Neumann János
Informatikai kar

DIPLOMAMUNKA

OE-NIK Hallgató neve: **Árvai Péter**
2022 Hallgató törzskönyvi száma: **T/005508/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Árvai Péter**
Törzskönyvi száma: T/005508/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Autonóm jármű biztonsági tesztelése
Security testing of an autonomous vehicle

Intézményi konzulens: Rigó Ernő
Külső konzulens:

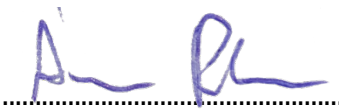
Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika specializáció

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.05.14.



.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Árvai Péter..... [redacted] ..... Mérnök informatikus BSc.

Telefon: Levelezési cím (pl: lakcím):

[redacted].....

Szakdolgozat / Diplomamunka[1] címe magyarul:

Autonóm jármű biztonsági tesztelése.....

Szakdolgozat / Diplomamunka[2] címe angolul:

Security testing of an autonomous vehicle.....

Intézményi konzulens: Külső konzulens:

Rigó Ernő..... Rigó Ernő.....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.03.04.	Kezdeti lépések megbeszélése	
2.	2021.04.06.	SZTAKI-ARNL kick-off meeting	
3.	2021.04.27.	Irodalomkutatás eredményeinek átbeszélése	
4.	2021.05.05	Élő ismerkedés a rendszerrel	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.05.14

.....
intézményi konzulens

1 Megfelelő aláhúzandó!
2 Megfelelő aláhúzandó!
3 Megfelelő aláhúzandó!
4 Megfelelő aláhúzandó!

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:
Árvai Péter..... [redacted] ..... Mérnök informatikus BSc.
Telefon: Levelezési cím (pl: lakcím):
[redacted].....
Szakdolgozat / Diplomamunka[1] címe magyarul:
Autonóm jármű biztonsági tesztelése.....
Szakdolgozat / Diplomamunka[2] címe angolul:
Security testing of an autonomous vehicle.....
Intézményi konzulens: Külső konzulens:
Rigó Ernő..... .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.16.	Teszt környezet tervezése	
2.	2022.04.20.	Malidar támadó szoftver követelményeinek meghatározása	
3.	2022.05.03.	Tesztek elvégzése, értékelése és javaslatok kidolgozása	
4.	2022.05.13	Utolsó átnézés leadás előtt	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.05.13

.....
intézményi konzulens

1 Megfelelő aláhúzandó!
2 Megfelelő aláhúzandó!
3 Megfelelő aláhúzandó!
4 Megfelelő aláhúzandó!

Tartalomjegyzék

1. BEVEZETÉS

Napjaink technológiai fejlődése az autóiparra is kiterjed. Az autók minél magasabb szintű automatizálása nem csak az utasok kényelmét, de biztonságát is szolgálja. A modern autókba beépített innovatív megoldások lehetővé teszik az utasok magasabb szintű informáltságát, biztonságát és kényelmét. Viszont a növelt automatizáltság új, eddig nem létező biztonsági réseket is jelenthetnek. A Széchenyi István Egyetem Járműipari Kutatóközpont és a SZTAKI létrehozott egy járműfejlesztési platformot, melynek célja autonóm járműirányítási funkciók megtervezése, különféle megoldások tesztelése, illetve jármű-jármű közötti kooperatív kommunikációs stratégiák kifejlesztése. Ezen szakdolgozatnak a célja ennek a platformnak a vizsgálata kíberbiztonsági szempontból.

1. ábra. Nissan Leaf autonóm platform

2. ELTERJEDT JÁRMŰ KOMPONENSEK

1992-ben a NASA felvetette a kíber-fizikai rendszerek (CPS) fogalmát, mely rendszerek magukba foglalják a számítás, kommunikáció és vezérlés elemeit. A CPS keretrendszer három szintre bontva csoportosítja a komponeseket: az érzkelés és végrehajtás réteg a rendszerek szenzorait és aktuátorait tartalmazzák. Az applikációs és vezérlő réteg főképp a felhasználói interakciókért felelős, és a két réteget az adatátviteli réteg köti összse. Habár ez a keretrendszer nem kifejezetten a járművek felépítésére lett létrehozva, azokra is igaz.

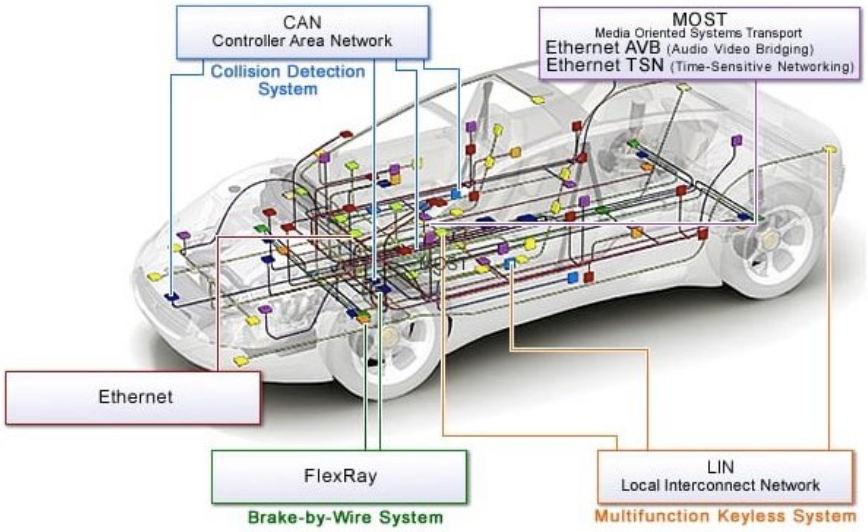
2.1. Elektromos vezérlők

Manapság a járműipari fejlesztések csaknem 90 százalékának valamilyen részét képzik az elektronikai komponensek [26]. Digitális alkatrészek először az 1970-es években jelentek meg a járművekben a fogyasztás optimalizálása érdekében, gazdasági, politikai okok, és a környezettudatoság erősödésének hatására. A motorszabályozó egység a kipufogógázok vizsgálatával képes hatékonyabb égés elérésére, az égésterekbe fecskendezett üzemanyag és levegő arányának szabályozásával [17]. Eleinte csak egy központi ECU végzett minden szabályzó feladatot, manapság már viszont egy olcsóbb árkatekóriás autó is 30–50 vezérlővel van felszerelve. Az idők során a beépített biztonsági, kényelmi, és szorakoztató funkciók egyre növelték a szabályzó elektronikák számát, egy 2009-es években vásárolt prémium kategóriás autóban körülbelül 100 millió sor kód futott, százszorosa egy F-22 Rapotor vadászgépnek [10].

2.2. Szenzorok és aktuátorok

A járművekbe beépített legtöbb szofisztikált funkcióhoz szükséges, hogy az autó képes legyen a környezetével kapcsolatba lépni, arról információkat gyűjteni, a környezetét fizikai úton befolyásolni. Ezekre a feladatokra a járműveket különböző típusú szenzorokkal és aktuátorokkal látjál el. A sávtartó, a dinamikus sebességtartó, vagy az egyéb sofőrt segítő alkalmazások elengedhetetlen részei a szenzorok. Ha a jármű nem képes a sávszéleket jelölő útburkolati jelek, vagy a saját, és előtte haladó autók sebességeinek mérésére, ezeket a funckiókat lehetetlen kivitelezni. Továbbá az utasok biztonsága érdekében gyakran fizikai közbeavatkozás is szükséges, például a légzsákok, biztonsági övek aktiválásával. Az aktuátorok kényelmi, információvisszacsatoló funckiókat is elvégezhetnek. Az újabb járművekben például a kormánykerék nincs mechanikai összeköttetésben a kerekekkel, ezáltal a sofőr kevésbé kap visszajeltést azon keresztül, nem "érzik" azt. Magasabb kategóriás autókban megjelentek a különböző vezetési módok (sport, comfort), melyek közt váltogatva, a kormánykerékbe szerelt motor változtatni tudja a vezetés élményét.

