

Jessica Sánchez Montané

ENTORN PER AVALUAR LA SEGURETAT D'UN CAN BUS

TREBALL DE FI DE GRAU

dirigit per Jordi Castellà Roca

Grau d'Enginyeria Informàtica



UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2019-2020

Resum.

El sector automobilístic està immers en grans avenços com són els cotxes autònoms i els vehicles connectats. Aquests automòbils tenen moltes funcionalitats que ens proporcionen una conducció més segura, eficient i còmoda. Tanmateix, per gaudir d'aquests avantatges és necessari connectar els vehicles amb dispositius externs o infraestructures remotes. Aquestes connexions fan possible la realització d'atacs remots de forma similar als que es produeixen en sistemes informàtics. Donades les conseqüències que es poden derivar s'han fet estudis de seguretat. Un dels punts més febles que s'han identificat és el CAN bus. Aquest protocol es va dissenyar per ser eficient, fàcil i econòmic però no es va tenir en compte la seguretat.

Per estudiar la seguretat i millores en el CAN bus s'ha dissenyat i implementat un CAN bus amb diferents mòduls que simulen diferents parts del cotxe. Aquests mòduls envien diferents tipus de missatges i realitzen certes accions depenent de la informació que reben. També s'ha introduït un mòdul que realitza diferents atacs suplantant altres terminals de l'entorn. Per finalitzar, s'ha implementat un mòdul que fa les funcions de IDS mitjançant Snort. Aquest analitza els paquets del CAN bus i alerta en cas de detectar un atac. Aquest mòdul també implementa l'algoritme k-means per poder identificar possibles atacs i extreure informació que permetrà definir noves regles pel IDS.

L'entorn obtingut és plenament funcional, modular i configurable. Es poden enviar missatges entre tots els mòduls de forma síncrona i asíncrona. La configuració de cada mòdul és flexible, és a dir, afegir o eliminar mòdul és molt senzill. El IDS (Snort) és capaç de llançar una alarma quan s'està realitzant un atac. A més a més, el k-means aconsegueix informació dels atacs que posteriorment permet definir noves regles a Snort.

Resumen.

El sector del automóvil está inmerso en grandes avances como son los coches autónomos y los coches conectados. Estos automóviles tienen muchas funcionalidades que nos brindan una conducción más segura, eficiente y cómoda. Sin embargo, para poder disfrutar de estas ventajas es necesario conectar los vehículos con dispositivos externos o infraestructura remotas. Estas conexiones hacen posible la realización de ataques remotos de forma similar a los que se producen en sistemas informáticos. Dadas las consecuencias que se pueden derivar se han hecho estudios de seguridad. Uno de los puntos más débiles que se han identificado es el CAN bus. Este protocolo fue diseñado para ser eficiente, rápido y económico pero no se tuvo en cuenta la seguridad.

Para estudiar la seguridad y mejoras en el CAN bus se ha diseñado e implementado un CAN bus con diferentes módulos que simulan diferentes partes del coche, Estos módulos envían diferentes tipos de mensajes y realizan ciertas acciones dependiendo de la información que reciben. También se ha introducido un módulo que realiza diferentes ataques suplantando a otros terminales. Finalmente se ha implementado un modulo que hace las funciones de IDS mediante Snort. Este analiza los paquetes del CAN bus y alerta en caso de detectar un ataque. Este moduló también implementa el algoritmo k-means para poder identificar posibles ataques y obtener información que permitirá definir nuevas reglas para el IDS.

El entorno de pruebas obtenido es plenamente funcional, modular y configurable. Se pueden enviar mensajes entre todos los módulos de forma síncrona y asíncrona. La configuración de cada módulo es flexible, es decir, añadir o eliminar un módulo es muy sencillo. El IDS es capaz de lanzar una alarma cuando se está realizando un ataque. Además, el k-means consigue la información de los ataques que posteriormente permite definir nuevas reglas en el Snort.

Abstract.

The automotive sector is seeing great advances such as the development of autonomous and connected cars. These automobiles have many functionalities that make our driving more efficient, convenient, and safer. However, in order to obtain such benefits, connections between the vehicle and external devices or remote infrastructures are necessary. These connections allow for remote attacks similar to those that threaten computer systems. A study of security has been done based on the possible risks associated. Following the study, one of the weakest nodes identified was the CAN bus. Such a protocol was designed to be efficient, cheap, and easy to use; however, security hasn't been taken into account.

In order to study the security issues and possible improvements of a CAN bus, a sample one was designed to fit a series of modules simulating different parts of a car. These modules send different types of messages and react depending on the input they receive. In addition, a module that impersonates other terminals of the environment was installed. Finally, we added a module that plays the part of IDS through the use of Snort—it analyzes the packets from the CAN bus and sends an alert when an attack is detected. This module also features the k-means algorithm to identify possible attacks and to obtain information that will then allow to define new rules for the Snort.

The resulting environment is completely functional, modular and customizable. It can send messages between all modules in synchronous or asynchronous form. The configuration of each module is flexible, meaning that adding or deleting modules is simple. The IDS (Snort) is capable of sending an alarm whenever an attack is detected. In addition, the k-means algorithm collects information about the attack, which then can be used to reconfigure the Snort.

Índex

| | | |
|----------|--|----|
| 1. | Introducció | 9 |
| 1.1. | Objectius | 10 |
| 1.2. | Motivació | 10 |
| 1.3. | Estructura de la memòria | 11 |
| 2. | Background | 12 |
| 2.1. | ECU | 12 |
| 2.2. | CAN bus | 12 |
| 2.2.1. | <i>Història</i> | 12 |
| 2.2.2. | <i>Funcionament</i> | 12 |
| 2.2.3. | <i>Format dels missatges</i> | 13 |
| 2.3. | Vulnerabilitats del CAN bus | 14 |
| 2.4. | Port OBDII | 14 |
| 2.5. | IDS | 14 |
| 2.5.1. | <i>Snort</i> | 15 |
| 2.6. | K-means | 15 |
| 3. | Arquitectura i disseny | 16 |
| 3.1. | Arquitectura i funcions | 16 |
| 3.1.1. | <i>Requisits</i> | 17 |
| 3.2. | Disseny | 17 |
| 3.2.1. | <i>Hardware</i> | 18 |
| 3.2.1.1. | <i>Raspberry Pi 3 B+</i> | 18 |
| 3.2.1.2. | <i>PiCAN2</i> | 18 |
| 3.2.1.3. | <i>Portàtil</i> | 19 |
| 3.2.1.4. | <i>Components auxiliars</i> | 19 |
| 3.2.1.5. | <i>Cables</i> | 19 |
| 3.2.2. | <i>Software</i> | 20 |
| 3.2.2.1. | <i>SocketCan</i> | 20 |
| 3.2.2.2. | <i>PuTTY i Xming</i> | 20 |
| 3.2.2.3. | <i>Canelloni</i> | 20 |
| 4. | Implementació | 21 |
| 4.1. | Preparació de l'entorn | 21 |
| 4.1.1. | <i>Modificacions físiques dels mòduls PiCAN2</i> | 21 |
| 4.1.2. | <i>Configuració Software</i> | 22 |
| 4.1.3. | <i>Preparació física del CAN bus</i> | 25 |

| | | |
|----------|---|----|
| 4.2. | Programa CAN bus | 26 |
| 4.2.1. | <i>Classe MyListener</i> | 26 |
| 4.2.2. | <i>Classe StopThread</i> | 26 |
| 4.2.3. | <i>Part principal</i> | 26 |
| 4.2.4. | <i>Parts específiques terminals legítims</i> | 27 |
| 4.2.5. | <i>Part específica mòdul atacant</i> | 27 |
| 4.3. | Programació de Serveis | 29 |
| 4.4. | IDS | 31 |
| 4.4.1. | <i>Connexió de l'ordinador amb el CAN bus</i> | 31 |
| 4.4.2. | <i>Implantació IDS</i> | 33 |
| 4.4.2.1. | <i>Snort</i> | 33 |
| 4.4.2.2. | <i>Kmeans</i> | 35 |
| 5. | Avaluació | 37 |
| 6. | Conclusions | 45 |
| 6.1. | Treball futur | 46 |

Índex de taules

| | |
|---|----|
| Taula 1. Línies fitxer /boot/config.txt | 22 |
| Taula 2. Codi font inicialize.sh | 22 |
| Taula 3. Comanda d'inicialització d'interfície CAN | 22 |
| Taula 4. Comanda d'instal·lació python-can. | 22 |
| Taula 5. Instal·lació llibreria SocketCan | 23 |
| Taula 6. Línies fitxer /etc/network/interfaces | 24 |
| Taula 7. Comanda canviar IP. | 24 |
| Taula 8. Codi convertir ordinador en router. | 25 |
| Taula 9. Script d'inicialització ECUI.sh | 29 |
| Taula 10. Canvi de localització i permisos ECUI.sh | 29 |
| Taula 11. Fitxer myservice.service | 30 |
| Taula 12. Comandes inicialitzar serveis. | 30 |
| Taula 13. Comanda instal·lació Canelloni | 32 |
| Taula 14. Instal·lació interfície can virtual | 32 |
| Taula 15. Comanda restricció vcan0 | 32 |
| Taula 16. Habilitar Canelloni ECUII | 32 |
| Taula 17. Habilitar Cannelloni mòdul IDS | 33 |
| Taula 18. Comanda instal·lació snort | 33 |
| Taula 19. Comanda executar Snort | 34 |
| Taula 20. Comanda mostrar dades CAN bus | 37 |

Índex de figures

| | |
|--|----|
| Figura 1. Trama CAN en format estàndard | 13 |
| Figura 2. K-means | 15 |
| Figura 3. Esquema arquitectura | 17 |
| Figura 4. Raspberry Pi3 B+ | 18 |
| Figura 5. Mòdul PiCAN2 | 18 |
| Figura 6. ECU I | 21 |
| Figura 7. ECU II | 21 |
| Figura 8. Xarxa IP entorn de proves | 23 |
| Figura 9. Connexió ECUI i ECUIV | 25 |
| Figura 10. CAN bus connectat | 25 |
| Figura 11. Entorn de proves amb mòdul IDS | 31 |
| Figura 12. Contingut missatges CAN en UDP | 33 |
| Figura 13. Norma Snort | 34 |
| Figura 14. Canvis visuals ECUIV | 37 |
| Figura 15. Candump ECUI | 38 |
| Figura 16. Candump ECUII | 38 |
| Figura 17. Candump ECUII amb botó ECUI | 39 |
| Figura 18. Candump ECUIV botó ECUIII | 39 |
| Figura 19. Candump ECUIV botó ECUIII i ECUI | 39 |
| Figura 20. Candump ECUII amb ECUIII i ECUIV | 40 |
| Figura 21. Ping entre ECUII i mòdul IDS | 40 |
| Figura 22. Captura Wireshark | 41 |
| Figura 23. K-means amb atac | 42 |
| Figura 24. K-means sense atac | 43 |
| Figura 25. Avís Snort | 44 |

1. Introducció

El sector automobilístic està en una època de canvis disruptius amb la creació i desenvolupament del cotxe elèctric, dels vehicles connectats i dels cotxes autònoms [1], [2]. Alguns autors afirmen que aquesta evolució en l'àmbit de l'automòbil comporta moltes millores que ajuden al conductor i a la societat en general. Algunes millores que es poden destacar són les següents [1], [4]:

- **Seguretat:** Al voltant del 94% dels accidents de trànsit són causats per errors humans. Una de les raons és la pèrdua d'atenció de les persones [1]. En els cotxes autònoms aquest problema disminueix, ja que, les màquines són precises i no tenen pèrdua d'atenció, a més poden rebre dades de l'entorn i avaluar-les. També poden prendre certes accions per intentar evitar certs perills, com per exemple el fet de reduir la velocitat si plou massa.
- **Eficiència:** Amb la capacitat computacional i la gran quantitat de dades que disposem en el món actual les màquines són capaces d'escollir diverses opcions per aconseguir una major eficiència en un gran ventall d'aspectes. Com per exemple: fer una conducció més eficient evitant un consum innecessari del combustible, impedir un desgast excessiu del motor i dels frens, etc.
- **Comoditat:** Des de fa temps un dels objectius principals de la tecnologia és, fer la vida de les persones més còmoda. Els cotxes actuals disposen d'una gran quantitat de funcions, com per exemple: claus remotes, connectar el mòbil al cotxe, control de creuer dinàmic, etc. Aquestes funcions permeten a les persones gaudir més del temps que passen en el vehicle amb diferents accions, com per exemple: poder connectar-te a Spotify, programar la calefacció, obri les portes del cotxe quan et trobes al costat de forma automàtica, etc.

Per poder gaudir de tots els avantatges que ens poden donar els cotxes intel·ligents és necessari que el vehicle estigui connectat a dispositius externs per poder enviar les dades dels diferents sensors, també és necessari que certs dispositius com per exemple el mòbil es connectin al vehicle per enviar certa informació. Aquestes connexions entre el vehicle i l'exterior poden permetre accés remot no desitjat, comportant vulnerabilitats en la seguretat. Aquest fet ha consentit diferents atacs al llarg dels anys causant problemes de diferents tipus, com per exemple: pèrdua d'informació, denegació de serveis, accions que posen en risc la integritat humana, etc [5], [6].

Un dels punts més febles de l'estructura dels vehicles és el *Controller Area Network* (CAN) bus [8], [9], [10]. Això és a causa que aquest protocol va ser pensat per ser ràpid, eficient, i amb un baix cost de producció. És a dir, és una tecnologia que no estava pensada per ser segura la qual cosa la fa vulnerable a què es connecti al bus un terminal no desitjat i comenci a realitzar diferents atacs informàtics [8], [9], [10].

Diversos estudis afirmen que amb la quantitat de dades que seran necessàries en els nous vehicles, l'amplada de banda i la longitud dels missatges que es poden enviar per un CAN bus no seran suficient per poder garantir una comunicació fiable i a temps reals [11], [12]. Per aquest fet es planteja que s'utilitzarà un altre tipus de tecnologia com per exemple comunicació mitjançant cables *Ethernet* o fins i tot de fibra òptica. Tanmateix, sàpiguen els cost d'aquestes tecnologies i la seva dificultat per ser escalable en comparació als CAN bus, es preveu que es continuarà utilitzant el protocol CAN en aquelles parts dels vehicles on el cost de producció és important i no es necessita molta amplada de banda [11].

1.1. Objectius

Els objectius del treball són els següents:

Entorn de proves

- Dissenyar i implementar un entorn de proves que implementi un CAN bus amb diferents components electrònics els quals fan la funció de diferents Electronic Control Units (ECU's) què es comuniquen entre si. Aquestes comunicacions contenen diferents missatges, és a dir, es permetrà especificar les dades, prioritats, mètodes d'enviament, etc. Els dispositius també reaccionen als diferents missatges que s'envien pel CAN bus i realitzen certes accions.

