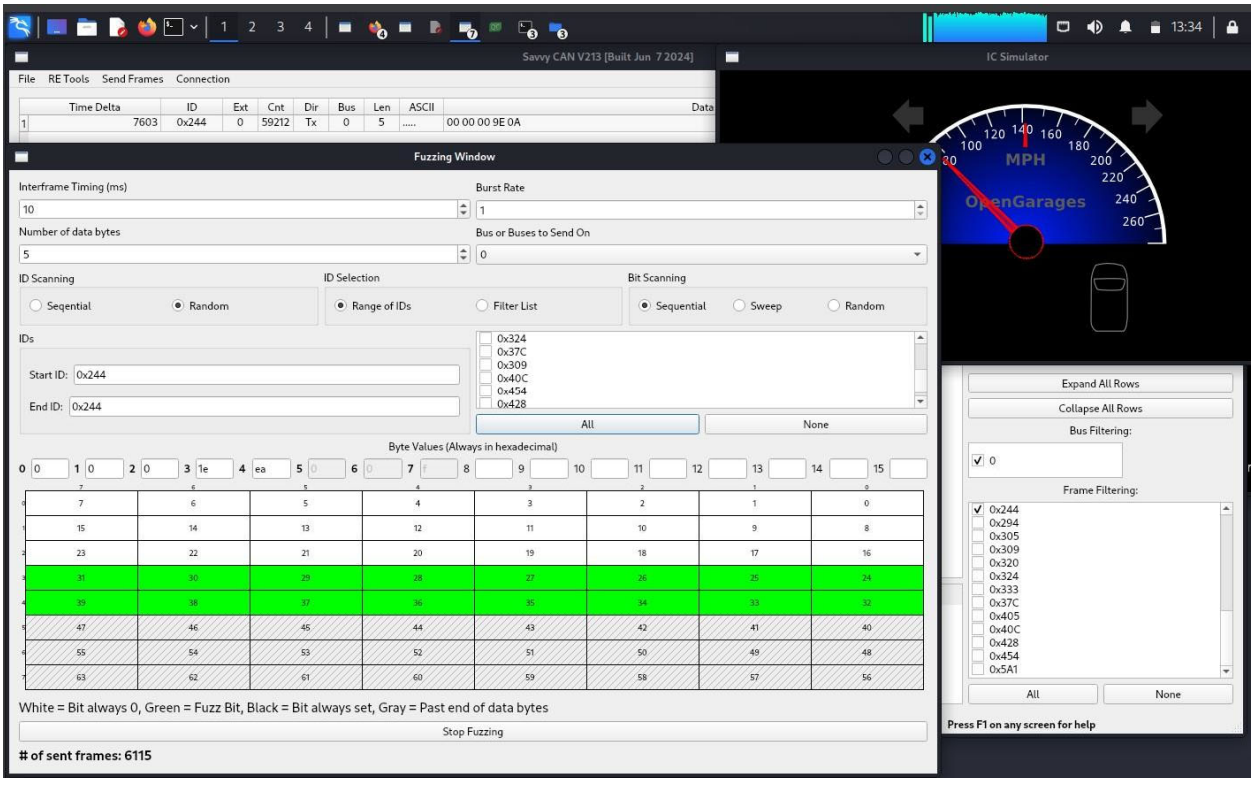
Fuzzing: This is a technique used in software testing where automated or semi-automated processes are used to send unexpected or malformed data (called "fuzz") to a target system in order to find vulnerabilities like buffer overflows, memory leaks, or other potential security issues.

Here in the screenshot ,
We can observe that 3rd and 4th rows are marked green this indicates that we are observing the change of bits only in the last 2 bytes .

Also we have considered only 5 number of data frames , hence we have considered only 5 rows from 0 to 4 .