2.3. Automotív fedélzeti hálózatok



2. ábra. Jármű fedélzei hálózatainak eloszlása [1]

2.3.1. Hálózati topológiák

A számításra képes elektronikák között fontos szerepet tölt be az információmegosztás, ebből kifolyólag ritka eset manapság ezeket magányosan alkalmazni. A számítógépeket (node-ok) azonban többféle képpen is össze lehet kötni, és különböző topológiák különböző előnyökkel, és hátrányokkal járnak.

A járművek fedélzetén is gyakran alkalmazott busz topológia az egyes ECU-kat egy darab busz vezetékre köti, ezen keresztül folyik minden kommunikáció. Ennek előnye, hogy a hálózat viszonylag egyszerű, és az ára is alacsony a felhasználandó kábelek hossza miatt, ezért ideális a rengeteg fedélzeti ECU összekötésére, például egy központi gerinc-buszon keresztül. Azonban, ha a központi busz vezeték meghibásodik, az egész hálózat kiesik, továbbá mivel az összes forgalom ezen halad át, az adatátviteli sebesség viszonylag alacsony, és valamilyen megoldás szükséges a kommunikáció irányítására (CSMA, TDMA).

A gyűrű hálózati felosztásban a node-ok egymás után vannak csatlakoztatva, valamint az első, és utolsó node is összeköttetésben áll, így alkotva meg a gyűrű formát, akár csak pontok egy körvönalon. A kommunikációs csomagok a szomszédos node-ok között, általában egy irányba haladnak, kétirányú kommunikációhoz újabb vezetékek szükségesek. A gyűrű topológiának, és az egyirányú adatáramlásnak köszönhetően a csomagütközések redukálódnak, illetve az átviteli sebesség a buszokhoz képest magasabb. A felépítés hátulütője, hogy ha egy node is kiesik, az az egész hálózatot érinti, illetve újabb node-ok hozzáadásához meg kell bontani az összeköttetéseket, ami gátolja a működést.

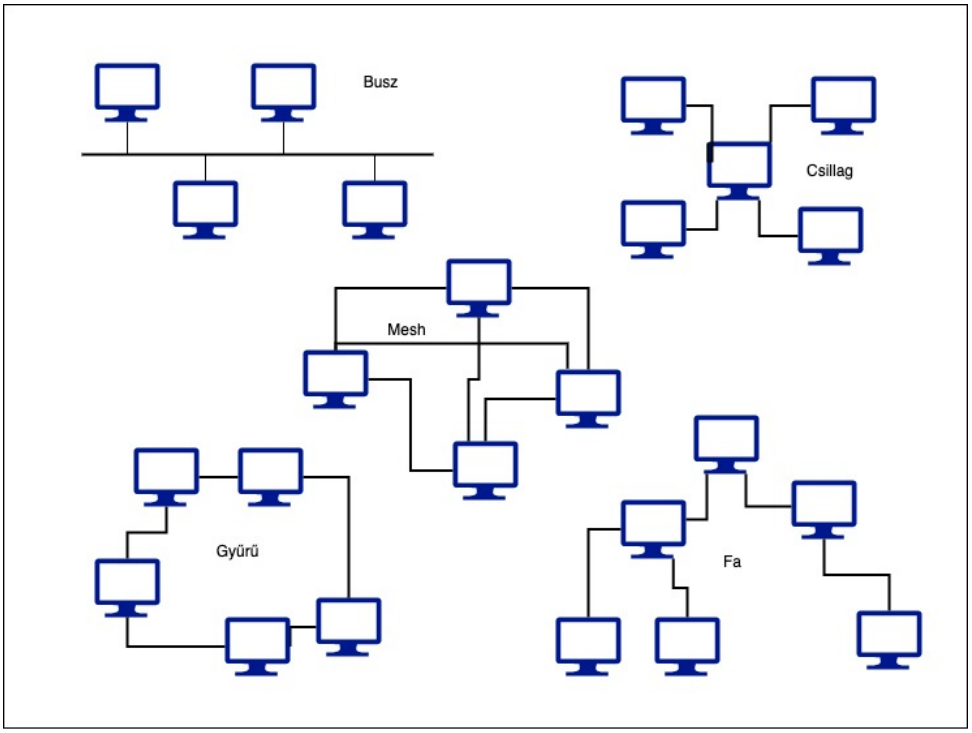
A csillag topológia napjaink legelterjedtebb hálózati felosztása. Lényege, hogy a minden node egy központi hubhoz csatlakozik, legyen az switch, vagy másik node. A

központi elem szerverként üzemel, és a beérkező csomagok továbbítását felügyeli. A hubok lehetnek passzívak, amik a beérkező csomagokat mindenféle feldolgozás nélkül broadcastelik minden irányba, illetve aktívak, melyek képesek a csomagokat, jeleket újra generálni, sőt bizonyos sérült packetek javítását is elvégzik. A topológia előnye a gyors kommunikáció, a központi hub általak biztosított könnyű üzemeltethetőség, a gyűrű felosztáshoz képest a kieső node-ok nem befolyásolják a hálózat többi részét, és a hálózat bővítése, vagy node-ok kivétele sem állítja le a kommunikációt. Hátránya azonban, hogy ha a központi elem kiesik, akkor az egész hálózat leáll, továbbá az adatátviteli sebesség a node-ok szaporodásával csökken.

A mesh topológiában minden node a hálózat összes többivel elemével összeköttetésben van. Ez adatátviteli, biztonsági, valamint megbízhatósági szempontból is kedvező, hiszen ha bármelyik node kiesik, a többi zavartalanul tovább tud kommunikálni. A csomagok küldésére két lehetséges módszer van: a routing, ahol valamilyen logika alapján (például legrövidebb út) megadott útvonalon haladnak a csomagok, és a flooding, ahol az egész hálózatot elárasztják azok, habár ez a performanciára is kihatással van. Ez a megoldás nem a legkedvezőbb, a nehezebb konfigurálhatóság, és a rengeteg vezeték, és egyéb hardware szükségletek miatt.

A fa felépítés egy hierarchikus modell, ahol minden node-nak több gyereke, és egy ős node-ja lehet. Ez alól csak a root node kivétel, aki a hierarchia tetején áll, neki őse nincs. A hálózat könnyen telepíthető és üzemeltethető, azonban a root node kiesésével az egész hálózat meghal.

A fentebb tágyalt hálózati topológiák kombinációjával úgynevezett hibrid hálók is létrehozhatók, melyek nagyon rugalmasak, azonban komplexitásuk miatt bonyolultabb a tervezésük.



3. ábra. Hálózati topológiák

2.3.2. CAN

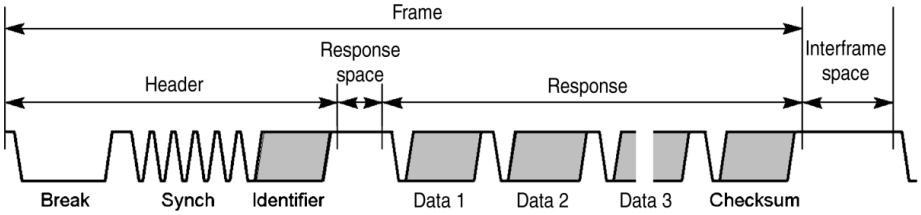
Komplex funkciókhoz az ECU-k között szoros együttműködésre van szükség, ezért a járműveket kommunikációs hálózatok szövik át. A különböző hálókat funkció szerint választják meg, melyeket Wolf és társai [28] öt csoportba osztva részletez. A legelterjedtebb fedélzeti hálózat a CAN (Controller Area Network), melyet a BOSCH fejlesztett ki. A hálózat az autóiparból ered, ahol az egyre gyarapodó vezetékkötegeket kívánták minimalizálni. A CAN egy ISO definiált soros kommunikációs standard, ahol a hálózat részeit képző ECU-kat (node) egy vezetékpárból felépülő busz köt össze. Az elektromos zavarok elleni immunitásra, a másodpercenként egy megabit adatátviteli sebességre, és a hibás adatok felismerésére és javítására képes hálózat manapság már az autóiparon kívül is elterjedt. A CAN egy multi-master hálózat, tehát nincs kijelölt vezérlő node, ami felügyelné a kommunikációt. Erre a célra CSMA (Carrier Sense Multiple Access) protokollt használ a busz egyidejű használatának elkerülése érdekében, minden node-nak várnia kell üzenetek küldése előtt, amíg fel nem szabadul a busz. Egyszerre küldött csomagok üztközésének feloldásához a standard egy prioritást jelző mezőt használ azok headerjeiben, ahol a magasabb prioritást élvező csomagok vehetik igénybe először a buszt. A protokoll négy frame típust definiál: adat frame adatok küldésére, remote frame adatok gyűjtésére, hiba frame, valamint overload frame adat, vagy remote frame-ek késleltetésére. [12].