Mòdul Atacant

- Dissenyar i implementar un mòdul d'atacant. Aquest mòdul envia missatges fent-se passar per un altre terminal i d'aquesta manera realitza diferents atacs. Els atacs realitzats s'observen i s'analitzen.

Mòdul IDS

- Dissenyar i implementar un mòdul IDS, aquest rep les dades que es van enviant pel CAN bus. Els missatges són analitzats per un IDS amb l'objectiu de detectar si s'estan produint atacs, un cop detectada l'amenaça s'informa a temps reals. El mòdul també executa l'algoritme k-means amb la finalitat d'identificar els atacs i extreure informació que s'utilitza per actualitzar el IDS.

1.2. Motivació

Com s'ha explicat anteriorment els cotxes actuals tenen moltes funcionalitats i per poder-les realitzar els automòbils estan connectats a multitud de dispositius externs, també és necessari que diferents components electrònics es puguin connectar als vehicles. Aquesta tendència es preveu que s'incrementarà en els pròxims anys. Tanmateix, tots aquests avenços obren la possibilitat de rebre atacs, els quals poden tindre greus conseqüències, ja que, es poden trobar persones dintre dels vehicles atacats.

Una de les principals motivacions per aquest treball és que, com s'ha demostrat que és molt difícil evitar els atacs, és important detectar-los. La introducció d'un IDS en els vehicles es preveu com una eina necessària per poder detectar atacs en els vehicles i permetre realitzar certes contramesures, com per exemple: avisar a la companyia, a les persones, etc.

Una altra motivació és la necessitat d'implementar un IDS que pugui interactuar amb diferents xarxes. Com s'ha explicat a la Introducció es preveu que en les arquitectures que es realitzaran en el futur, el protocol CAN conviurà amb altres protocols de comunicació dins d'un automòbil. Per aquest motiu, és necessària, la implementació d'un sistema que tingui la capacitat de detectar atacs en diferents tipus de xarxa.

Per últim cal esmentar que es va voler realitzar aquest projecte perquè el desenvolupament de l'entorn de proves proposat i la implantació d'un IDS en el sistema és completament viable i pot generar interès en el sector automobilístic, no només pels resultats obtinguts, sinó també per futures investigacions que es podrien realitzar partint de l'entorn.

1.3. Estructura de la memòria

La Secció 2 conté una breu descripció dels conceptes teòrics i de les tecnologies emprades al treball. Com poden ser els elements dels vehicles, el protocol CAN, el k-means, etc.

L'Arquitectura i el disseny del sistema proposat s'introdueix a la Secció 3. Aquesta inclou els actors implementats i les funcions que realitzen cadascun. A més, també es descriuen els requisits no funcionals de l'entorn, el disseny realitzat i l'explicació de les decisions del hardware i software escollit.

La Secció 4 detalla la implementació del treball, és a dir, els passos necessaris per construir l'entorn físicament, l'aclariment del codi programat i les configuracions que s'han hagut de realitzar.

L'Avaluació del correcte funcionament del sistema amb les seves funcionalitats està a la Secció 5. Finalment es presenten les conclusions i treball futur a la Secció 6.

2. Background

En aquest apartat es descriuen els conceptes previs necessaris per poder entendre el treball de forma correcta.

2.1. ECU

Les ECU's són els dispositius responsables de controlar, regular i alternar les operacions dels sistemes electrònics dels vehicles. Controlen gran part de les característiques del cotxe com per exemple, el sistema de frenat i el d'injecció de combustible [13].

Hi ha diferents tipus d'ECU [14]:

- *Door Control Unit* (DCU): Controlen i monitoren diferents accessoris electrònics de la porta del vehicle.
- ECU: No s'ha de confondre amb el terme general, s'anomenen *Electronic Control Module* (ECM). Controlen actuadors que treballen en la combustió interna del vehicle, amb la finalitat de garantir el funcionament òptim del motor.
- *Human-machine Interface* (HMI): Controlen les interaccions entre la màquina i l'humà.
- *Powertrain Control Module* (PCM): Controlen el motor del cotxe, generalment combina l'ECM i el TCU.
- *Speed Control Unit* (SCU): Controlen de forma automàtica la velocitat del turisme, comunament és conegut com a control de creuer.
- *Transmission Control Unit* (TCU): Controlen les transmissions automàtiques.
- *Battery Management System* (BMS): Qualsevol dispositiu que controla el funcionament de les bateries.

2.2. CAN bus

El CAN és un protocol de comunicació que no necessita un ordinador que faci de host, utilitzat a la indústria de l'automòbil. Els cotxes actuals estan plens de petits sistemes i ECU's que es comuniquen entre si mitjançant el protocol CAN [16].

2.2.1. Història

El desenvolupament del protocol CAN bus va començar l'any 1983 per l'empresa Robert Bosch GmbH. Tot i que, va ser llançat oficialment l'any 1986 en el congrés de la societat d'enginyeria automobilística, organitzat en Detroit, Michigan. Els primers controladors CAN van ser produïts per Intel i Philips i van sortir al mercat l'any 1987. La producció del primer cotxe basat en el CAN bus amb múltiples cables va ser l'any 1991 per part de l'empresa Mercedes-Benz.

Bosch va llançar diferents versions del protocol CAN, la versió utilitzada actualment és de l'any 1991 i s'anomena CAN2.0. Hi ha dos tipus de versions del CAN2.0. El CAN2.0A que és el format estàndard, utilitza 11 bits d'identificador i l'altra versió és el CAN2.0B que té 29 bits d'identificador, també s'anomena format extens.

2.2.2. Funcionament

El CAN bus és un sistema que utilitza dos cables: el CAN *High* (CANH) i el CAN *Low* (CANL). Mentre el bus es troba en un estat de repòs les dues línies es mantenen a un voltatge de 2,5V. Quan una dada és enviada, el CANH augmenta el seu voltatge en 1,25V, mentre que el CANL

disminueix el seu voltatge en 1,25 V, fent així que entre els dos cables hi hagi una diferència de 2,5V, aquest sistema té com a finalitat evitar errors de transmissió a causa del soroll. En el cas que en rebre un paquet si aquesta propietat no es compleixi, el paquet és refusat.

Amb la finalitat de poder terminar els dos cables del CAN bus. En els dos extrems del CAN bus es troba una resistència de 120-ohms.

2.2.3. Format dels missatges

El CAN bus és un bus on es troben els terminals connectats, tots els components poden enviar missatge i tota la resta de terminals reben tots els missatges enviats. A l'estil de l'*User Datagram Protocol* (UDP) *broadcast* en la xarxa d'*Ethernet*. Per tant, és trivial per un dispositiu fer-se passar per un altre, ja que tots poden escoltar i enviar qualsevol missatge.

A la Figura 1 es pot veure el format estàndard dels paquets CAN, és un format basat en quatre components principals:

- **Identificador:** Identifica els diferents tipus de dispositius que hi ha sobre el CAN bus i la seva prioritat, ja que si dos terminals volen enviar dos missatges alhora, s'enviarà primer el missatge que té l'identificador més alt. Els dispositius utilitzen aquest camp amb la finalitat de saber si el missatge els interessa o el poden descartar.
- **Bit d'extensió d'identificador:** Bit que s'utilitza per identificar si el paquet és del format estàndard o del format estès. Si el format es troba a 0, és estàndard i si es troba a 1 és estès.
- **Codi de longitud de dades:** Indica de quina mida són les dades, té un rang d'entre 0 i 8 bytes.
- **Dades:** Dades que es volen transmetre. El màxim nombre de dades que és es poden transmetre en el format estàndard és de 8 bytes. Tot i que alguns models apliquen el *padding* perquè tots els missatges tinguin com a mida 8 bytes.

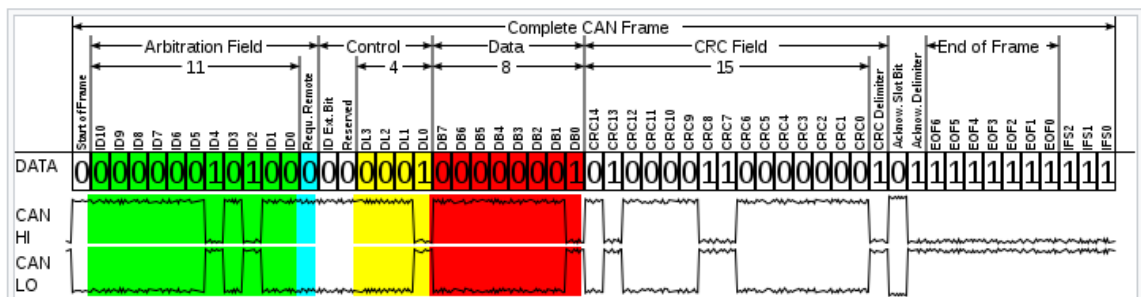


Figura 1. Trama CAN en format estàndard

Existeixen quatre tipus de missatges diferents que s'envien a través del CAN bus:

- **Trama de dades:** S'envia la informació habitual en el format explicat anteriorment.
- **Trama remota:** Tipus de trames que són enviades quan un node necessita unes dades d'un altre node. La seva estructura es diferencia de les trames de dades perquè no tenen camp de dades i tenen el bit *Request Transmission Remote* (RTR) a 1.

- Trama d'error: No segueix el format de trames explicat anteriorment. Quan un terminal detecta un error envia trames d'errors i fa que la resta de nodes també enviïn missatges d'error. Un complex mecanisme evita que un terminal faci una denegació de servei inundant la xarxa de trames d'error.
- Trama de sobre càrrega: Igual que la trama d'error tampoc utilitza el format explicat. Quan un node té massa treball, envia una trama de sobre càrrega i llavors el CAN bus augmenta el temps d'espera entre trames.

2.3. Vulnerabilitats del CAN bus

Com s'ha explicat a la Introducció el CAN bus tot i ser un sistema molt robust i eficient per transferir dades entre els diferents nodes del vehicle, diversos estudis han demostrat que és una tecnologia amb grans vulnerabilitats. Antigament aquestes vulnerabilitats eren més difícils d'explotar, ja que era necessària una connexió física en bus. No obstant els cotxes moderns tenen altres tipus de connexions, com per exemple: Bluetooth, Wifi, GPS, etc. Aquests components faciliten la introducció de terminals maliciosos al sistema, la qual cosa fa que es necessitin prendre diferents mesures de seguretat[17]. Algunes de les vulnerabilitats més importants del protocol són les següents:

- Missatges *multicast*: Tots els missatges que s'envien són escoltats per tots els terminals. La qual cosa fa que l'atacant a més de veure els missatges de forma passiva pugui realitzar enginyeria inversa i pugui saber en quin format injectar els paquets per realitzar l'atac.
- Falta d'autenticació: Un missatge enviat pel CAN bus no es pot saber qui l'ha enviat, aquest fet facilita realitzar un atac mascarada. Aquest atac es produeix quan un individu intentar suplantar la identitat d'un altre.
- Punt d'entrada comú: Un cop connectar un terminal en el bus, pots comunicar-te a qualsevol dels terminals del CAN bus.
- Amplada de banda limitat: El fet d'una amplada de banda petita comporta que algorismes complexos de seguretat no poden ser utilitzats.
- Múltiples sistemes integrats: En el CAN bus estan connectats diferents terminals molt diversos entre si. Aquesta propietat dificulta poder incorporar un sistema de seguretat senzill per tots els sistemes.

2.4. Port OBDII

És un terminal que serveix per comunicar les diferents ECU's i components electrònics del vehicle amb un ordinador o amb qualsevol dispositiu que tingui una eina de diagnòstic per OBD-II. Aquesta comunicació s'utilitza per poder detectar qualsevol error en el vehicle i monitora el funcionament de certes funcions, com per exemple poder controlar l'emissió dels gasos que realitza el vehicle. En la majoria de cotxes actual el port OBD II és el mètode utilitzat per connectar-se amb els CAN busses que es troben dins dels vehicles.

2.5. IDS

El IDS és un dispositiu o una aplicació que analitza el tràfic que es va enviant a través d'una xarxa, per detectar possibles atacs, quan detecta un possible atac pot realitzar diferents accions, com per exemple llençar una alarma. La formes d'esbrinar si un missatge és enviat per un atacant o és un

missatge habitual de la xarxa són molt diverses, es poden basar en l'identificador dels missatges (en el cas que el tinguin), patrons en les dades dels paquets, etc.

2.5.1. Snort

El Snort és un *Network Based Intrusion Detection System* (NIDS) de codi obert escrit en llenguatge C. Aquesta eina analitza els paquets d'una xarxa IP amb la finalitat de poder detectar una càrrega útil perillosa o anomalies sospitoses. El Snort per saber si un paquet és sospitós es basa en un sistema de regles internes. Aquestes regles especifiquen certes accions a realitzar amb els paquets que compleixen certes condicions, les regles es poden eliminar, modificar, habilitar i fins i tot crear-ne de noves depenent de les necessitats de cada cas.

2.6. K-means

El k-Means clustering és un algorisme de *Machine Learning* de la categoria d'aprenentatge no supervisat, és a dir, les dades d'entrada no estan etiquetades en una categoria, per les variables de sortida l'algorisme estudia les característiques de les dades i crea les seves pròpies classes.

Com el seu nom indica classifica les dades d'entrada en k clústers. Els *clústers* són grups de valors on les dades són molt similars dins del mateix clúster i molt diferents amb les dades d'un altre clúster [26].

Els fonaments de l'algorisme es basa en la reubicació dels centroides. Els centroides són els centres dels diferents clústers. Al començament se situen en una posició aleatòria, llavors les dades és categoritzant segons el centroide que tenen més pròxim, un cop les dades es terminen d'etiquetar el centroide es reubica al centre del clúster amb els valors que s'acaben de classificar. El procediment de reubicar els centroides i classificar les dades es va repetint de forma recursiva fins que s'arriba a una condició de parada, aquesta condició pot ser nombre d'iteracions, poca variació entre les classificacions de les dades, etc. En la Figura 2 es pot veure de forma gràfica un exemple de com van canviant la classificació de les dades i la reubicació dels centroides.