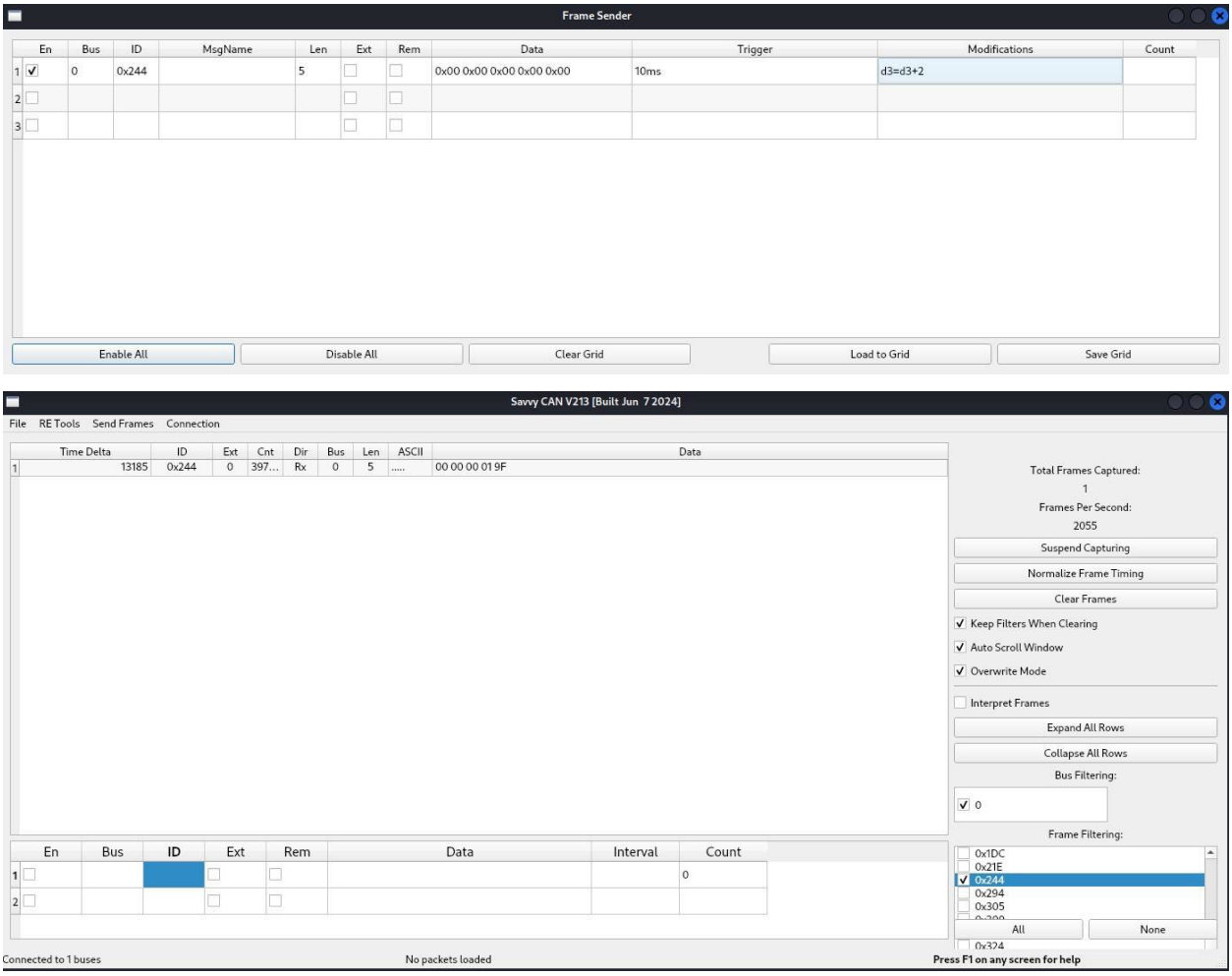
We have considered the 0x244 bit to indicate the actions being performed , the operation its performing here is “ Tachometer / accelerometer / speedometer “ .


Savvy CAN V213 [Built Jun 7 2024]
File RE Tools
Fuzzing Window
Interframe Timing (ms)
10
Burst Rate
1
Number of data bytes
8
Bus or Buses to Send On
0
ID Scanning
Seqential
Random
ID Selection
Range of IDs
Filter List
Bit Scanning
Sequential
Sweep
Random
IDs
Start ID:
End ID:
0x039
0x095
0x133
0x136
0x13A
0x13F
All
None
Byte Values (Always in hexadecimal)
White = Bit always 0, Green = Fuzz Bit, Black = Bit always set, Gray = Past end of data bytes
Start Fuzzing
# of sent frames:
0x1A4
0x1AA
All
None
Connected to 1 buses
No packets loaded
Press F1 on any screen for help


Savvy CAN V213 [Built Jun 7 2024]
File RE Tools Send Frames Connection
Time Delta ID Ext Cnt Dir Bus Len ASCII Data
7603 0x244 0 59212 Tx 0 5 ..... 00 00 00 9E 0A
IC Simulator
MPH
OpenGarages
Fuzzing Window
Interframe Timing (ms)
10
Burst Rate
1
Number of data bytes
5
Bus or Buses to Send On
0
ID Scanning
Seqential
Random
ID Selection
Range of IDs
Filter List
Bit Scanning
Sequential
Sweep
Random
IDs
Start ID: 0x244
End ID: 0x244
0x324
0x37C
0x309
0x40C
0x454
0x428
All
None
Byte Values (Always in hexadecimal)
White = Bit always 0, Green = Fuzz Bit, Black = Bit always set, Gray = Past end of data bytes
Stop Fuzzing
# of sent frames: 6115
Expand All Rows
Collapse All Rows
Bus Filtering:
0
Frame Filtering:
0x244
0x294
0x305
0x309
0x320
0x324
0x333
0x37C
0x405
0x40C
0x428
0x454
0x5A1
All
None
Press F1 on any screen for help

10.Sending Custom frames

Example:- here we want to send a Bus 0, ID 0x244 (tachometer), data as 0x00 0x00 0x00 0x00 0x00 and increase the 3rd byte by 2 every time, so on modification you could easily write d3=d3+2. Once done, make sure to check on Enable (EN) checkbox.



Trigger here , this indicates the number of frames or the action that its performing is delayed by 10 milliseconds .

SavvyCAN's Frame Sender feature allows users to modify packets in real-time while sending custom frames. For instance, we can manipulate the tachometer by altering specific bytes in the frame. By observing how the tachometer responds to changes in certain bytes, such as the 3rd and 4th bytes increasing with throttle, we can send custom frames and dynamically adjust these bytes

to observe the corresponding changes in the tachometer reading. This involves specifying hex values in the data column, formatting the ID as 0x123, setting the trigger value in milliseconds, defining the delay between frames, and detailing the modifications on bytes.

CONCLUSION :-

The project aims to strengthen automotive Controller Area Network (CAN) systems through comprehensive fuzzing with SavvyCAN on ICSim. Methods include injecting malformed frames and dynamically modifying data to uncover vulnerabilities. Objectives include proactive testing, vulnerability identification, and proposing mitigation strategies. The goal is to enhance automotive cybersecurity and pave the way for future advancements, promising immediate improvements and laying the groundwork for evolving CAN security testing methodologies.