A CAN adat frame az egyetlen módja az adatok közlésének, és az alábbi mezőkből épül fel: az első bit az úgynevezett state-of-frame (SOF) az üzenet kezdetét jelzi. Ezt követi a 11 bit hosszú SID (Standard IDentifyer), ez határozza meg az üzenetek prioritását. A SID csak az alap frame-ek esetén 11 bit hosszú, létezik egy kiegészített (extended) változat, ami extra 18 bitet használ. Az RTR (remote transmission request), amennyiben ez 0 értéket vesz fel azt jelzi, hogy a jelenlegi egy adat frame, 1-es érték esetén pedig remote frame. Ezt követi egy identifier extension bit, amihez a standard 11 bit hosszú SID esetén 0 értéket kell társítani, extended adat frame esetén pedig 1-et. Az RB0 bit konvenció alapján 0 értékkel kell, hogy rendelkezzen. Az RTR, IDE, valamint RB0 biteket vezérlő biteknek is nevezik. Ezeket követi a data length code (DLC), mely az azt követő adatok hosszát bájtok számában adja meg, ami nullától nyolc bájtig terjedhet. A fogadó node-ok a CRC (cyclic redundancy check) biteket felhasználva ellnőrizhetik a beérkező adat validitását. A következő mező (ACK) a validáció eredményét jelzik. Az ACK frame első bájtja alatt a küldő fél 1-es értéket ír a buszra, amennyiben a validáció sikeres volt, a fogadó felek azt lehúzzák 0-ra. Amennyiben ez nem történik meg, a küldő node észleli, hogy hiba történt, és később újból próbálkozik a küldéssel. A frame-ek végét a 7 bit hosszú EOF (end of frame) mező jelzi, aminek az értékei csak 1-esek lehetnek.

2.3.3. LIN

Kisebb, önálló hálózatokhoz - mint az elektromos ablakok, tükrök, vagy központi zár - LIN-t (Local Interconnect Network) alkalmaznak, mely egy olcsó, alacsony sávszélességgel rendelkező CAN alternatívaként lett kifejlesztve. Az egyszerű, alhálózatokra kifejlesztett LIN egy darab master, és maximum 16 darab slave node-ból

épül fel. Ebből a szerkezetből adódóan nincs szükség a CAN hálózatokban alkalmazott csomag ütközések elkerülésére. A LIN frame (4) egy header-ből és válaszból áll. A kommunikációt a master kezdeményezi, ami előre definiált ütemezés alapján headereket küld a buszra, amikre aztán a slave-ek válaszolnak.



4. ábra. LIN frame [2]

A headerek első mezője a sync break, ami a slave node-oknak ad jelzést az üzenet kezdetéről. Ezt követi egy sync mező, aminek előre meghatározott értéke van: 0x55 (binárisan 01010101). Ennek a váltakozó jelnek a segítségével a slave node-ok, az emelkedő és eső élek elemzésével, újra tudják számítani a buszon használt baud értéket, tehát az adatátvitel sebességét, így a node-ok összhangba kerülnek. Ezt követi egy azonosító, amiket a node-ok feldolgozva a következőket hajthatják végre: inaktívak maradnak, másik node-ok általk küldött üzeneteket fogadnak, vagy üzeneteket küldenek a buszra. Egyszerre csak egy node küldhet üzeneteket, amik 2-től 8 bájt hosszú adat mezőből, illetve egy 8 bájtos checksum mezőből állnak. A checksum mező az adatok validitásának ellenőrzésére szolgál. Az alap checksum mező csak az adatokra vonatkozik, viszont a második generációs LIN protokoll ehhez a headerben található azonosítót is hozzá csatolja [2].

2.3.4. FlexRay

A FlexRay kommunikációra két csatornát használ (A és B), és csatornánként 10 Mbit/s adatátviteli sebességre képes. Egy node csatlakozhat csak egy, vagy akár mindkét csatornához is, és a hálózat többféle topológia alapján is elrendezhető (aktív- vagy passzív csillag, busz, és hibrid topológiák), ezáltal rengetegféle szituációban alkalmazható. A FlexRay-t gyakran a járművek gerinchálózataként is alkalmazzák. A CAN prioritás alapú csomagküldési mechanizmusából adódóan bizonyos üzenetek késve vehetik igénybe a hálózatot, ezért biztonságkritikus területeken - például Drive-by-Wire - nem alkalmazható. Erre nyújt megoldás a FlexRay, és az által alkalmazott TDMA (Time Division Multiple Access), aminek az alapjait állandó, általában egy- és öt milisecundum közötti hosszal rendelkező kommunikációs ciklusok alkotják. Egy ciklus 4 szegmensből épül fel: statikus szegmens az állandó , determinisztikus adatok részére; dinamikus szegmens a CAN-hez hasonló nem determinisztikus adatoknak; hálózat kezelésére fentartott szegmens, illetve egy szünet, ami elősegíti a node-ok közötti szinkronizációt. A statikus szegmens előre definiált cellákra vannak osztva, amikből minden node egyet-egyet kap. A node-ok csak a saját cellájuk idejében küldhetnek adatot a buszra, amennyiben ezt nem teszik meg várniuk kell a következő ciklusig. A FlexRay ritkán küldött adatok számára tartja

fent a dinamikus szegmenst. Ez minicellákra van osztva, melyeket a node-ok szükség szerint lefoglalhatnak, mely ekkor teljes méretű frame-mé növekszik. A szegmens hossza állandó, így csak bizonyos mennyiségű adat fér el benne, és a CAN-hez hasonlóan prioriás alapján dől el, hogy egy adott frame kap-e helyet. Amennyiben betelt a szegmens, a node újrapróbálkozhat a következő ciklusban.

A node-ok a rendelkezésükre álló cellákat megadott frame formátum alapján tölthetik fel. Az adatok a payload mezőben, maximum 254 bájtnyi helyet foglalhatnak el, melyet footer mező követ három nyolcbites CRC értékkel ellátva az adatok validitásának vizsgálatára. Minden frame elején 5 bájt header kap helyet. Az első 5 bit különböző státuszokat jeleznek, például ha a payload mező nem tartalmaz adatokat. Ezt követi a 11 bit hosszú frame ID, amit minden node a hálózat tervezése során kap meg, valamint a cellakiosztások is ezek alapján történnek. Ez után a payload mező hossza található, amit a header ellenőrzésére használható CRC követ. Végül, a payload kezdete előtt, egy 6 bites ciklus számláló található.

A FlexRay legalapabb időegysége a microtick, amit minden node a saját belső oszcillátora alapján határoz meg, tehát ez node-onként egyedi. Ebből származik le a macrotick, ami a rendszerszintű időzítésekért felelős. Egy macrotick mindig microtickek egész többszöröse, viszont fontos, hogy ez szorzó node-onként eltérhet. A macrotick értéke futás alatt változhat, amit egy szinkronizációs algoritmus állít [25].