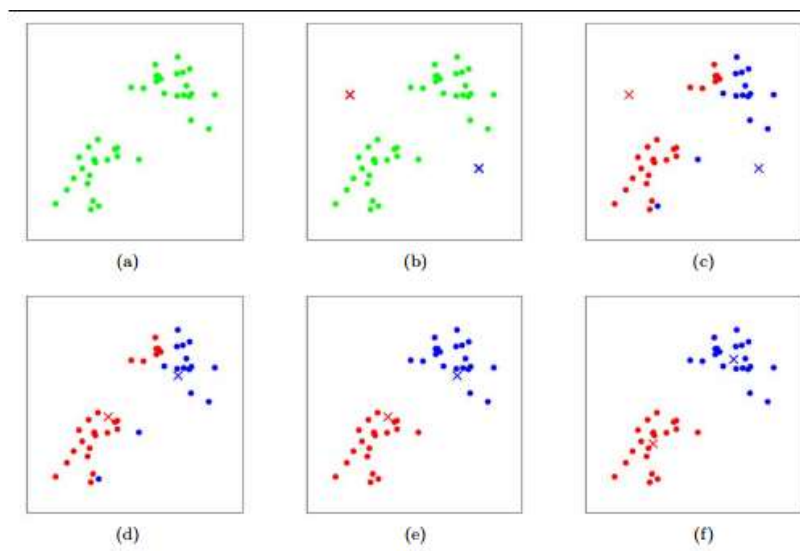


Figura 2. K-means

3. Arquitectura i disseny

A continuació es descriu l'arquitectura plantejada per satisfer els objectius que es volen assolir. S'explica quins actors es pretenen desenvolupar, quines accions es vol que realitzin cadascun, és a dir, el tipus de missatges que es desitgen enviar i quines reaccions als paquets rebuts s'espera que tinguin, també s'especifica com es pretén que els actors estiguin connectats entre si. A més, es descriu les propietats que són necessàries i/o desitjades que tingui l'entorn perquè compleixi les necessitats per les quals es va voler desenvolupar. Per últim, es redacta el disseny implementat i les decisions realitzades.

3.1. Arquitectura i funcions

L'arquitectura proposada tal com s'observa a la Figura 3 està formada per 4 terminals numerats del I al IV amb els noms d'ECUI, ECUII, etc. També es troba composta per un mòdul extern que té implementat un IDS, s'utilitza un l'ordinador. Com s'ha explicat anteriorment al Format dels missatges, els terminals d'un CAN bus envien missatges i tots els poden veure, però, normalment els missatges van destinats a components específics. L'objectiu és que els diferents terminals realitzin les següents accions:

- ECUI: Aquest terminal enviï missatges a l'ECUIV, es vol que enviï els paquets de forma asíncrona.
- ECUII: Rebi les dades de l'ECUII i que mostri de forma visual una aproximació dels valors de les dades enviades.
- ECUIV: Aquest terminal rebi els missatges de l'ECUI i que quan arribi un nou paquet s'indiqui de forma visual. A més, que enviï dades de forma continuada a l'ECUII cada cert període de temps.
- ECUIII: Aquest terminal simuli l'atacant. Es pretén que enviï missatges de forma asíncrona a l'ECUIV fent-se passar per l'ECUI. També que enviï dades errònies de forma periòdica cada cert període de temps a l'ECUII fent-se passar per l'ECUIV. Cal esmentar, que es desitja que enviï missatges amb més freqüència que el terminal legítim per evitar que els missatges de l'ECUIV puguin causar algun canvi. Amb la finalitat de tenir prioritat per enviar missatges pel CAN bus es pretén que els missatges de l'atacant tinguin un identificador major que els enviats pels altres terminals.
- Mòdul IDS: Aquest terminal es vol que sigui un ordinador connectat al bus mitjançant un cable OBD II, aquest mòdul es pretén que tingui instal·lat un IDS que analitzi tots els missatges del CAN bus i enviï un missatge d'avís per la pantalla de l'ordinador quan detecti un atac en el CAN bus. En aquest mòdul també es vol que s'executi l'algoritme k-means amb la finalitat d'identificar possibles atacs i extreure informació que permet actualitzar el IDS.

Tal com mostra la Figura 3 els diferents terminals també es poden entendre com simulacions de diferents parts dels vehicles. Aquestes parts poden ser per exemple els frens, el motor i el termòmetre.

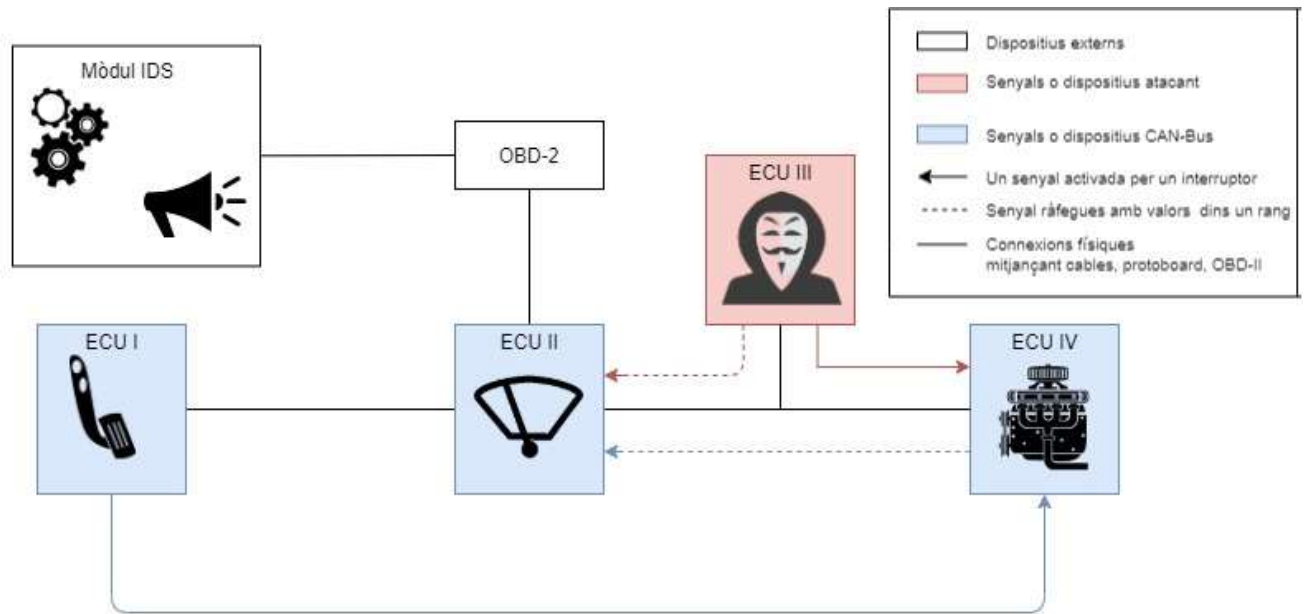


Figura 3. Esquema arquitectura

3.1.1. Requisits

Independentment de les funcions, es vol que aquesta arquitectura compleixi una sèrie de requisits, que tot i no ser necessaris per al correcte funcionament de l'entorn de proves, són aspectes importants a tenir en compte per millorar la qualitat i l'eficiència del projecte. Els requisits són els següents:

- Minimitzar les preparacions necessàries per habilitar l'entorn de proves un cop construït.
- Poder aplicar els coneixements adquirits en aquest entorn en un cotxe real.
- Fer que l'entorn sigui el més compacte i transportable possible.
- Poder afegir nous terminals al CAN bus de la forma més fàcil i ràpida possible.
- Poder afegir noves funcions o canviar les existents de forma fàcil i eficient.
- Aplicar diferents atacs.

3.2. Disseny

A partir de l'arquitectura i requisits exposats anteriorment en aquesta secció es mostra el procés de disseny. A més dels productes *Software* i *Hardware* escollits també es comentaren les altres opcions que s'han plantejat per realitzar les mateixes funcions i els motius per les quals han sigut descartades.

3.2.1. Hardware

Els components hardware que s'han utilitzat per realitzar l'arquitectura del CAN bus explicada anteriorment han sigut els següents:

3.2.1.1. Raspberry Pi 3 B+

Les Raspberry Pi són petits ordinadors funcionals que tenen una dimensió d'una petita placa electrònica. Aquest producte s'han tornat molt popular en els últims anys, ja que són molt petits, tenen bastant potència per poder executar diferents programes i també solen tindre un preu bastant reduït.

El model utilitzat és la Raspberry Pi 3 B+ que té els següents components principals:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- RAM: 1GB LPDDR2 SDRAM
- Wi-Fi + Bluetooth: 2.4GHz i 5GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.2, BLE
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)
- GPIO de 40 pins
- HDMI
- Ports USB 2.0
- Port CSI per connectar una càmera.
- Port DSI per connectar una pantalla tàctil
- Sortida d'àudio estèreo i vídeo compost
- Micro-SD
- Power-over-Ethernet (PoE)



Figura 4. Raspberry Pi3 B+

3.2.1.2. PiCAN2

És un mòdul per les Raspberry Pi, en concret per les Raspberry Pi3. Proporciona les eines necessàries per convertir una Raspberry Pi en un terminal d'un CAN bus. Utilitza un controlador CAN MCP2515 i un transceptor MCP2551. És possible connectar els cables del bus via DB9 o amb un terminal de 4 vies. Uns dels principals avantatges d'aquest dispositiu és que, és molt fàcil la instal·lació del *SocketCan*, a més, també dóna la possibilitat d'escriure el codi tant en C com en Python [19].



Figura 5. Mòdul PiCAN2

A més de la combinació de la Raspberry Pi i el PiCAN2 per obtenir els terminals del CAN bus en aquest entorn de proves també es va plantejar la possibilitat d'utilitzar una placa Arduino i connectar de forma manual un receptor i un transmissor CAN. Els motius de l'elecció van ser els següents:

- La instal·lació era la més simple i compacta. Això és degut al fet que, tant els transmissors com els receptors CAN van integrats en la PiCAN2. A més, el mòdul també conté un led i forats per poder afegir uns petits botons utilitzats en circuits electrònics. Aquests elements són necessaris per a poder enviar missatges en els moments desitjats i tindre la capacitat de poder visualitzar alguns canvis en els terminals en el moment que arriben missatges rellevants.
- Facilitat per poder afegir més mòduls de forma fàcil i sense la necessitat de fer gaires modificacions en l'entorn ja creat.
- La Raspberry Pi porta un sistema operatiu basat en Linux. És un entorn en el qual he treballat prèviament facilitant la programació, les instal·lacions dels softwares, etc. Aquest coneixement previ també facilita poder automatitzar els processos necessaris amb la finalitat de minimitzar les preparacions necessàries per habilitar l'entorn de proves un cop construït.
- El PiCAN2 permet una fàcil instal·lació del software necessari per enviar i rebre missatges del bus, el *SocketCan*.

3.2.1.3. Portàtil

En l'entorn de proves una de les funcionalitats del portàtil es ser utilitzat com a monitor, teclat i ratolí de la Raspberry Pi mitjançant una comunicació ssh amb un cable *Ethernet*.

Per poder utilitzar les Raspberry Pi, a més d'utilitzar l'ordinador es van plantejar les següents alternatives:

- Utilitzar un monitor, un teclat, i un ratolí i anar alternant-los entre les diferents Raspberry Pi.
- Utilitzar un monitor, un teclat i un ratolí per cadascuna de les Raspberry Pi.

Es va escollir l'opció ssh, ja que era la que menys espai comportava i permetia passar d'una Raspberry Pi a un altre únicament canviant la connexió del cable *Ethernet*.

El portàtil també és utilitzat per ser el mòdul IDS que s'ha explicat anteriorment a l'Arquitectura i funcions.

3.2.1.4. Components auxiliars

S'han utilitzat els següents components auxiliars amb la finalitat de poder completar l'entorn de proves:

- La placa *protoboard* s'ha utilitzat per poder connectar els cables del CAN bus creat.
- Els mini botons utilitzats en el mòdul PiCAN2.

3.2.1.5. Cables

En l'entorn de proves s'han necessitat els següents tipus de cables:

- Cables *protoboard* per connectar els mòduls, a la *protoboard* per poder connectar els dispositius entre ells, creant les connexions físiques del CAN bus.
- Cable *Ethernet* per connectar el portàtil amb les Raspberry Pi.

- Cable per realitzar un pont per utilitzar les resistències de 120 ohms dels mòduls PiCAN2.

3.2.2. *Software*

A continuació es realitza una breu descripció dels programes i llibreries auxiliars que s'han utilitzat per poder desenvolupar l'entorn de proves.

3.2.2.1. *SocketCan*

El SocketCan és una implementació del protocol CAN creada per Volkswagen Research pel Kernel de Linux. Es va crear amb l'objectiu de què la integració sigues el més similar possible a la dels protocols TCP¹/IP per tal que als programadors els hi sigui fàcil aprendre a treballar amb *sockets* CAN [20].

El mòdul PiCAN2 permet utilitzar llibreries que utilitzen el SocketCan tant per a C com per a Python. Es va escollir utilitzar Python, perquè per internet hi ha més informació, a més, en aquest entorn de proves el temps de resposta no és un aspecte crític. La llibreria utilitzada en Python és la python-can, aquesta llibreria a partir de la versió 3.3 utilitza de forma directa el software SocketCan.

3.2.2.2. *PuTTY i Xming*

Amb la finalitat de poder utilitzar l'ordinador per manipular la Raspberry Pi3, s'ha utilitzat el programa PuTTY per realitzar la connexió en ssh en Windows. També s'ha utilitzat el programa Xming que permet manipular un ordinador amb connexió ssh amb una interfície gràfica, aquesta funcionalitat facilita la manipulació de les Raspberry Pi 3, ja que, algunes accions es dificulten sense la presència d'una interfície gràfica com per exemple la programació en Python.

3.2.2.3. *Canelloni*

És una llibreria que permet convertir els missatges en protocol CAN a UDP i viceversa, un dels objectius d'aquesta transformació és poder comunicar dues màquines mitjançant una comunicació per una xarxa *Ethernet*. Un cop els missatges en UDP arriben al seu destí es tornen a convertir en dades amb el protocol CAN. En l'entorn de proves la transformació de les dades CAN a UDP s'utilitza perquè el Snort pugui analitzar el contingut dels paquets en cerca d'anomalies. La comunicació amb Canelloni entre les dues màquines pot tindre pèrdua de paquets i tampoc hi ha garantia que els paquets arribin en l'ordre correcte, per tant, és una eina vàlida per un entorn de proves com el que es vol desenvolupar però no és recomanable per l'àmbit industrial [21].

¹ *Transmission Control Protocol*

4. Implementació

A continuació és mostra el procés d'implementació dels mòduls legítims, del mòdul de l'atacant i del mòdul IDS.

4.1. Preparació de l'entorn

Seguidament s'explicà detalladament els passos necessaris per habilitar l'entorn de proves per posteriorment poder afegir les funcionalitats a cadascun dels terminals.