2.3.5. MOST

A különböző multimédiás eszközök autókba szivárgása megteremtette a szükséget a nagy sávszélességű hálózatokra. A MOST (Media Oriented System Transport) egy ISO/OSI standardot követő, száloptikás hálózat, infotainment rendszerek közötti kommunikációra, ami akár 150 Mbit/s-es sávszélességre is képes (MOST150). A hálózatra maximum 64 node-ot lehet gyűrű, csillag, és busz topológiákban csatlakoztatni. A MOST rendszert két szempont alapján tervezték: egyszerű, funkcióorientált felépítés, melynek eredményeként megszülettek az úgynevezett funkciós blokkok. Továbbá a rendszernek képesnek kell lennie csomag alapú, streamelt, és vezérlő adatok továbbítására is, amit a frame-ek struktúrájával valósítottak meg.

A MOST rendszerek alapját a funkciós blokkok képzik, melyek az OSI modell legfelső, applikációs szintjén helyezkednek el, és egy interface-t definiálnak, melyen keresztül az adott MOST eszközt lehet vezérelni. A funkciós blokkok, a programozási nyelvekből ismert tulajdonságokat, illetve függvényeket definiálnak. Egy CD lejátszó esetén például tulajdonság lehet a lejátszandó szám, függvény pedig a CD olvasó kinyitása/becsukása.

A funkciós blokkok használatának szemszögéből a MOST három szerepet különböztet meg. A legalsó szinten a slave-ek találhatóak. Ezek egyszerű MOST eszközök, melyek blokkjaikon keresztül szolgáltatják funckionalitásukat. Ezeket a funkciós blokkokat a controllerek irányítják komplexebb feladatok elérése érdekében, például a rádió vezérelheti a CD lejátszót, illetve az erősítőt is. Ezek mellett a controllereknek is lehetnek saját funkciós blokkjaik. A hierarchiában legfelül a felhasználói interakcióra alkalmas HMI (human machine interface) helyezkedik el, melyen keresztül a felhasználók, némi absztakción keresztül vezérelhetik, konfigurálhatják a MOST rendszert.

A MOST data link réteg definiálja a kommunikációhoz használt frame-ek struktúráját. A FlexRay-hez hasonlóan a MOST két csatornát definiál: egyet adatok szinkron streamelésére, egyet adatcsomagok aszinkron küldésére, viszont eltérően egy harmadik, vezérlő jelek számára elkülönített csatorna is létezik [3].

2.3.6. Bluetooth

Bluetooth rövid hatótávolságú összeköttetést biztosít hordozható eszközök számára. A technológia a kedvező ára, és sokoldalúsága miatt széleskörben elterjedt, és a mai autók infotainment rendszereinek alap felszereltségét képzi. A járművekben a Bluetooth nem tölt be bitzonságkritikus szerepet, főleg a fedélzeti rendszerek, és a felhasználók eszközei között igyekszik elmosni a határt, például egyes telefonos funkciók vezérlésének az autó felhasználói felületére történő átruházásával. A Bluetooth a 2.4 GHz ISM spektumban bocsát ki rádiójeleket másodpercenként 1 Mb adatátviteli sebességgel, melyet FHSS-sel (frequency-hopping spread spectrum), a követítésre használt rádió freqvencia gyors - másodpercenként 1600 - váltogatásával tesz ellenállóbbá zavarjelek ellen. Minden Bluetooth eszköznek van egy gyárilag beleégetett IEEE típusú, 48 bites azonosítója, és egy belső órája, ami másodpercenként 3200-szor üt (frekvencia váltások között kétszer). Bluetooth eszközök piconetekbe rendeződve kommunikálnak, melyek egy master, és maximum 7 slave eszközből állnak. Ezek a szerepek nincsnek beleégetve az eszközökbe, piconetenként változhatnak. A master és slave eszközök a kommunikációs médiumot TDD (time-division multiplex) alapján osztják fel: a master a páros, míg a slave-ek a páratlan sorszámú idősávokban közvetíthetnek [8][20].

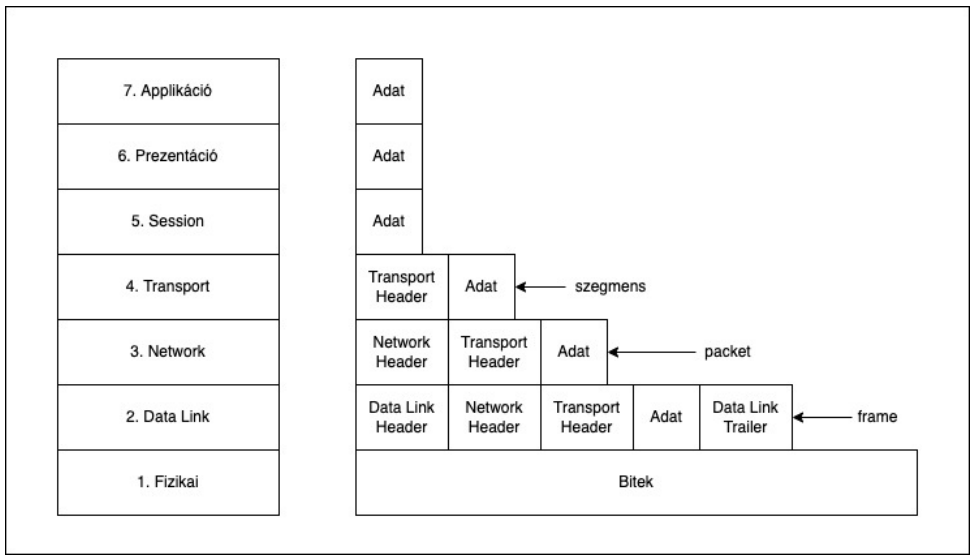
2.3.7. ISO/OSI modell

Az OSI modell (Open Systems Interconnection) egy univerzális elméleti modell, mely a telekommunikációs és számítogépes kommunikációt standardizálja, bármiféle technológiai konkrétum előírása nélkül. Habár a modell csak az elméleti síkon létezik, fontos megemlíteni, hiszen rengeteg konkrét kommunikációs protokollnak az alapját képzi. A modell az adatok folyamát hét rétegre bontja.

A legfelső, hetedik az applikációs réteg. Ez jelenti az üzenetek kiindulási- és végpontját, a felhasználói programok ezen keresztül tudják a hálózati funkciókat használni. A hetes rétegben alkalmazott funkciókra példa a file megosztás, vagy weboldalag böngészése. Egy szinttel lejjebb található a prezentációs réteg, amely az üzeneteket az applikációs réteg által definiált formára hozza, valamint titkosítási, tömörítési, és operációs rendszerek közötti inkompatibilitások átfedését végzi. Ez alatt található a session réteg, ami a helyi (local) és a távoli (remote) kommunikációs fél közti "session" felépítéséért, fenntartásáért, végül lebontásáért felel. Ebben a rétegben helyezkednek el a tartománynévrendszer protokollok, mint például a DNS. A stack negyedik szintjén a transport réteg található, ami a változó méretű adatok küldését látja el. Ennek érdekében szükség szerint azokat feldarabolja, ezt másnéven szegmentációnak is hívják. A transport réteg protokolljai lehetnek kapcsolat-orientált, vagy kapcsolat nélküliek, amikre jó példa a TCP és az UDP protokoll (habár azokat a TCP/IP stack definiálja). A kapcsolat-orientált proto-

kollok különböző mechanizmusok segítésével (TCP handshake) látják el az üzenetek megbízható, veszteségmentes küldését, ezzel némi gyorsaságot veszítenek a kapcsolat nélküli protokollokkal szemben. A hálózati (network) réteg az üzenetek címzetthez való eljuttatását végzik. A transport rétegben keletkezett szegmenseket további, kisebb elemekre, packetekre bontja, és számukra a megfelelő útvonalat kiválasztva (routing) felcímzi azokat. A data link réteg, hasonlóan a hálózati réteghez, a packetek címzésével foglalkoznak, azonban csak a lokális hálózaton belül. Erre a hálózati node-okba égetett MAC címeket használják. A legalsó a fizikai réteg, ami a digitális adatok és kimenő/beérkező jelek közötti konvertálásáért felelős. Többek között rendelkezik a használt logikai feszültségi szintekről, csatlakozókról, vagy közvetítési médiumról (kábel, optikai vezeték). A fizikai rétegben definiált ismertebb protokollok például a Bluetooth, vagy a CAN.

Az üzenetek a feladó számítógép applikációs rétegében keletkeznek, majd a stacken lefelé haladva a rétegek egymásnak adják azt. Egyes rétegek csak adatmanipulációval foglalkoznak, míg a transport, network, és data link rétegek headerek és trailerek formájában extra adatokat fűznek hozzájuk, mely folyamatot encapsulationnek nevezik. Az OSI modell frame-je a második rétegben áll elő, melyet a 5. ábra mutat. A címzett oldalán a beérkezett frame-ek az ellenkező irányba, felfelé haladnak át a rétegeken, míg az üzenet el nem jut a felhasználói szoftverhez.



5. ábra. OSI stack, és az előállított frame

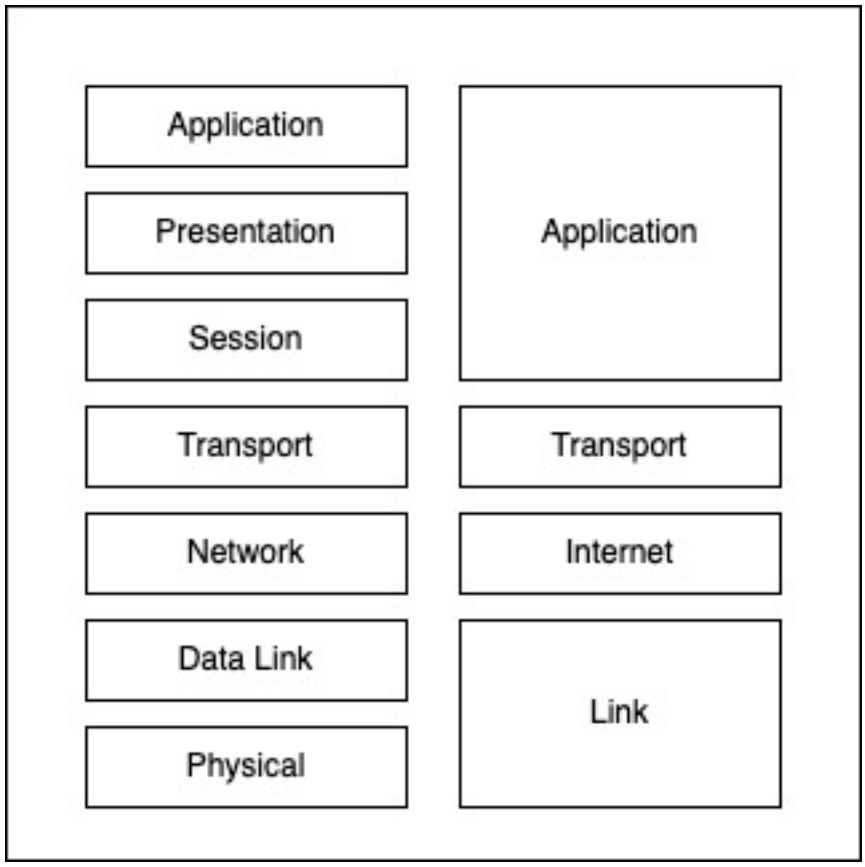
2.3.8. TCP/IP alapú hálózatok

A TCP/IP alapú hálózatok az internet szerves részeként a világ nagy részét átfedik. TCP/IP és a fentebb tárgyalt OSI modell stackjei nagyrészt megegyeznek, azonban a stacket hét helyett csak négy rétegre bontja. Ez annak az eredménye, hogy a session és prezentációs rétegek az applikációs réteggel összeolvadtak, míg a fizikai réteget a modell nem definiál a hardware függetlenség érdekében. A link réteg a TCP/IP stack legalsó rétege, mely nagyjából az OSI modell data-link rétegének felel meg. Az internet réteg felelős a csomagok kézbesítésének lebonyolításáért, amit

idegen szóval routingnak neveznek. Ebben a rétegben található az internet protokoll, melyek a routing szabályrendszerét, és a hálózatok node-jainak kiosztott virtuális címzést (IP cím) definiálja. A réteg a transport réteg felől beérkező adatokból úgynevezett datagrammokat csinál IP header hozzácsatolásával (encapsulation), ami többek között a feladó és a címzett IP címét tartalmazza. Ebben a rétegben található a hálózatok feltérképezésére használt ARP és RARP protokollok, melyek a fizikai (MAC) és a virtuális (IP) címek összeegyeztetését végzik, továbbá a közismert ping funkció által használt ICMP protokoll is.

A transport réteg a kommunikáló végpontok közötti adatátvitelről rendelkezik. A felek közötti összeköttetés lehet kapcsolatorientált, ilyen a transmission control protokoll (TCP), ami egy megbízható, hibák kezelésére alkalmas, adatátviteli csatornát biztosít végpontok között. Minden TCP kapcsolatot egy három lépéses kézfogás procedúra előz meg, melynek lényege egy megbízható kommunikációs csatorna létrehozása. A TCP-vel szemben a user datagram protokol (UDP), habár gyorsabb, de megbízhatatlan, üzenetorientált kapcsolatot definiál. Az UDP üzeneteket küldő applikáció nem törődik a fogadó féllel, nem alkalmaz a TCP-hez hasonló handshake eljárást, az elküldött packeteket nem vizsgálja, hogy megérkeztek-e. Ezért az UDP olyan területeken alkalmazható, ahol számít a gyorsaság, és némi adatvesztés is megengedhető, például internetes telefonhívásoknál (VoIP). Applikáció specifikus csatornák létrehozásának érdekében a réteg számozott portokat hoz létre, melyeknek egy része konvenció alapján le van foglalva, mint például a 22-es port az ssh kapcsolatoknak, vagy a 443-as port a https forgalomnak, 1024-nél nagyobb portok viszont szabadon használhatók. Az applikációs réteg az osi modell felső három rétegéhez hasonló feladatokat lát el, valamint itt vannak definiálva a felhasználói szoftverek által használt prokollok, mint a HTTP, FTP, vagy SMTP [9].

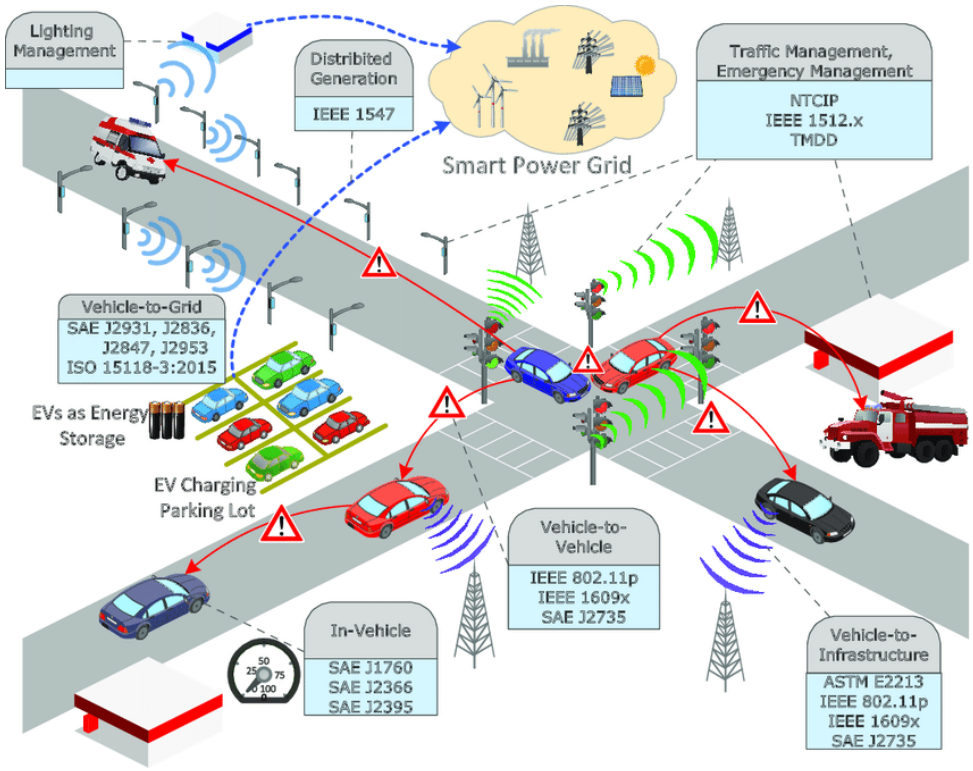
Habár a járműiparban kevésbé elterjedt, Rainer Steffen és társai megvizsgálták egy teljesen IP alapú járműhálózat lehetőségeit [26], és habár a magasabb követelményeket megkövetelő applikációk szükségeit is képes teljesíteni, nem minden ECU-nak van szüksége a teljes IP stack-re. Ezek mellett sok pozitív aspektusa is van az ilyen típusú hálóknak: leegyszerűsödik a fedélzeti kommunikáció, és a fejlesztés menete is, a már létező számtalan internetes technológiák és ismeretek felhasználásával.