4.1.1. Modificacions físiques dels mòduls PiCAN2

Primer de tot, s'afegeix a cadascun dels mòduls PiCAN2 els components necessaris per poder realitzar l'entorn de proves [19].

- Al mòdul PiCAN2 que s'utilitza a l'ECUI, s'afegeix, un petit boto a la part S1, a més se li realitza la soldadura en el component J3 per tal d'activar la resistència de 120 ohms. Aquesta resistència és necessària en els dos terminals que es troben en cadascun dels extrems del CAN bus.



Figura 6. ECU I

- Al mòdul de l'ECUII es fa un pont entre els punts de la part OBDII necessaris per poder usar el connector OBDII mitjançant el connector DB9 com és mostra a la Figura 7.

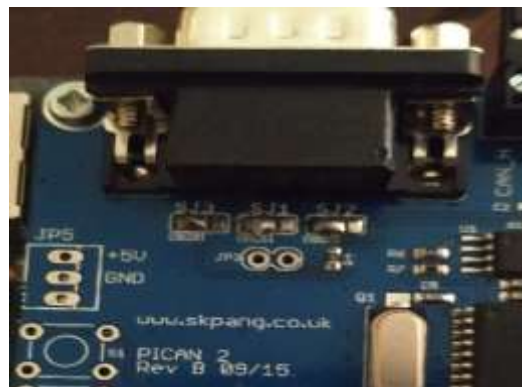


Figura 7. ECU II

- Al mòdul de la placa de l'atacant s'incorporen dos botons, un en el component S1 i un altre en el component S2.
- Al mòdul de l'ECU IV també se l'ha hagut d'afegir la soldadura per activar la resistència de 120 ohms tal com és mostra a la Figura 6 i s'explica anteriorment.

4.1.2. Configuració Software

Després de realitzar les modificacions pertinents a les plaques PiCAN2, es realitzen les instal·lacions per poder habilitar la interfície CAN en la Raspberry Pi i per poder utilitzar les llibreries SocketCan i python-can [19].

Per poder habilitar la interfície CAN, cal afegir les línies que es poden veure en la Taula 1 al final del fitxer /boot/config.txt:

| |
|---|
| dtparam=spi=on |
| dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25 |
| dtoverlay=spi-bcm2835-overlay |

Taula 1. Línies fitxer /boot/config.txt

Amb la finalitat d'agilitzar el procés en tots els terminals es va crear el script amb el codi observat en la Taula 2.

| |
|--|
| #!/bin/bash |
| echo "dtparam=spi=on" >> /boot/config.txt |
| echo "dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25" >> /boot/config.txt |
| echo "dtoverlay=spi-bcm2835-overlay" >> /boot/config.txt |

Taula 2. Codi font inicialize.shTaula

Cada vegada que es vol utilitzar la interfície del CAN bus, és necessari habilitar la interfície. S'ha de posar en el terminal la següent comanda.

| |
|--|
| sudo /sbin/ip link set can0 up type can bitrate 500000 |
|--|

Taula 3. Comanda d'inicialització d'interfície CAN

Després de configurar la interfície CAN amb el nom de can0, s'instal·la la llibreria de python-can. Primer de tot, es descarrega el repositori de la llibreria [22], després es descomprimeix la carpeta. Finalment per instal·lar-la s'ha d'executar la comanda següent:

| |
|-------------------------------|
| sudo python3 setup.py install |
|-------------------------------|

Taula 4. Comanda d'instal·lació python-can.

Per poder utilitzar les comandes de la llibreria SocketCan, és necessari executar la comanda d'instal·lació que es pot veure en la Taula 5.

```
sudo apt-install can-utils
```

Taula 5. Instal·lació llibreria SocketCan

Un cop finalitzades les instal·lacions necessàries per poder rebre i enviar missatges per un CAN bus, s'assigna una IP estàtica a totes les Raspberry Pi, amb la finalitat de poder crear scripts de configuració sense necessitat de comprovar les IP assignades cada vegada que s'encén l'entorn. Un altre objectiu és facilitar la connexió ssh a Ubuntu. Les IP assignades a cada component de l'arquitectura van ser:

- Mòdul IDS: 192.168.137.1
- ECUI: 192.168.137.211
- ECUII: 192.168.137.212
- ECUIII: 192.168.137.213
- ECUIV: 192.168.137.214

A la **Figura 5** és mostra quina estructura a nivell de IP té l'entorn de proves. El mòdul IDS es connecta a les ECU's de forma individual mitjançant un cable *Ethernet*. Les ECU's continuen comunicant-se mitjançant les connexions del CAN bus.

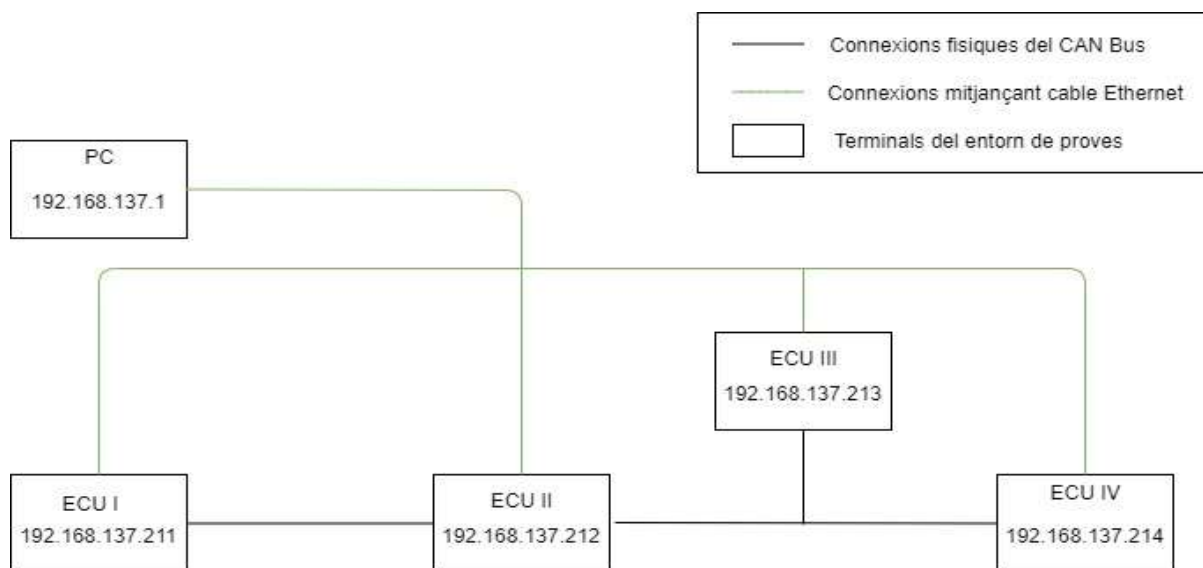


Figura 8. Xarxa IP entorn de proves

Per assignar a les Raspberry Pi les IP estàtiques de forma fixa es modifica el fitxer `/etc/network/interfaces` i s'afegeixen les línies següents:

| |
|---|
| <code>Auto eth0 /* Interfície de xarxa que està connectada a l'ordinador en el cas de les Raspberry Pi de l'entorn de proves eth0 */</code> |
| <code>iface eth0 inet static</code> |
| <code>address 192.168.137.211/* IP de l'ECU en el cas de l'ECUI 192.168.137.211 */</code> |
| <code>netmask 255.255.255.0</code> |
| <code>network 192.168.137.0 /* IP de la xarxa en el cas de l'entorn de proves 192.168.137.0 */</code> |
| <code>broadcast 192.168.137.255</code> |
| <code>gateway 192.168.137.1 /* IP del router en el cas de l'entorn de proves 192.168.137.1 */</code> |

Taula 6. Línies fitxer `/etc/network/interfaces`

Després de realitzar les modificacions en el fitxer de cadascuna de les ECU's en reiniciar els dispositius aquests sempre tindran la IP assignada. Seguidament es va configurar el mòdul IDS que s'utilitza també com a router per l'entorn de proves, les configuracions que es realitzen per cada sistema operatiu són les següents:

- En el cas de Windows, la IP de la interfície de xarxa de l'ordinador utilitzat sempre es manté en la 192.168.137.1. L'única configuració necessària és habilitar l'opció de xarxa compartida en la xarxa wifi a la qual s'està connectat en el moment de voler comunicar-se amb el CAN. En aquest moment, l'ordinador començarà a actuar com a router per l'entorn.
- En el cas de Linux la IP no coincideix amb les IP's de l'entorn de proves per tant, primer de tot, és necessari canviar-la. Com aquest ordinador no s'utilitza exclusivament per aquest entorn de proves es va decidir no canviar la IP de forma fixa. Per modificar la IP de forma temporal s'executa la comanda que es troba a la Taula 7 cada vegada que és necessari.

```
ifconfig eth0 192.168.137.1 netmask 255.255.255.0
```

Taula 7. Comanda canviar IP.

Seguidament és necessari, activa el mode *forwarding*. Per finalitzar, és necessari habilitar el tallafoc perquè realitzi NAT per tots els paquets que vinguin de la interfície de xarxa connectada a l'entorn de proves, en aquest cas és la interfície `wlp3s0`. En la Taula 8 es mostra les comandes necessàries per activar el mode *forwarding* i per realitzar les modificacions del tallafoc esmentades.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

```
iptables -A FORWARD -j ACCEPT
```



```
iptables -t nat -A POSTROUTING -s 192.168.137.0/24 -o
wlp3s0 -j MASQUERADE
```

Taula 8. Codi convertir ordinador en router.

Un cop finalitzades les instal·lacions necessàries, és fan unes petites proves de funcionament per comprovar que no hi ha cap component defectuós tant en les Raspberry com a les PiCAN 2. Amb aquest objectiu es programa un petit codi en Python on s'activa la interfície CAN. També es comprova el correcte funcionament dels leds i els botons de la PiCAN2 mitjançant la llibreria GPIO de la Raspberry. Si el terminal té botons, el programa de prova consisteix a encendre el *led* si és prémer el botó, en el cas de no tenir botons es programen els leds, perquè s'encenguin i s'apaguin cada segon, tres vegades consecutives.

4.1.3. Preparació física del CAN bus

Després de comprovar el correcte funcionament individual de cada terminal, es comença a realitzar la connexió física dels terminals que formen l'entorn. Cal esmentar que és necessari que les dues ECU's dels extrems siguin les que tenen activades les resistències de 120 ohms. Primer de tot es realitza la connexió dels components extrems, és a dir de l'ECUI i l'ECUIV. Tal com es pot veure a la Figura 9 la connexió es fa connectant en el terminal de cargol de la PiCAN2 el cable de CAN-L d'un terminal amb el cable del CAN-L de l'altre terminal, aquesta connexió entre ECU's també s'ha de realitzar amb el CAN-H [19]. Aquesta connexió es fa mitjançant els cables de la placa de proves i la placa de proves.



Figura 9. Connexió ECUI i ECUIV

Seguidament es realitza la connexió de la ECUII i ECUIII. S'introdueix en la placa de proves entre el cable CAN-L de l'ECUI i el cable CAN-L de l'ECUIV, el cable CAN-L de l'ECUII i també s'introdueix el cable CAN-L de l'ECUIII. També es va introduir el CAN-H de l'ECUII i de l'ECUIII entre els cables CAN-H de l'ECUI i l'ECUIV. La connexió dels cables al mòdul PiCAN2 es fa de la mateixa manera que a les ECU's explicades anteriorment. A la Figura 10 es pot veure una imatge del CAN bus amb totes les ECU's connectades.

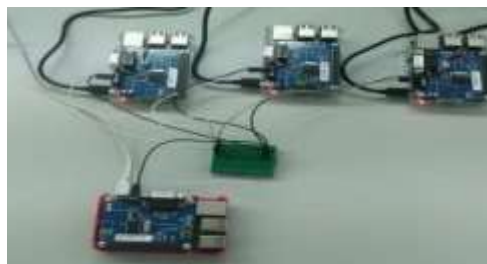


Figura 10. CAN bus connectat

4.2. Programa CAN bus

En aquesta part de l'enunciat es comenta el codi programat per desenvolupar les ECU's. Primer de tot s'expliquen les classes utilitzades, seguidament es defineix la part principal del programa que executen els terminals. Per finalitzar es descriuen les funcions que executen les ECU's.

4.2.1. Classe *MyListener*

MyListener és una extensió de la classe de python-can *can.Listener*. *MyListener* té com a objectiu principal executar el mètode sobreescrit *on_message_receive* quan arriba un missatge pel CAN bus. Aquesta funcionalitat està basada en el sistema *Publish/Subscribe* intern de la llibreria python-can, on les classes heretades del *can.Listener* se subscriuen a l'objecte *can.Notifier* per tal que siguin avisades quan arriba un missatge nou pel bus. Per defecte, quan arriba un missatge les classes executen el mètode *on_message_receive* [23].

En la classe *MyListener* un cop s'entra al mètode s'executa una funció que s'ha hagut d'enviar per paràmetre en inicialitzar la classe, de forma opcional també es pot enviar el paràmetre *needMsg*. Aquesta variable és un booleà que si és cert, en fer la crida a la funció esmentada anteriorment, s'envia com a paràmetre el missatge que s'ha enviat pel CAN bus. Per defecte la variable *needMsg* està inicialitzada a fals.

4.2.2. Classe *StopThread*

StopThread és una classe que hereta de *threading.Thread* de Python. Aquesta nova classe creada té com a finalitat poder controlar l'execució dels *threads* és a dir, afegeix certes funcionalitats als *threads* de Python perquè es puguin configurar d'una manera fàcil. Les accions que es poden realitzar són:

- Escollir quina funció es realitzarà en segon pla.
- Escollir el temps d'espera entre repeticions de la funció, per defecte és 1 segon.
- Poder aturar o reiniciar l'execució de la funció sense necessitat d'eliminar el *thread* de forma permanent, per defecte el *thread* comença executant la funció tot i que es pot configurar el contrari passant per paràmetre la variable *work* a fals.
- Eliminar el *thread* de forma permanent.

4.2.3. Part principal

La part principal consisteix a realitzar les parts que són comunes per totes les màquines i després dependent del valor enviat per paràmetres executa les funcionalitats específiques d'una ECU o d'un altre. Les parts comunes dels terminals són:

- La inicialització dels mòduls GPIO de la Raspberry, en estar connectada al mòdul PiCAN2 alguns números de la GPIO corresponent a components del mòdul.
- La inicialització de la interfície CAN, *can0*.
- La inicialització del CAN bus, és necessari especificar quin rang d'identificador es vol que s'escolti per cadascun dels terminals, aquest filtre es decideix enviant per paràmetre un diccionari amb l'identificador i una màscara. Perquè el missatge sigui acceptat pel mòdul, han de coincidir els resultats de les següents operacions:
 - Identificador del missatge && màscara
 - Identificador del terminal && màscara

L'ECUI i l'ECUIII s'inicialitzen sense cap filtre, en canvi l'ECUII només escolta missatges amb l'identificador de 0x7ce a 0x7ff i l'ECUIV missatges amb l'identificador de 0x2ce a 0x2ff.