6. ábra. OSI hálózat modell és TCP/IP stack

2.4. Járművek közötti hálózatok - VANET

A járművek technológia fejlődése nem csak azok fedélzetén jelenik meg. A gyorsabb, hatékonyabb, és nem utolsó sorban biztonságosabb közlekedés elérésének egyik kulcspontja az intelligens közlekedési rendszereket (ITS) alkotó a VANET (Vehicular Ad-hoc Network) hálózatok. A VANET hálók részeit képző járművek nem csak egymással, de az útmenti inftastruktúra elemeivel (RSU - road side unit), sőt még a gyalogosok mobil eszközeivel is képesek adatot cserélni, ami által egy sokkal részletesebb, és tágabb képet kapnak a környezetükről. A V2X (vehicle to everything) kommunikációra részben celluláris hálókat (főleg 5G), illetve az erre a célra kifejlesztett WAVE (Wireless Access in Vehicular Environment) technológiát alkalmazzák. A WAVE stack alapját a DSRC (Dedicated Short Range Communication) alkotja, aminek pontos alkalmazását a széles körben elterjedt WiFi standard kiegészítése, az IEEE 802.11p definiál. Habár a szabványok nagyrészt ki vannak dolgozva, a hálózat szokványostól eltérő természetéből adódó kihívások miatt, például a gyorsan változó topológia miatt, a V2X technológia még nem elterjedt [18].



7. ábra. Intelligens közlekedési rendszer (ITS) [27]

2.5. OBD II

A fedélzeti diagnosztikai rendszer (On-Board Diagnostics) beépítését elrendelő szabályzatok eleinte a járművek kibocsájtásának mérséklete volt a célja. A rendszert felhasználva évente tesztelték az autók kibocsájtását, és csak akkor engedték azokat vissza a forgalomba, ha megfeleltek az előírt követelményeknek, így ösztönözve a vásárlókat a könyezettudatosabb járművek megvásárlására. Mivel nehéz volt a különféle diagnosztikai rendszerekből a szabványosítás hiányában kiolvasni az adatokat, ez a terv becsődölt. Ezt kívánta orvosolni az OBD II, mely szabványosította a rendszerhez kapcsolható olvasó eszközök csatlakozóját, és az információk közlésére alkalmazott kommunikációs sémát. A fedélzeti diagnosztikai rendszer az autó csaknem egészéről képes diagnoszikai információkat szolgáltatni, tehát eléri a fedélzei hálózatokat, és ECU-kat, sőt nem csak azok olvasására, de írására is képesek (például firmware update). A rendszert legtöbbször az autószerelők használják, de léteznek lelkes rajongók, akik jobb teljesítmény érdekében tuningolják rajta keresztül autójukat. [22]

8. ábra. OBD II port és olvasó [4][5]

3. TÁMADÁSI VEKTOROK MODERN JÁRMŰVEK ELLEN

3.1. Jelenlegi biztonsági megoldások

Habár a múltban a járművekbe épített biztonsági mechanizmusoknak nem tulajdonítottak nagy szerepet, némi védelem megtalálható bennük. Biztonságkritikus ECU-kban megjelentek a kriptográfiai elemek a kommunikáció biztosításra, a fizikai manipuláció ellen is fel lettek vértezve, sőt a legmodernebb ECU-k a szoftver módosítást is detektálni tudják bootolás ideje alatt. A különböző buszokat összekötő gateway-ek az áthaladó csomagok szűrésére is képesek, így akadályozva meg a kártékony üzenetek terjedését. A teljes védetlenségnél jobb megoldások ezek, a legtöbb esetben azonban könnyen kijátszhatók.

3.2. A jármű fedélzeti rendszerei elleni támadások

Habár egy modern autóban különféle hálózatok sokféle feladatot látnak el, komplexebb feladatok miatt közöttük léteznek átjárási pontok (gateway), tehát szinte minden komponens csatlakozik a CAN, vagy FlexRay gerinchálózathoz, így annak a támadása és sikeres kontrollálása az egész rendszert megbéníthatja. A támadók dolgát megkönnyíti az is, hogy a hálózaton haladó üzenetek általában nincsenek titkosítva, és habár az üzenetek struktúrája nem ismert, és a gyártók sem teszik ezeket publikussá, azok tartalma dekódolható, és bizonyos autó modellek esetén magánszemélyek munkájának köszönhetően online megtalálható. További biztonsági rést jelent, hogy a hálózat node-jai nem képesek a beérkező üzenetek validitására. Ezeket a tulajdonságokat kihasználva hamis üzeneteket lehet a hálózatba juttatni, ahogyan azt Tobias Hoppe és társai is bemutatták tesztjeikben [15], ahol egy megfertőzött ECU 200 km/h sebesség felett CAN üzenetek injektálásával leengedte az ablakokat, majd a hálózaton blokkolta a további kommunikációt, így akadályozva meg az utast az ablak felhúzásában. Ahogy Hoppe-ék is bemutatták, a CAN hálózatok DoS támadások ellen sem nyújtanak védelmet. Erre a prioritás orientált CSMA/CD kommunikációs szabályrendszer kihasználásával van lehetőség, a csatorna legmagasabb prioritású üzenetekkel történő elárasztásával. A magas prioritásnak köszönhetően ezek mindig elsőbbséget élveznek, ezáltal a többi ECU üzenetei nem kerülnek továbbításra. A CAN autentikáció nélküli kommunikációját kihasználva új node-ok telepíthetők, melyek képesek lehallgatni, és befolyásolni az egész hálózatot, sőt a CAN protokoll automatikus hiba lokalizációs funkcióját kihasználva - ami a meghibásodott ECU-k leválasztását is támogatja - a támadók kedvükre alakíthatják a hálózatot. A CAN hálózat elérésének legkönnyebb módja az OBD II port. A múltban ehhez speciális eszközökre volt szükség, azonban az említett Európai szabályzatok hatására a gyártók átálltak a Windows-alapú PC-k támogatására, amihez OBD II interface adapter (PassThru), és speciális szoftver szükséges. Ezek a PassThru adapterek általában támogatják a vezeték nélküli csatlakozást, amihez egy saját WiFi hálózatot hoznak létre. Ezen eszközök megfertőzésével a támadók az autójavító műhelybe beérkező összes járművet meg tudják fertőzni, és ehhez még csak a folytonos jelenlétük sem szükséges [11].

A spoofing támadások a rendszer elemei közötti kommunikációs hálózatokat (CERT target) veszik célba, amihez szükség van egy elemre, ami a hálózat részét képzi, és amit a támadó kontrollál. Ez lehet egy kompromittált vezérlő egység, vagy az ODB II portba csatlakoztatott eszköz. Az általunk tesztelt autonóm platform bizonyos szenzorai TCP/IP alapú Ethernet hálózaton keresztül szolgáltatják a begyűjtött adataikat. A CAN-hez hasonlóan ezen a hálózaton is lehetséges az üzenetek (packetek) lehallgatása, kártékony packetek injektálása, továbbá a DoS támadásoknak is ki van téve. Habár az Ethernet hálózatok a forgalomban lévő autókban nem elérhetők, és így ezek a támadási vektorok éles támadásban nem alkalmazhatók, ennek ellenére az általunk végzett teszteknél hasznosak lehetnek.

Az ECU-kon futó szoftverek a memória hiányában nincsenek titkosítva, ezáltal az kiolvasható és elemezhető. Legtöbb esetben ezek a szoftverek low-level nyelven (C) íródtak, és azok elemzése támadható hibákat fedhet fel. Ezek a támadások azonban csak információ gyűjtésként, a támadások előkészítéseként szolgálnak, éles helyzetben nem alkalmazhatóak, mivel túl sok időt vesznek igénybe. ECU-k kiiktatására használható eszköz az EMP generátor, ami rövid, nagy energiájú elektromágneses hullámokkal süti ki az áramköröket. EMP generátorokat viszonylag olcsón lehet építeni, azonban ezek nem elég erősek ahhoz, hogy egész autókat hatástalanítsanak [Potential Cyberattack on Automated Vehicles].

3.3. Külvilág felől érkező támadások

Amíg a 3.2-es pontban tárgyalt támadások már a rendszer részeként kerülnek bevetésre, addig a kívülről érkezők célja irányulhat ennek a belső hozzáférésnek a megszerzésére.

A fedélzeti hálózatok támadásának egyik lehetséges módja a 2.5-as pontban tárgyalt OBD II interface. Szintén célpontok lehetnek azok az eszközök, amikkel maguk a felhasználók is kapcsolatban vannak, mint például a multimédia rendszer, ami több vezeték nélküli jelet (AM/FM rádió, Bluetooth) is fogad, CD olvasóval, USB bemenetekkel is rendelkezik, és legtöbbször a CAN (vagy az erre a célra fejlesztett MOST) hálózatra is fel vannak csatlakozva. Stephen Checkoway es társai cikkükben [11] tárgyalják ezek sérülékenységeit. A különböző ECU-kon futó szoftverek visszafejtésével kidolgoztak olyan módszereket, melyekkel az autó teljes kontrollját át tudták venni. Például a multimédia rendszer CD-lejátszó egységén sikeresen hajtottak végre buffer overflow támadást, és létrehoztak olyan CD-ket, amelyeket eljátszva azok a jármű CAN hálózatára publikáltak üzeneteket. Charlie Miller és Chris Valasek cikkükben [19] a Bluetooth-t jelölték meg az egyik legnagyobb támadási vektorként a modern autókban a protokoll komplexitása miatt, ráadásul a Bluetooth manapság az autók alap felszereltségének részét képzik, így a támadóknak megbízható kaput jelenthetnek a rendszerbe. Checkowaz és társai [11] egy jármű Bluetooth stackjének elemzésével sikeresen végrehajtottak RCE (Remote Code Execution) támadást. Ehhez azonban szükség van egy párosított Bluetooth eszközre (telefon, tablet), amire a szerzők két lehetséges kivitelezési útvonalat vázoltak fel: a könnyebbik megoldás az autó tulajának eszközeinek a megfertőzése, hiszen az már hívások fogadása, navigáció, vagy zene lejátszásához már jó eséllyel párosítva van a járművel. A második megoldás saját eszköz párosítása, aminek az első lépése a jármű Bluetooth címének

(48 bites ID) a feltárása. Minden párosítás megerősítéséhez a párosítandó eszközön meg kell adni a jármű fedélzeti kijelzőjén feltüntetett, négy random számból álló PIN kódot. A cikkben tesztelt jármű Bluetooth modulja azonban szoftveresen kezdeményezett párosítási kérelmekre is válaszol, ráadásul ez a folyamat teljesen a háttérben zajlik, a tulaj nem értesül a párosítás kezdeményezéséről, létrejöttéről. A támadás szoftveres jellegéből adódik, hogy az elvárt PIN kód ismerete nem fontos, azt ki lehet brute force-olni. A módszer nehézségét azonban az jelenti, hogy a Bluetooth stack válaszidejéből adódóan csak 8–9 próbálkozást lehet végrehajtani percenként, ami átlagosan autónként 10 órát jelent, ami alatt a járműnek működnie kell, és viszonylag közel kell tartózkodni hozzá.

Az autonóm rendszerek egyik nagy kihívása a különböző szenzorok által gyűjtött adatok fúziója, ami egyfajta szűrőként is bevethető a bejövő kártékony adatokkal szemben, azonban ehhez megfelelő szintű redundancia szükséges. Amennyiben az adatok csak két, vagy annál kevesebb forrásból származnak, és a források eltérő értékeket közölnek, a fúziót végrehajtó egység nem képes eldönteni, melyik a valódi, és melyik a hamis adat. Tehát az efféle biztonsági szűréshez legalább három eltérő adatforrás, és azok többségének a validitása (nem a támadó vezérli, valós adatokat prezentál) szükséges. V2X technológiával felszerelt autók a redundáns adatokat a közlekedésben résztvevő többi járművektől is fogadhatnak, habár azokat sosem lehet olyan megbízható forrásnak tekinteni, mint a fedélzeti szenzorokat, mivel a feladó identitásáról, vagy annak validitásáról nincs adat. Az általunk tesztelt platformon több LIDAR, és GPS modul is található, azok azonban különböző funkciók miatt kerültek fel a járműre, például a LIDAR-ok esetében mindegyik az autót körbevevő térnek különböző szeletét fedik le; az autó előtt és mellett érzékelnek. A GPS esetén adott a redundancia, azonban platform csak két modullal van felszerelve, ami a pontatlanságok kiszűrésére alkalmas, a támadások hamis értékeire nem. A redundáns szenzorrendszer az autógyártóknak - és következetesen a vásárlóknak is - nagyobb költségeket jelentenek, ezáltal ezt a megoldást nem is alkalmazzák. A fedélzeti szenzorok ellen gyakran bevetett támadás a jamming. Ehhez a szenzorokhoz hasonló eszközzel, folytonosan üzenetek küldésével, vagy kommunikációhoz használt médium blokkolásával gátolják az adatok küldését, fogadását. Szintén elterjedt módszer még a spoofing, ahol a szenzorok a valóshoz hasonló, de a támadó által generált információkat fogadnak. A két technika gyakran együtt alkalmazandó, mint például az autók távoli kinyitására használt roll jam támadás. A tesz platform valamennyi szenzorán végre lehet hajtani legalább az egyik említett támadást. Jonathan Petit és társa [21] magas sikerességi valószínűséget társít a GPS szenzorokat érő jamming, illetve spoofing támadásoknak. Jamming eszközök már 20 USD érték körül is kaphatóak, amikkel az autótolvajok hatástalanítani tudják a lopásvédő rendszereket. A jamming támadások elleni védekezés nehéz feladat, hiszen ezeknél a szenzoroknál a környezeti akadályok miatt (alagút) megszokott a jel elvesztése. A piacon szofisztikáltabb, spoofing-ra is képes eszközöket is lehet kapni, ezek azonban jóval drágábbak. A környezet észlelésére használt kamerákat (RGB, LIDAR) nagyerejű infravörös fénnyel könnyel el lehet vakítani. Ilyen támadások ellen az optikai fényszűrő sem véd, egy katonai megoldás váltakozó színű laserekkel játszotta ki azokat. A RADAR szenzorok ellen egy lehetséges támadás keretein belül hamis rádiójele-

ket vetíthetünk a jármű irányába, amit aztán az valós akadályokként észlel, vagy folytonos sugárzás esetén a szenzor blokkolása is elérhető. Erre használható eszköz a DRFM ismétlő (Digital Radio Frequency Memory), ami a beérkező radar jeleket eltárolja, melyek igény szerint újra lejátszhatók. A jármű szenzorai az utasok információinak megszerzésére is használhatóak. Manapság az autógyártók kötelesek járműveiket TPMS (Tire Pressure Monitoring System) érzékelőkkel ellátni. Kutatóknak sikerül ezen szenzorok által sugárzott keréknyomás adatokból rekonstruálni a jármű által megtett útvonalat [14].