- La finalització de les connexions del bus i de la GPIO, en el cas que el programa s'aturi per qualsevol mena d'error.

4.2.4. *Parts específiques terminals legítims*

Cadascuna de les ECU's de l'entorn de proves tenen unes funcionalitats específiques per poder desenvolupar les accions d'escrites en Arquitectura i funcions. En aquest apartat es descriuen les funcionalitats de l'ECU I, II, IV.

ECUI

Primer de tot, s'inicialitza la GPIO 24 com a botó. Amb la PiCAN2 connectada a la Raspberry, la GPIO 24 correspon al botó que es pot posar en el punt S1. Després, és crear un missatge CAN on la part important del missatge és l'identificador, el qual es troba dins del rang d'identificadors que escolta l'ECUIV. Dins d'un bucle infinit cada segon es va comprovant si s'està prement el botó S1. En el cas d'estar premut s'envia el missatge creat pel CAN bus.

ECUII

La part específica de l'ECUII comença inicialitzant la GPIO 22 que està vinculada al led del mòdul de la PiCAN2. Després, s'afegeix la crida a un mètode per poder controlar el led de forma analògica i d'aquesta manera poder variar la intensitat de la llum. Seguidament, s'inicialitza la classe MyListener explicada anteriorment. La funció que s'executa quan arriba un missatge que ha de ser escoltat per l'ECUII consisteix a observar les dades del missatge Si el valor de les dades és inferior a 85, no encén en led, en el cas d'estar entre 85 i 105 s'encén la llum del led però només a un 10% d'intensitat i si el valor enviat supera el número 105, s'encén al 100% d'intensitat. En cas d'haver un error a més de tancar les connexions comunes també es deixa d'escoltar el bus.

ECUIV

Es comença inicialitzant la GPIO22 per poder utilitzar el led de la PiCAN 2. Seguidament, s'inicialitza la classe StopThread explicada anteriorment. La funció que s'executa en segon pla és anar enviant de forma contínua cada segon un missatge amb l'identificador 0x2de, què és un que escolta l'ECUII. Els continguts dels missatges enviats de forma periòdica són un nombre aleatori entre 70 i 110. Després, s'inicialitza la classe MyListener perquè quan arriba un missatge que ha d'escoltar l'ECUIV s'encengui el led durant 1 segon. En finalitzar el programa es mata el *thread* del StopThread, també es deixa d'escoltar el bus a més de terminar la resta de connexions en la part principal.

4.2.5. *Part específica mòdul atacant*

A continuació es descriurà com s'ha desenvolupat el mòdul de l'atacant en l'entorn de proves.

Com s'ha explicat anteriorment l'objectiu del mòdul atacant és poder enviar als diferents terminals de l'entorn, diferents missatges i que les diferents ECU's els processin pensant que són missatges legítims. Els atacs realitzats són els següents:

- Enviar missatges amb un identificador que escolta l'ECUIV amb la finalitat de fer-se passar per l'ECUI, en aquest atac no s'està impedit que les dades de l'ECUI siguin processades, només s'afegeix que si l'atacant envia dades, també es processin

de la mateixa forma. Un exemple d'aquest tipus d'atac en un cotxe seria el fet de poder enviar al motor el senyal que s'està accionant el fre perquè redueixi la velocitat. L'important no és evitar que es pugi premé el fre sinó poder enviar el mateix senyal quan es desitja, com per exemple en una autopista. En el cas que els dos terminals vulguin enviar un missatge alhora, l'atacant té més prioritat per què els seus missatges tenen un identificador amb un valor superior.

- Enviar missatges amb un identificador que escolta l'ECUII amb la finalitat de fer-se passar per l'ECUIV. A diferència de l'atac explicat anteriorment, l'atacant vol evitar que es processin les dades de l'ECUIV perquè prengui només les accions pertinents per les dades enviades per l'atacant. Per aconseguir l'objectiu plantejat és necessari enviar les dades a l'ECUII amb més freqüència, amb la finalitat de què quan arriba una dada de l'ECUIV i l'ECUII comença a realitzar les accions corresponent al missatge, ràpidament arriba una dada de l'atacant i llavors l'ECUII deixa de realitzar les accions de les dades legítimes per atendre les dades de l'atacant. D'aquesta manera s'aconsegueix que les accions que pren el terminal la major part del temps són les esperades per l'atacant. Cal esmentar que per intentar disminuir la quantitat de missatges enviats per l'ECUIV la prioritat dels missatges de l'atacant és superior perquè en cas que els dos missatges vulguin enviar alhora un missatge, s'envii el missatge de l'atacant.

Seguidament es descriu el programa del mòdul atacant per poder realitzar els atacs descrits anteriorment. Primer de tot, s'inicialitzen la GPIO 23 i la GPIO 24 com a botons, en el mòdul PiCAN2 aquestes GPIO coincideixen amb els botons que es poden posar en la en les parts S1 i S2 respectivament. Després s'inicialitzen dues instàncies de la classe `StopThread` explicada anteriorment amb els paràmetres i funcions següents:

Thread 1

En aquest thread es realitza l'atac a l'ECUII, la funció que s'executa en segon pla consisteix a enviar un missatge amb l'identificador 0x2ff i el valor 150. L'identificador es troba en el rang que escolta l'ECUII i a més, és un número superior a l'identificador dels missatges de l'ECUIV. S'envia de forma contínua el valor de 150 perquè el led de la ECUII sempre estigui encès al 100% d'intensitat. Com a configuració de la classe `StopThread` a més d'enviar la funció per paràmetre també s'especifica el temps d'espera entre execucions de la funció com a 0,25 segons. Per finalitzar, s'especifica que el *thread* no comenci executant la funció sinó que es mantingui a l'espera.

Thread 2

En aquest thread es realitza l'atac a l'ECUIV, la funció que s'executa en segon pla consisteix a comprovar si s'està prement el botó S1. En el cas d'estar premut s'envia un missatge pel CAN bus amb l'identificador 0x7ff. L'identificador es troba en el rang que escolta l'ECUIV i a més és un número superior a l'identificador dels missatges de l'ECUI. A diferència del *thread 1* el temps d'espera entre execucions al thread 2 es manté en 1 segon i la funció es comença a executar un cop habilitat el *thread*.

Un cop inicialitzats els dos *thread* comença l'execució del bucle on es comprova que els dos threads es continuen executant de forma correcta, dins del bucle es comprova cada segon si s'està prement el botó del component S2. En el cas d'estar premut comprova si la instància del thread 1 s'està executant, si la funció està desactivada, s'activa i si està activa es desactiva. Aquesta funcionalitat s'utilitza perquè el mòdul de l'atacant només comenci a enviar missatges a l'ECUII,

quan es prem el botó S2, també s'aconsegueix que deixi d'enviar missatges quan es torna a prémer el botó S2. En el cas que un dels dos threads deixi de funcionar es finalitza l'execució i es comencen a tancar les connexions. A part de les connexions comunes en tots els terminals és necessari finalitzar l'execució dels dos *threads*.

4.3. Programació de Serveis

Amb l'objectiu de què les Raspberry Pi realitzin les seves funcions sense necessitat d'executar els programes de forma manual en tots els terminals, es va implantar l'execució dels programes com a serveis del sistema operatiu, s'utilitza el sistema systemd. Els codis donats a continuació són els utilitzats per l'ECUI però les explicacions i comandes són iguals per tots els terminals, a diferència de canviar els corresponents noms dels scripts i paràmetres. Primer de tot, es crea un petit script en bash que es pot veure en la Taula 9. Script d'inicialització En aquest, es busca pel sistema operatiu el fitxer del programa ECU.py, un cop trobat s'executa el fitxer en Python 3, s'envia com a argument el número de l'ECU on es troba el script, en aquest exemple s'envia el número 1 perquè és l'ECUI.

| |
|---|
| <code>#!/bin/bash</code> |
| <code>file=\$(find /home -type f -name "ECU.py")</code> |
| <code>python3 "\$file" 1</code> |

Taula 9. Script d'inicialització ECUI.sh

Seguidament es copia el script a la carpeta d'executables del sistema operatiu i s'afegeixen els permisos pertinent executant les comandes que es poden veure en la Taula 9.

| |
|---|
| <code>sudo cp ECUI.sh /usr/bin/ECUI.sh</code> |
| <code>sudo chmod +x /usr/bin/ECUI.sh</code> |

Taula 10. Canvi de localització i permisos ECUI.sh

A continuació, es crea el fitxer del servei. Tal com es mostra en la Taula 11 el script s'executarà després de l'estat multi-user, aquest estat del sistema és el punt on estan actives totes les funcions de la xarxa i es poden acceptar logins, però les GUI no estan inicialitzades. La variable de StandardOutput especifica que les sortides estàndard del programa van al fitxer can.log, en comptes d'anar al fitxer syslog com està definit per defecte. El servei executa el script /bin/bash i el script ECUI.sh.

| |
|--|
| [Unit] |
| Description= Active CAN bus module. |
| |
| [Service] |
| Type=simple |
| ExecStart=/bin/bash /usr/bin/ECUI.sh |
| StandardOutput = file:/var/log/can.log |
| |
| [Install] |
| WantedBy=multi-user.target |

Taula 11. Fitxer myservice.service

Per finalitzar cal executar les comandes que es poden observar a la Taula 12 per moure el fitxer a la carpeta de serveis del sistema operatiu /etc/systemd/system i per donar-li els permisos necessaris.

| |
|---|
| sudo cp myservice.service /etc/systemd/system/myservice.service |
| sudo chmod 644 /etc/systemd/system/myservice.service |

Taula 12. Comandes inicialitzar serveis.

4.4. IDS

En aquesta secció s'expliquen els passos necessaris per a la implantació correcta del mòdul IDS en l'entorn de proves. Com s'ha explicat anteriorment a Arquitectura i funcions el mòdul IDS consisteix a connectar l'ordinador al CAN bus. L'ordinador té incorporat el Snort que analitza les dades CAN i informa quan es produeix un atac. El mòdul IDS també pot executar l'algoritme k-means sobre els missatges CAN per poder obtenir informació dels atacs.

Cal esmentar que, en aquest treball només s'implementa un IDS amb la capacitat de poder detectar els atacs produïts a l'ECUII, això és degut al fet que en els atacs a l'ECUIV, és difícil trobar patrons que diferenciïn entre un missatge atac i un missatge legítim. Això és a causa que, tant els missatges de l'ECUI com els missatges de l'atacant envien les mateixes dades i els dos envien un sol missatge de forma asíncrona.

4.4.1. Connexió de l'ordinador amb el CAN bus

A continuació es redacta els procediments necessaris per realitzar la connexió entre l'entorn de proves i el mòdul IDS. Aquesta connexió permet registrar les dades que s'envien pel CAN bus. Cal esmentar que la connexió entre l'ordinador i el CAN bus es realitza mitjançant un cable *Ethernet* en comptes d'un cable OBD-II com es tenia plantejat principalment. Els motius d'aquesta decisió van ser els següents:

- Perquè l'ordinador pugui detectar les dades del CAN bus a través d'un cable OBD-II fa falta un hardware específic el qual no s'havia tingut previst en el pressupost inicial i té un preu aproximat de 50 euros.
- Connectar una ECU a l'ordinador mitjançant un cable *Ethernet* feia possible poder utilitzar el software *Canelloni* i poder transformar les dades de protocol CAN a protocol UDP. En tindre les dades en format UDP es pot utilitzar el IDS Snort.

A la Figura 11 es pot observar el CAN bus amb el mòdul IDS connectat a l'ECUII mitjançant el cable *Ethernet*.



Figura 11. Entorn de proves amb mòdul IDS

Primer de tot s'instal·la la llibreria de Canelloni tant en el mòdul IDS com en l'ECUII. Es descarrega la llibreria del repositori [21] llavors es compila i seguidament s'instal·la, executant dins del directori del repositori les comandes que es poden veure en la *Taula 13*.

| |
|---|
| <code>cmake -DCMAKE_BUILD_TYPE=Release</code> |
| <code>make</code> |
| <code>make install</code> |

Taula 13. Comanda instal·lació Canelloni

És necessari tindre una interfície CAN en l'ordinador per poder manipular les dades que arriben de l'entorn un cop s'han tornat a convertir en el protocol CAN. Seguidament en el mòdul IDS, com no hi ha components que puguin ser utilitzats per crear una interfície CAN, és necessari crear una interfície CAN virtual. Per crear la interfície virtual és necessari tindre instal·lada la llibreria SocketCan i executa les comandes observades en la Taula 14.

| |
|--|
| <code>sudo modprobe vcan</code> |
| <code>sudo ip link add name vcan0 type vcan</code> |
| <code>sudo ip link set dev vcan0 up</code> |

Taula 14. Instal·lació interfície can virtual

Com en les interfícies digitals no hi ha una limitació física en la velocitat i la quantitat de dades que es poden enviar per segon, és recomanable limitar la quantitat de dades i la latència en què s'envien els missatges per garantir que la interfície física els pugui processar i evitar pèrdua de paquets. Per configurar aquestes restriccions cal executar la comanda de la Taula 15.

```
sudo tc qdisc add dev vcan0 root tbf rate 300kbit latency 100ms
burst 1000
```

Taula 15. Comanda restricció vcan0

Un cop finalitzada la configuració de la interfície vcan0 s'executen les comandes per començar a enviar missatges des de la interfície can0 de l'ECUII a la interfície vcan0 del mòdul IDS. Aquestes comandes s'han d'executar cada vegada que es vol començar a enviar missatges entre els dos terminals. En l'ECUII cal executar la comanda de la Taula 16 i en el mòdul IDS cal executar la comanda de la Taula 17.

```
cannelloni -I can0 -R 192.168.137.1 -r 20000 -l 20000 -t 10000
```

Taula 16. Habilitar Canelloni ECUII


```
cannelloni -I vcan0 -R 192.168.137.212 -r 20000 -l 20000 -t 10000
```

Taula 17. Habilitar Cannelloni mòdul IDS

En deixar l'execució de les comandes en els terminals, ja es comencen a transmetre els missatges per la xarxa *Ethernet*. En la Figura 12 es pot veure la informació dels paquets de l'entorn de proves que arriben al mòdul IDS, es pot observar com els missatges CAN bus arriben a l'ordinador en format UDP. També es pot remarcar que tota la informació del paquet CAN es troba en el cos dels paquets UDP.



```
^C*** Caught Int-Signal
06/08-18:36:00.519462 B8:27:EB:67:37:9A -> 98:29:A6:2D:E1:C4 type:0x800 len:0x3C
192.168.137.212:20000 -> 192.168.137.1:20000 UDP TTL:64 TOS:0x0 ID:43539 IpLen:2
0 DgmLen:46 DF
Len: 18
02 00 BC 00 01 00 00 02 DE 08 00 00 00 00 00 00 .....
00 48 .H
```

Figura 12. Contingut missatges CAN en UDP

4.4.2. Implementació IDS

A continuació es detalla la implementació d'instal·lar el IDS en el mòdul i la creació de la regla per detectar l'atac a l'ECUII.

4.4.2.1. Snort

Com s'ha explicat anteriorment el Snort és un IDS de codi obert que per detectar si un missatge és maliciós, utilitza un sistema de regles intern. Per instal·lar el programa fa falta executar la comanda que es mostra en la Taula 18.

```
sudo apt get install snort
```

Taula 18. Comanda instal·lació snort

Un cop instal·lat el Snort en el mòdul IDS és necessari configurar diferents aspectes del programa com per exemple: quina és la xarxa pròpia, quin és el *path* dels diferents fitxers del programa, quines regles té què utilitzar, etc. Aquestes configuracions s'han de realitzar en el fitxer de configuració, per defecte el fitxer es troba en la ubicació `/etc/snort/snort.conf`. Per aquest treball els únics canvis en el fitxer van ser canviar la IP de la xarxa a la de l'entorn de proves, és a dir la 192.168.137.1/24 i afegir el fitxer de les regles pròpies.

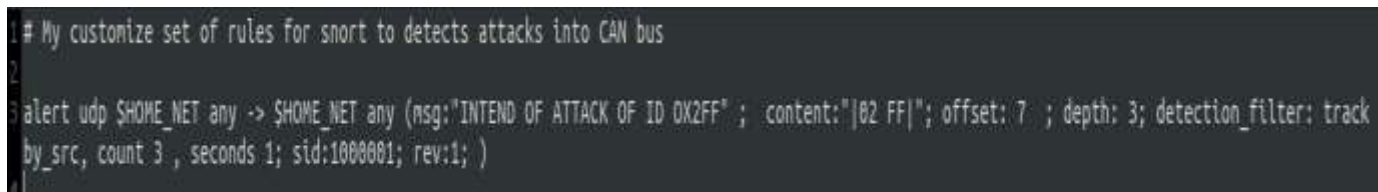
Snort té instal·lats un conjunt de fitxers amb diferents regles ja creades que poden detectar diferents atacs típics com per exemple: atac de NFS, atacs de denegació de servei, *backdoors*, etc. Tanmateix, també dona l'opció d'afegir fitxers amb conjunts de regles pròpies, per defecte els fitxers de normes es troben en el directori `/etc/snort/rules` i porten l'extensió de fitxer `.rules`.

Les regles Snort es basen a descriure les característiques que ha de tindre un missatge per tal de ser categoritzat com a atac i prendre les accions pertinents. Pel tipus d'atac que es realitza a l'ECUII s'ha tingut en compte sobretot l'identificador del missatge i la quantitat de paquets que

s'envien per segon, ja que, són els dos factors més importants que diferencien els missatges enviats per l'ECUII dels missatges de l'atacant. Amb la finalitat de poder visualitzar quin és l'identificador de l'atacant i la freqüència que envia el missatge s'utilitza l'algorisme k-mean que s'explicarà més endavant.

A Figura 13 es pot veure la regla utilitzada per detectar els atacs produïts a l'ECUII. Les característiques que ha de tindre un missatge per tal que el Snort el detecti com a un atacat són les següents:

- Ha de ser un missatge UDP enviat per la xarxa local a un altre dispositiu de la xarxa local, aquesta característica representa la comunicació del mòdul IDS amb l'entorn de proves mitjançant Canelloni.
- Ha de tindre en el cos del paquet el contingut 02 FF, és l'identificador que tenen els missatges CAN que envia el mòdul atacant. Per assegurar que la part del missatge que s'analitza és on es troba informació de l'identificador del missatge s'ha afegit a la norma les especificacions d'*offset* i *depth*. L'*offset* són la quantitat de bytes des del principi de les dades que el programa ha de desplaçar-se abans de començar de buscar el contingut especificat. En canvi, el *depth* és el rang de bytes des de l'offset on el programa ha de buscar el contingut. Per exemple si a la configuració es troba especificat un offset de 7 i un depth de 3 el programa buscarà els valors especificats a partir del 8 byte i només el buscarà els valor en els 3 bytes següents [24].
- El missatge ha de ser enviat més de tres cops cada segon. Aquesta característica s'aconsegueix amb la configuració `detection_filter: track_by_src, count, second` [25].



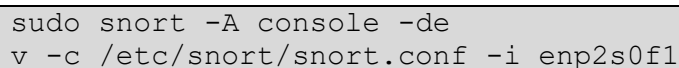
```

1 # My customize set of rules for snort to detects attacks into CAN bus
2
3 alert udp $HOME_NET any -> $HOME_NET any (msg:"INTEND OF ATTACK OF ID 0X2FF" ; content:"|02 FF|"; offset: 7 ; depth: 3; detection_filter: track
  by_src, count 3 , seconds 1; sid:1000001; rev:1; )

```

Figura 13. Norma Snort

Un cop configurada la norma, cal deixar el Snort executant-se per tal que alerti en el cas de detectar un ataca a l'ECUII. A la Taula 19 es pot observar la comanda necessària per executar el Snort amb les regles implantades. On `enp2s0f1` és el nom de la interfície de xarxa del mòdul IDS que està connectada a l'entorn, el Snort analitza els missatges que són rebuts o enviats per aquesta interfície.



```

sudo snort -A console -dev
v -c /etc/snort/snort.conf -i enp2s0f1

```

Taula 19. Comanda executar Snort

4.4.2.2. *Kmeans*

En aquest treball l'algorisme del k-means és utilitzat per analitzar les dades que s'envien pel CAN bus amb la finalitat de poder detectar de forma visual si en els missatges observats s'han produït atacs i poder treure informació, com per exemple: detectar l'identificador de l'atacant, la freqüència de les dades, el tipus de dades enviades, etc. Cal esmentar que no es poden detectar atacs a temps real, ja que fa falta tindre les dades emmagatzemades per poder executar l'algorisme, tanmateix, la informació treu del k-means s'utilitza per configurar les regles del Snort i així poder prevenir les pròximes amenaces.

El valor de les dades utilitzades en el k-means per obtenir els diferents clústers són la localització dels punts en un gràfic de tres dimensions, els valors en cadascun dels eixos de coordenades s'obtenen de tres factors dels missatges enviats pel CAN bus: l'identificador del missatge, les dades del missatge i en quin moment ha arribat el missatge. Es van decidir aquests factors perquè són els que més poden diferenciar els diferents tipus de missatges i separar els missatges d'un atacat dels missatges legítims.

El programa que realitza l'algorisme té les següents parts principals:

Inicialització

- Per rebre i tractar les dades del CAN bus és necessari com en la resta de terminals inicialitzar la interfície CAN i el bus, cal tenir en compte que a diferència de les altres ECU's que tenen el mòdul PiCAN2 en el mòdul IDS s'ha utilitzat una interfície CAN virtual amb el nom vcan0. Seguidament, s'afegeix el filtre per només escoltar les dades amb un identificador entre 0x7ce i 0x7ff. S'inicialitza la classe MyListener, amb una funció que permet emmagatzemar la informació dels missatges que arriben al mòdul. A més, s'emmagatzema en una variable, el temps inicial del programa.

Recollida de dades

- Com s'ha explicat anteriorment per realitzar l'algorisme s'han utilitzat els valors de tres aspectes dels missatges. Amb la finalitat d'emmagatzemar les diferents característiques dels missatges sense confondre a quin missatge pertanyen, s'implementa un diccionari on cadascuna de les claus representa un dels eixos de coordenades. Cada clau conté una llista on es guarden els valors per cadascun dels missatges. Per aconseguir el temps en què ha arribat el missatge es resta el temps de l'inici del programa amb els temps registrat quan ha arribat el missatge.

Bucle principal

- Amb la finalitat de rebre dades, el bucle principal consisteix en un comptador de 10 segons, quan el temps finalitza, es deixa d'escoltar pel CAN bus.

Kmeans

- Un cop finalitzat el temps per recollir dades s'emmagatzema la informació en un *dataFrame*, és un mètode que organitza les dades en un espai de coordenades on cadascun dels eixos són les diferents claus del diccionari on s'han emmagatzemat les dades del missatge.
- S'executa l'algorisme kmeans, s'utilitza una llibreria de Python anomenada *sklearn.cluster* amb la finalitat de tindre un algorisme més testejat per diversos usuaris del que podria estar un algoritme implementat manualment. S'ha decidit

realitzar el k-means amb una classificació de dos clústers. Ja que, l'objectiu és diferenciar si els missatges corresponen a un atacant o no.

- Finalment es representen els resultats obtinguts en un gràfic de punts, en ser un espai de coordenades en tres dimensions es va decidir mostrar la informació en dos gràfics de dues dimensions on en un es mostren les coordenades del temps i de les dades i en l'altre les coordenades de les dates i dels identificadors. Per representar els gràfics s'ha utilitzat la llibreria matplotlib, és una llibreria per poder crear gràfics en Python amb una sintaxi semblant a la del llenguatge MATLAB. El color de les dades en el gràfic depenen del clúster assignat, a més de les dades analitzades en el gràfic s'afegeixen els centroides com a punts vermells.

Finalització

- Un cop mostrat el gràfic amb els resultats obtinguts com a la resta d'ECU's de l'entorn, es finalitza la connexió amb el CAN bus.

5. Avaluació

En aquest apartat es mostren els resultats obtinguts en les proves realitzades per garantir que es compleixen els objectius especificats en l'apartat Arquitectura i funcions. Les proves es separen en tres grans grups: avaluació de les ECU's I,II i IV, avaluació del funcionament del mòdul de l'atacant i avaluació del mòdul IDS tant del Snort com de l'algoritme Kmeans.

Per poder validar que les dades rebudes pels diferents terminals de l'entorn de proves són les correctes s'utilitza la comanda de la Taula 20.

```
candump can0
```

Taula 20. Comanda mostrar dades CAN bus

És una comanda de la llibreria SocketCAN que escriu per pantalla del terminal la informació dels missatges que es reben per la interfície CAN can0.

Primer de tot, en els terminals es comprova que els resultats visuals siguin els esperats i seguidament es garanteix que en cadascun dels terminals arriben els missatges correctes. L'objectiu de la ECUI és enviar missatges de forma asíncrona amb un identificador que escolta l'ECUIV, quan arriba un nou missatge amb un identificador dins del seu rang, l'ECUIV mostra un canvi visual. Com s'ha explicat anteriorment per assolir aquest objectiu, l'ECUI té instal·lat un botó que quan es prem, envia un missatge asíncron, aquest és detectat per l'ECUIV i amb la finalitat d'informar d'un nou missatge, encén el led. Només s'envia un missatge pel CAN bus per cada segon que es prem el botó, aquesta restricció es va programar per evitar que es col·lapsi la xarxa en prémer el boto.

En la Figura 14 es poden observar els canvis visuals esperats. Per garantir el funcionament desitjat del filtre d'identificadors configurat en el terminal, es canvia l'identificador dels missatges de l'ECUI a un que no ha d'analitzar l'ECUIV. S'observa que encara que es premi el botó de l'ECUI el led de l'ECUIV no s'encén.



Figura 14. Canvis visuals ECUIV

Un cop validat que els terminals realitzen les accions desitjades es garanteix que els missatges que arriben pel CAN bus són els esperats. En la Figura 15 es veu el resultat en executar la comanda *candump* en l'ECUIV en estar prement el botó de l'ECUI.

```
pi@raspberrypi:~ $ candump can0
can0 7DE [8] 00 19 00 01 03 01 04 01
can0 7DE [8] 00 19 00 01 03 01 04 01
can0 7DE [8] 00 19 00 01 03 01 04 01
```

Figura 15. Candump ECUI

La següent funcionalitat testejada és l'enviament de missatges de l'ECUIV i com tracta els paquets CAN l'ECUII. Com s'ha explicat anteriorment l'ECUIV envia missatges de forma periòdica amb valors aleatoris dins d'un rang, amb un identificador que analitza l'ECUII. A diferència de l'ECUII, l'ECUIV mostra de forma visual certa informació de les dades dels missatges rebuts. Per desenvolupar aquesta funcionalitat en l'entorn de proves, l'ECUIV envia cada segon un missatge amb un identificador que observa l'ECUII i amb un valor aleatori entre 70 i 110. L'ECUII quan rep un paquet observa el seu valor, si el valor es troba entre 70 i 85 no encén el led, si es troba entre 85 i 105 s'encén el led al 10% de la seva intensitat i s'encén el llum al 100% si el valor és major de 105.

Quan s'encén l'entorn de proves es veu com de forma automàtica el led de l'ECUII s'encén i s'apaga automàticament, com que la probabilitat de què un valor sigui major de 105 és del 5%, normalment la llum s'encén amb molt poca intensitat, no obstant en alguns moments es pot observar durant un segon una llum amb més intensitat. Un cop comprovat que es produeixen les accions esperades, es realitzen les següents comprovacions per observar que l'entorn funciona com s'esperava:

- Es va verificar que prement el botó de l'ECUI el patró del led de l'ECUII no varia.
- Es va verificar que es continua encenent el led de l'ECUIV en prémer el botó de l'ECUI.

Seguidament es comprova que els missatges del CAN bus són els correctes, a la Figura 16 es pot veure el resultat de la comanda *candump* a l'ECUII. L'únic canvi entre els missatges és l'últim byte en el camp dels missatges, les dades es visualitzen en hexadecimal però en fer la conversió es pot observar que cap valor és superior a 110 o inferior a 70.

```
pi@raspberrypi:~ $ candump can0
can0 2DE [8] 00 00 00 00 00 00 00 61
can0 2DE [8] 00 00 00 00 00 00 00 4D
can0 2DE [8] 00 00 00 00 00 00 00 4B
can0 2DE [8] 00 00 00 00 00 00 00 4D
can0 2DE [8] 00 00 00 00 00 00 00 4B
```

Figura 16. Candump ECUII

En la Figura 17 es pot observar el resultat d'executar la comanda en l'ECUII a l'hora de prémer el botó de l'ECUI, es pot observar que entre els missatges enviats per l'ECUIV es troba el missatge de l'ECUI, d'aquesta manera es pot confirmar que tots els terminals d'un CAN bus poden rebre tots els missatges que s'envien per la xarxa. Els missatges de l'ECUIV s'envien de forma periòdica cada segon, l'ECUI només envia un missatge per cada segon que es té el botó premut, per tant, és

estadísticament improbable que els dos terminals vulguin enviar missatges al mateix temps. Tanmateix, els missatges de l'ECUI tenen més prioritat per tant en el cas que es vulguin enviar els dos missatges alhora s'enviarà abans el missatge de l'ECUI que el de l'ECUIV.

```
pi@raspberrypi:~ $ candump can0
can0 2DE [8] 00 00 00 00 00 00 00 63
can0 2DE [8] 00 00 00 00 00 00 00 59
can0 2DE [8] 00 00 00 00 00 00 00 63
can0 2DE [8] 00 00 00 00 00 00 00 4C
can0 2DE [8] 00 00 00 00 00 00 00 67
can0 7DE [8] 00 19 00 01 03 01 04 01
can0 2DE [8] 00 00 00 00 00 00 00 6B
```

Figura 17, Candump ECUII amb botó ECUI

Després es comprova com afecten els atacs produïts per l'ECUIII a la resta de l'entorn de proves. El botó S1 del mòdul PiCAN 2 de l'ECUIII correspon a l'activació de l'atac a l'ECUIV i el botó S2 correspon a l'activació de l'atac a l'ECUII. Quan es prem el botó S1, s'envia un únic missatge de forma asíncrona amb un identificador que analitza l'ECUI, l'identificador té un número superior a l'identificador dels missatges de l'ECUI amb la finalitat de què els missatges de l'atacant tinguin prioritats en cas de col·lisió.

Quan es prem el botó es pot observar com s'encén el led de l'ECUIV de la mateixa forma que si el missatge hagués estat enviat per l'ECUI. Es pot observar en la Figura 18 el resultat de la comanda *candump* en l'ECUIV quan s'està prement el botó S1 de l'ECUIII, es pot observar com l'única diferència amb els missatges de l'ECUI és l'identificador.

```
pi@raspberrypi:~ $ candump can0
can0 7FF [8] 00 19 00 01 03 01 04 01
can0 7FF [8] 00 19 00 01 03 01 04 01
```

Figura 18. Candump ECUIV botó ECUIII

En prémer el botó de l'ECUI alhora que es prem el botó S1 de l'ECUIII visualment no es pot veure la diferència a si només s'estigués prement un dels botons, el led de l'ECUIV es continua encenent. Malgrat això, a la Figura 19 es pot veure com en l'ECUIV arriben missatges dels dos terminals.

```
Cpi@raspberrypi:~ $ candump can0
can0 7DE [8] 00 19 00 01 03 01 04 01
can0 7FF [8] 00 19 00 01 03 01 04 01
can0 7DE [8] 00 19 00 01 03 01 04 01
can0 7FF [8] 00 19 00 01 03 01 04 01
```

Figura 19. Candump ECUIV botó ECUIII i ECUI

Seguidament es realitza l'atac a l'ECUII, en prémer el botó S2 es comencen a enviar valors de forma periòdica cada 0,25 segons, es deixen d'enviar missatges si és tronar a prémer el botó. Els missatges tenen un identificador que analitza l'ECUII i és un número major que l'identificador dels missatges de l'ECUIV, d'aquesta forma en cas de col·lisions els missatges de l'atacant tenen més prioritats. El valor dels missatges és sempre de 150.

A diferència de l'atac anterior es pot veure una diferència visual quan s'envien missatges de l'atacant a quan s'envien els missatges legítims. Es pot observar que quan es prem el botó S2 de l'ECUI, el led de l'ECUII comença a estar encès al 100% d'intensitat la gran quantitat de temps. Això és gràcies al fet que tot i seguir arribant missatges de l'ECUIV no es poden visualitzar els canvis en la intensitat del led, ja que quan arriba un missatge legítim pràcticament de forma immediata arriba un missatge de l'atacant que en tindre sempre un contingut amb valor superior a 105 encén el led al 100%. Per produir aquesta denegació de servei el mòdul de l'atacant envia missatges a l'ECUII amb molta més freqüència que l'ECUI, també en cas de col·lisions seran enviats els missatges del mòdul de l'atacant gràcies a un identificador de missatges amb un número superior.

A la Figura 20 es pot veure com en l'ECUII la gran majoria de missatges que arriben són de l'atacant tot i que també es reben els missatges de l'ECUIV.

```
can0  2FF  [8]  00 00 00 00 00 00 00 00 96
can0  2FF  [8]  00 00 00 00 00 00 00 00 96
can0  2DE  [8]  00 00 00 00 00 00 00 00 62
can0  2FF  [8]  00 00 00 00 00 00 00 00 96
can0  2FF  [8]  00 00 00 00 00 00 00 00 96
can0  2FF  [8]  00 00 00 00 00 00 00 00 96
can0  2FF  [8]  00 00 00 00 00 00 00 00 96
```

Figura 20. Candump ECUII amb ECUIII i ECUIV

Un cop testejat el mòdul de l'atacant es comença a comprovar el funcionament del mòdul IDS. Primer de tot, un cop connectats l'ECUII i el mòdul IDS mitjançant un cable *Ethernet* es comprova que les IP es troben ben configurades realitzant un ping des del mòdul IDS a la IP 192.168.137.212 que és la IP que té assignada l'ECUII tal com es pot veure a la Figura 8. També es realitza un ping en el sentit invers, és a dir, s'executà un ping des de l'ECUII a la IP 192.168.137.1. En la Figura 21 es pot veure com els dos ping es poden realitzar correctament.

```
PING 192.168.137.212 (192.168.137.212) 56(84) bytes of data.
64 bytes from 192.168.137.212: icmp_seq=1 ttl=64 time=0.727 ms
64 bytes from 192.168.137.212: icmp_seq=2 ttl=64 time=0.423 ms
64 bytes from 192.168.137.212: icmp_seq=3 ttl=64 time=0.450 ms
^C
```

```
pi@raspberrypi:~ $ ping 192.168.137.1
PING 192.168.137.1 (192.168.137.1) 56(84) bytes of data.
64 bytes from 192.168.137.1: icmp_seq=1 ttl=64 time=0.639 ms
64 bytes from 192.168.137.1: icmp_seq=2 ttl=64 time=0.449 ms
```

Figura 21. Ping entre ECUII i mòdul IDS

Seguidament s'activa la transferència dels paquets CAN bus al mòdul IDS amb *Cannelloni*. Amb la finalitat de detectar que els paquets arriben de forma correcta s'utilitza l'eina Wireshark per observar tots els paquets que arriben per la interfície de xarxa on està connectat el cable *Ethernet*. En la Figura 22 es pot veure una sèrie de paquets capturats pel Wireshark, es pot observar que els paquets són en protocol UDP i que en el cos es troba la informació del CAN bus.

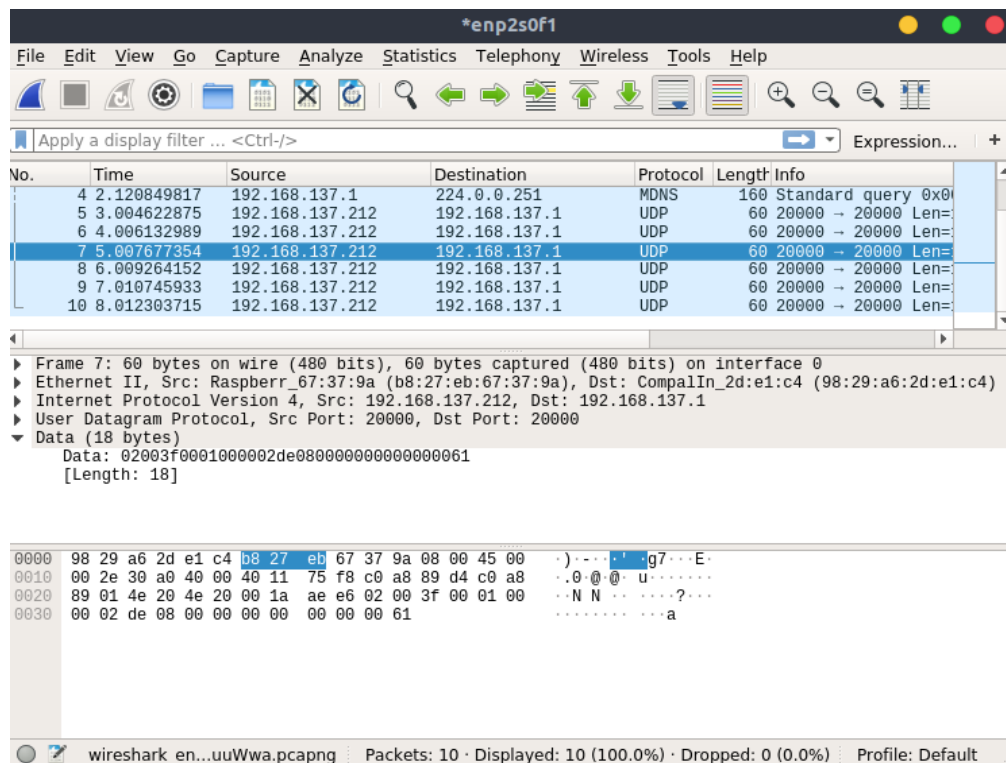


Figura 22. Captura Wireshark

Després de comprovar que els paquets CAN són rebuts de forma correcta al mòdul IDS, es testja el funcionament i s'analitzen els resultats de l'algoritme k-means. Com s'ha explicat anteriorment a *Kmeans* el programa recull les dades que van arribant del CAN bus durant 10 segons i seguidament executa l'algoritme mostrant el resultat en dos gràfics. Un amb els valors de la dada del missatge i el temps d'arribada i un altre amb els valors de la dada i l'identificador del missatge.

El programa s'executa en dues situacions diferents:

- El mòdul atacant està enviant missatges a l'ECUII.
- No s'estan realitzant atacs.

Cal esmentar que no es van realitzar més proves per falta de temps tot i ser necessàries per a una correcta validació de l'efectivitat de l'algoritme. Això és perquè aquesta funcionalitat va ser implementada com a extra en el projecte, per tant no se li va poder dedicar el mateix temps que a la resta de funcionalitats.

Primer de tot s'executa el programa mentre s'està realitzant l'atac a l'ECUII. En la Figura 23 es pot observar el resultat de l'algoritme k-means quan recull dades mentre s'està produint un atac. Es pot observar com l'algoritme a dividit les dades en dos clústers, les dades d'un clúster es troben en groc i les de l'altre en lila. Els centroides de cadascun dels clústers són representats amb un punt de color vermell.

Es pot veure com es troba una gran diferència entre els dos grups de missatges, tant per la freqüència d'enviament dels missatges com en l'identificador. Sense informació prèvia de l'atac, si se sap que el comportament habitual de les dades es que s'enviïn a una freqüència aproximada d'un segon, es pot deduir que els missatges enviats amb l'identificador 1890 són paquets no legítims i a més que s'estan enviant aproximadament 4 missatges per segon.

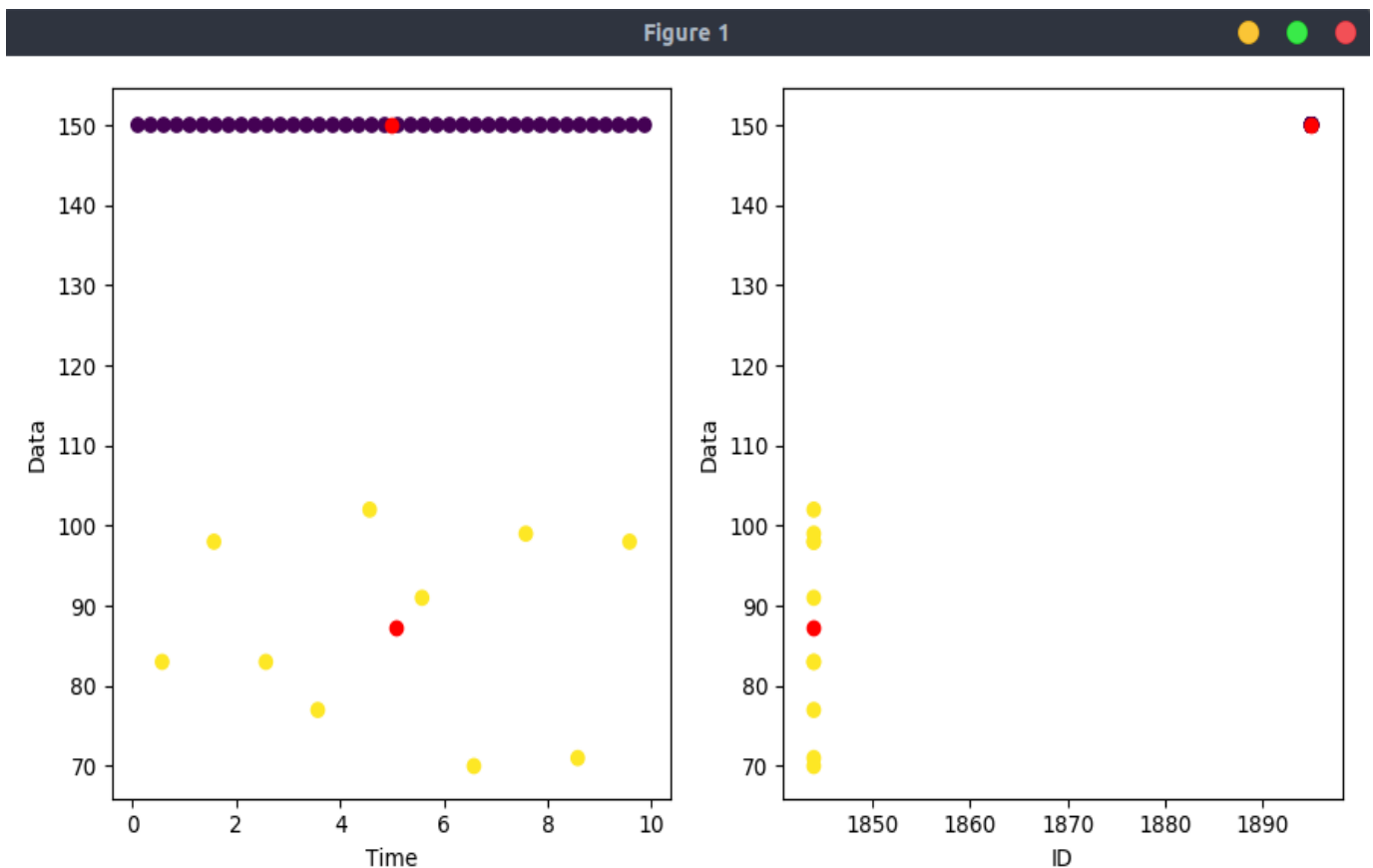


Figura 23. K-means amb atac

En la Figura 24 es troba el gràfic resultant quan en l'entorn de proves no s'estan produint atacs a l'ECUII. A diferència que en la Figura 23, no es veu una gran variació entre els dos grups de clústers, ja que els dos grups tenen aproximadament la mateixa freqüència d'enviament i són enviats amb el mateix identificador. Per tant, tot i que el k-means per defecte separa les dades en dos clústers, en el gràfic no es troben evidències per pensar que s'està produint un atac.

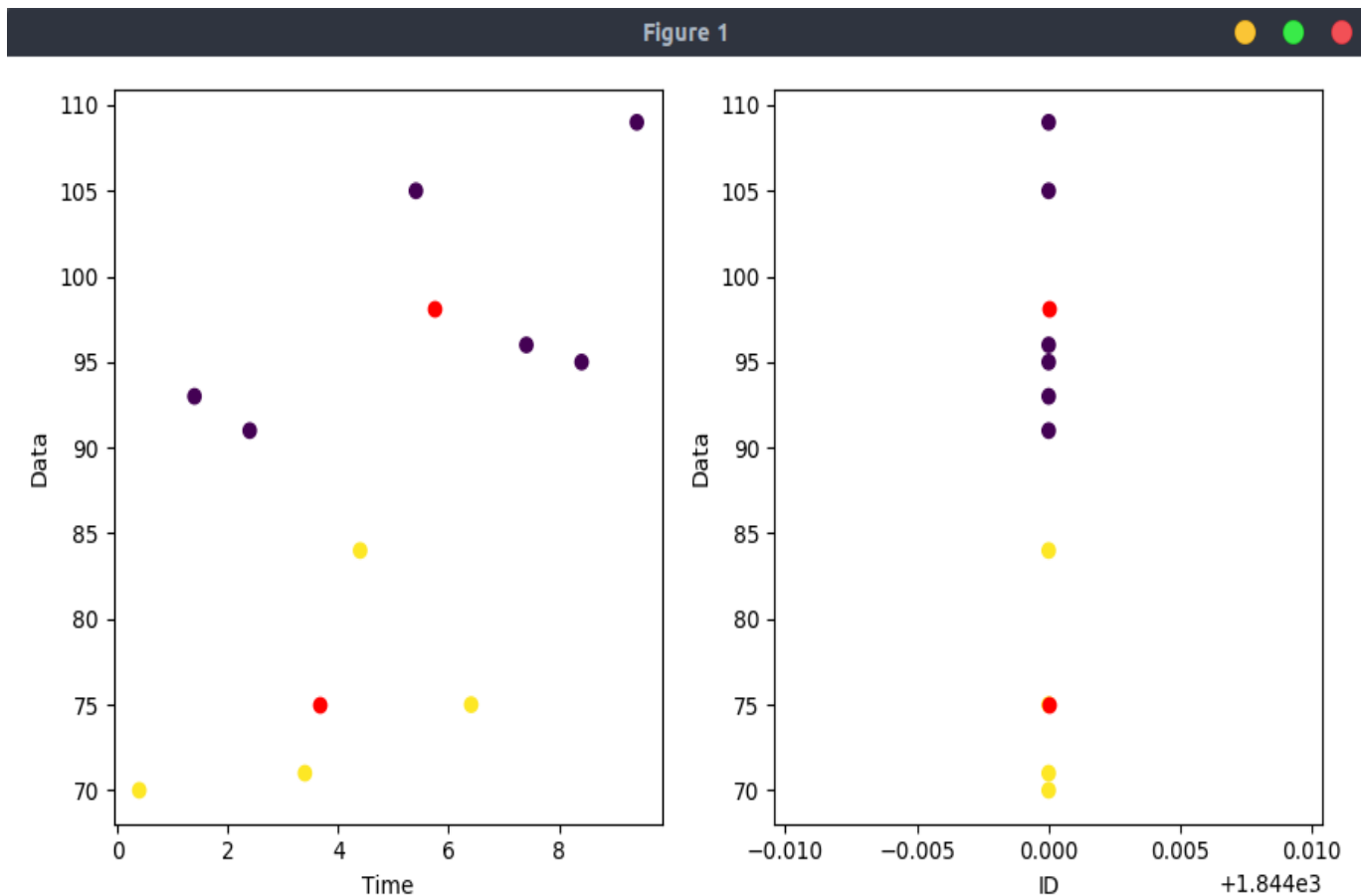
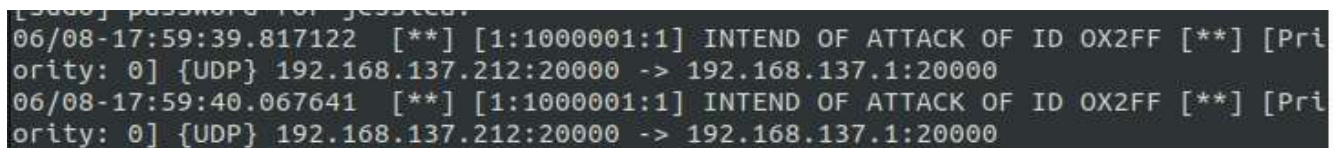


Figura 24. K-means sense atac

Per finalitzar, un cop obtinguda la informació dels missatges de l'atacant, es valida el funcionament de la regla Snort. Primer de tot, en la norma de la Figura 13 s'introdueix l'identificador dels missatges i la quantitat de missatges enviats per segon. Seguidament, s'executa la comanda de la Taula 19 amb la finalitat de què el IDS comenci a analitzar els paquets i avisi en cas de detectar una amenaça.

En la Figura 25 es pot veure l'avís que envia per pantalla el Snort quan detecta que l'atacant està enviant missatges a l'ECUII. Per testejar el IDS s'ha comprovat que el comportament del Snort és el desitjat en les següents situacions:

- Quan es realitza atac a l'ECUII el Snort envia l'avís.
- Quan no s'estan realitzant atacs però l'ECUIV està enviant missatges a l'ECUII el Snort no detecta cap amenaça,
- Quan el mòdul atacant envia missatges a l'ECUIV el Snort no detecta cap amenaça, aquest comportament és correcte, ja que el IDS només està programat per detectar els atacs dirigits a l'ECUII.



```

[0000] passwor...
06/08-17:59:39.817122  [**] [1:1000001:1] INTEND OF ATTACK OF ID 0X2FF [**] [Pri
ority: 0] {UDP} 192.168.137.212:20000 -> 192.168.137.1:20000
06/08-17:59:40.067641  [**] [1:1000001:1] INTEND OF ATTACK OF ID 0X2FF [**] [Pri
ority: 0] {UDP} 192.168.137.212:20000 -> 192.168.137.1:20000

```

Figura 25. Avís Snort

6. Conclusions

En aquest treball s'ha dissenyat i desenvolupat un CAN bus. Aquest és plenament funcional i permet enviar i rebre missatges. Al bus s'hi han connectat diverses Raspberry Pi que simulen les ECU's que trobaríem en un vehicle. En cadascuna de les Raspberry s'ha inclòs la funcionalitat per enviar i/o rebre missatges. Les funcionalitats implementades per cadascuna de les Raspberry han sigut les següents:

- En prémer un botó envia un missatge a un altre dispositiu.
- En rebre un missatge que ha d'escoltar, la Raspberry Pi encén un led. A més, cada segon envia un missatge amb un valor entre 70 i 110.
- En rebre un missatge que ha d'escoltar la Raspberry Pi observa el valor del paquet. Depenent del rang del valor, el led s'encén amb molta intensitat, s'encén amb poca intensitat o no s'encén.

Tot i ser simulacions d'ECU's en el CAN bus desenvolupat es podrien injectar trames de cotxes, fins i tot es podrien utilitzar les Raspberrys i els programes desenvolupats per introduir dades en el CAN bus d'un vehicle. L'entorn implementat és modular i permet poder afegir tantes Raspberry com es desitgin d'una forma senzilla i ràpida.

A continuació un dels mòduls s'ha emprat per fer atacs de manera que podia alterar el comportament dels altres mòduls. El mòdul envia missatges amb una prioritat i una freqüència d'enviaments superior respecte la resta de terminals.

Cal destacar que s'ha connectat al CAN bus un ordinador amb Snort per detectar atacs. Aquesta configuració permetria obtenir dades i detectar atacs amb altres tipus de busos. Si els vehicles incorporen xarxes *Ethernet*, es podrien detectar atacs en aquestes.

Finalment també s'ha desenvolupat un algorisme de IA, el k-means, per complementar Snort. Amb Snort tenim patrons, és a dir, si es produeix una acció que prèviament s'ha identificat com un atac salta un avís. Amb el k-means podem detectar situacions anòmales i implementar un patró al Snort per llençar una alarma.

El correcte funcionament del treball es comprova amb les accions que mostren els terminals de forma visual i amb l'anàlisi dels missatges que es van enviant per la xarxa. En executar l'entorn de proves podem observar com sense realitzar cap acció el led de l'ECUII va variant la seva intensitat. En prémer el botó de l'ECUI s'encén el led de l'ECUIV. En activar l'atac a l'ECUII, la Raspberry manté el led encès al 100% la major part del temps. En activar l'atac a l'ECUIV aquesta encén el led. En examinar els missatges del CAN bus s'observa que tots els terminals envien els missatges de forma correcta. També es pot comprovar com tots els terminals poden rebre tots els paquets.

En executar l'algoritme k-means mentre es produeix un atac a l'ECUII es pot observar que l'algoritme és capaç de diferenciar entre els missatges del mòdul de l'atacant i els missatges legítims. En introduir la informació en el Snort i començar a analitzar dades es pot veure com en executar un atac a l'ECUII es llença l'alarma.

6.1. Treball futur

Aquest treball dóna la possibilitat a un gran ventall de millores i ampliacions que es poden realitzar a partir de l'entorn de proves desenvolupat. Malgrat això, en un treball de final de carrera són impossibles d'implementar correctament degut a la falta de temps. En aquest apartat es descriuran algunes millores que es podrien realitzar, que s'han pensat durant la realització del treball.

Una possible ampliació del treball consisteix a ampliar el tipus de dades que ha de considerar el Snort i d'aquesta forma que sigui capaç de poder detectar més atacs que es poden produir en un CAN bus. Per aquesta ampliació també seria desitjable ampliar els tipus d'atacs i la varietat de les dades que s'envien.

Un altre aspecte que m'agradaria haver desenvolupat, és introduir, en l'entorn de proves, paquets CAN trets d'un cotxe amb la finalitat d'analitzar-los i poder realitzar diferents accions com per exemple enginyeria inversa.

Aprofitant el CAN bus es podria provar l'eficiència de diferents mètodes de seguretat com per exemple autenticació de terminals mitjançant claus privades, signatures, etc. També es podria implementar un Firewall en algun punt de l'entorn.

Per concloure, seria interessant poder desenvolupar una IA més complexa i entrenada amb més dades, amb la finalitat que sigui capaç de detectar més tipus d'atacs en diferents situacions.

7. Referències

- [1] <https://www.technologyreview.com/2019/02/15/137381/self-driving-cars-take-the-wheel/> [Innovacions cotxe elèctric] 01/05/2020
- [2] <https://www.sciencedirect.com/science/article/pii/S2046043017300035> [Innovacions cotxe elèctric] 01/05/2020
- [3] <https://www.rswebsols.com/tutorials/technology/benefits-self-driving-cars> [Vehicles Inteligents] 02/10/2019
- [4] <https://www.itsdigest.com/10-advantages-autonomous-vehicles> [Vehicles Inteligents] 02/10/2019
- [5] <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/> [Atacs Cotxes Intel·ligents] 02/10/2019
- [6] <https://www.upstream.auto/research/automotive-cybersecurity/?id=null> [Atacs Cotxes Intel·ligents] 02/10/2019
- [7] <https://whatis.techtarget.com/definition/vehicle-intelligence> [Vehicles Inteligents] 02/10/2019
- [8] <https://www.sciencedirect.com/science/article/pii/S2351978918312794> [Atacs CAN bus] 30/10/2019
- [9] https://link.springer.com/chapter/10.1007/978-3-030-03748-2_38 [Atacs CAN bus] 30/10/2019
- [10] <https://www.researchgate.net/publication/320150950> Cyber security attacks to modern vehicular systems [Atacs CAN bus] 30/10/2019
- [11] <https://www.electronicdesign.com/markets/automotive/article/21806349/automotive-Ethernet-the-future-of-incar-networking> [Arquitectures futures] 01/05/2020
- [12] <https://www.researchgate.net/publication/308086489> Replacement of the Controller Area Network CAN protocol for future automotive bus system solutions by substitution via optical networks [Arquitectures futures] 01/05/2020
- [13] <https://www.pistonheads.com/features/ph-features/what-is-an-electronic-control-unit-ph-explains/37/771> [Definició d' ECU] 18/11/2019
- [14] https://en.wikipedia.org/wiki/Electronic_control_unit [Tipus d'ECU's] 21/11/2019
- [15] https://cecas.clemson.edu/cvel/auto/systems/ep_steering.html [Definició PSCU] 21/11/2019
- [16] https://es.wikipedia.org/wiki/Bus_CAN [Funcionament del CAN bus] 11/12/2019
- [17] https://canvas.uw.edu/files/47669787/download?download_frd=1 [Vulnerabilitats del CAN bus] 22/01/2020
- [18] https://en.wikipedia.org/wiki/On-board_diagnostics [Port OBD-II] 10/02/2020
- [19] http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2UG13.pdf [Guia d'usuari PiCAN2] 22/04/2020
- [20] <https://www.kernel.org/doc/Documentation/networking/can.txt> [Informació del SocketCan] 02/04/2020
- [21] <https://github.com/mguentner/cannelloni> [Informació i repositori cannelloni] 21/04/2020
- [22] <https://bitbucket.org/hardbyte/python-can/get/4085cffd2519.zip> [Repositori python-can] 23/04/2020
- [23] <https://python-can.readthedocs.io/en/master/listeners.html#can.Listener> [Informació Listeners python-can] 09/05/2020
- [24] <https://medium.com/@alpinoacademy/writing-custom-snort-rules-e9abe10932e1> [Regles Snort] 12/05/2020
- [25] <https://www.snort.org/faq/readme-filters> [Regles Snort] 12/05/2020
- [26] <https://blog.easysol.net/machine-learning-algorithms-3/> [Explicació K-means] 24/05/2020