Az ITS (Intelligent Transport System) kulcsfontosságú elemei a járművekből, és az infrastruktúra elemeiből felépülő hálózatok, más néven VANET-ek (Vehicular Ad-Hoc Network). A járművek és az út menti eszközök kommunikációjának a feladata a közlekedéssel kapcsolatos - köztük biztonság-kritikus - információk megosztása, így ennek a hálózatnak a kontrollja kulcsfontosságú tényező a forgalom irányításához, és az utasok védelméhez. A [23] cikk szerint a sybil támadás az egyik legveszélyesebb a VENET-ek számára. A támadás lényege, hogy egy node (jármű) több identitást szimulál, amit a hálózat kedv szerinti alakítására használ, például más autók manipulálására, őt körbevevő szellem-autókkal, hogy az ütközés elkerülése végett az másik útvonalra térjen. A járművek és RSU-k közötti információcserét DoS támadásokkal lehet blokkolni. Ezáltal az autonóm rendszerek fontos inputoktól esnek el, aminek súlyos következményei is lehetnek. A MANET-ek (Mobile Ad-Hoc Network) ellen kifejlesztett JellyFish egyik példája a támadásnak, ami jól átültethető a járművek világába. A JellyFish a routing protokollhoz alkalmazkodik - ezáltal nehezebb a detektálása -, és a packetek fogadása, továbbítása közbe iktatott késleltetéssel zavarja a hálózat működését. A DoS támadásnak több fajtája is alkalmazható a VANET-ek ellen, például: flooding, jamming. A blackhole támadás a DoS támadáshoz hasonlóan az üzenetek átvitelét próbálja megakadályozni a hálózat alakításával. Ehhez a routing protokollt használja ki, és az üzenetek címzettjeihez a legrövidebb útként tünteti fel magát, hogy a környező járművek rajta keresztül küldjék packetjeiket, amiket aztán ő elvet. Az irodalomkutatás keretei között feltártuk a járművek különböző rendszereit biztonsági szempontból, a felsorolt technikák azonban csak egy részhalmaza a lehetséges támadási vektoroknak.

3.4. Elektromos járművekkel kapcsolatos támadások

Az elektromos és robbanómotoros járműveket érintő biztonsági vektorok között nagy az átfedés, azonban az elektromos autókban új biztonságkritikus elemek is megjelentek. Egy kritikus pontjuk az akkumulátor, melyek nagy feszültégekkel és áramerőségekkel operálnak, és a jelenleg elterjedt Litium Ion akkumulátor cellák fizikai sérülés, vagy nem megfelelő töltöttségi szint elérése esetén fel is gyúlhatnak, robbanhatnak. Az akkumulátorokat egy úgynevezett BMS (Battery Management System) vezérli, ami felügyeli és szabályozza a fogyasztás, töltés folyamatát, valamint egyéb biztonsági funckiókat is ellát, például monitorozza a hőmérsékletet. Az akkumlátorok idővel degradálódnak és cserét igényelnek, viszont a mobiltelefonokkal ellentétben a hamisított akkumulátor nagyobb veszélyt jelent.

Az akkumulátorok megjelenésével a töltő nyílások is megjelentek. Eleinte ezek csak energiaátvitelre voltak alkalmasak, azonban idővel kialakultak fejlettebb pro-

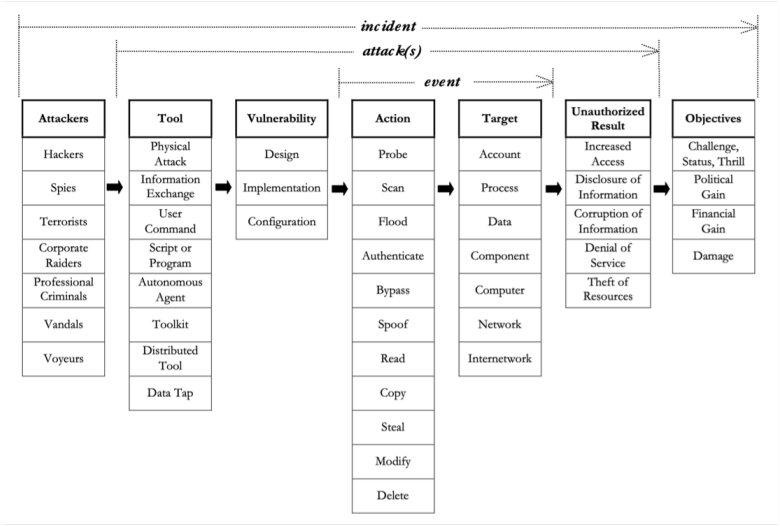
tokollok a BMS és a töltőállomás közötti kommunikációra. A jövőben ezeket a funkciókat multimédia streamelés, vagy firmware update is kiegészítheti, tehát egy rosszul implementált töltő csatlakozó az ODB porthoz hasonló veszélyeket rejthet, amihez ráadásul nem is kell a támadónak az autó utasterébe jutnia.

Elektromos járművek esetén a drive-by-wire funckciók biztonságára sokkal nagyobb figyelmet kell fordítani a konvencionális meghajtással rendelkezőkkel szemben, mivel a limitált hatótáv miatt azok jelentős mértékben hagyatkoznak a regeneratív fékezésre, tehát mechanikai kapcsolat a fékpedál és fékpofák között nem lehetséges, azokat teljes mértékben elektronikusan kell vezérelni. A drive-by-wire ECU-k biztonságára ebből a szempontból nagyobb figyelmet kell fordítani.

4. TÁMADÁSI VEKTOROK KATEGORIZÁLÁSA

4.1. CERT taxonómia

Számítógépes és hálózati incidensek csoportosítására, és megértésére alkották meg a CERT rendszertant [16]. Ahogy az a 9. ábrán is látható, a taxonómia legkisebb egysége az event. Általánosságban elmondható, hogy egy event valamilyen rendszer állapotának a megváltozását jelenti. Kiberbiztonság szemszögéből nézve ez bizonyos cselekedeteket (action) jelent, melyek célpontok (target) állapotát kívánják megváltoztatni. Például: actionnek tekinthető egy rendszerbe történő bejelentkezés, melynek a célpontja a felhasználói fiók. Fontos megjegyezni, hogy az event nem rendelkezik a támadás sikerességéről. A CERT taxonómia a támadásokat öt olyan logikai lépésre bontja, melyekkel a támadó jogtalan hozzáférést szerezhet egy rendszeren belül. Az eventek a rendszer sérülékenységeinek (vulnerability) kihasználásával jönnek létre, melyek kivitelezéséhez a támadó eszközöket (tool) használ. Egyes támadások jól megkülönböztethetők egymástól azok kivitelezője (attacker), a támadás karakterisztikái, időzítései, vagy elérni kívánt céljai (objective) alapján. Az ilyen, támadásokat logikai alapon megkülönböztető csoportokat a rendszertan incidenseknek nevezi.



9. ábra. CERT taxonómia támadási incidensek kategorizálására [16]

4.2. Failure Mode and Effect Analysis

Jonathan Petit és társa [21] Failure Mode and Effect Analysis módszerhez hasonló megoldást alkalmaztak a támadások kategorizálására, melynek lényege, hogy egy termék, vagy eljárás potenciális meghibásodási lehetőségei, azok következményei és tulajdonságai beazonosításra kerüljenek. Maga a támadás módszerén kívül figyelembe vették a támadás kivitelezhetőségét, ami a támadó képességein, és tudásán kívül a rendelkezésére álló forrásokat (szükséges eszközök elérhetősége) is magába

foglalja. Ezen kívül a fizikai hozzáférés szükségességét, a támadás észlelhetőségét (utas és rendszer által), a támadás az autóra jelentő következményeit, a létrehozott veszélyeket (közlekedési baleset), és a támadás sikerességének a valószínűségét is figyelembe vették, ami például egy könnyen kivitelezhető, de könnyen észre is vehető támadásoknál alacsony.

4.3. CIA triád

A CIA triád egy kíberbiztonsági modell, melynek segítségével rendszerekben támadási vektorok fedezhetők fel, valamint ezekre megoldást is nyújtanak. A betűszó elemei definiálják az auditált rendszerek azon tulajdonságait, melyeknek hiánya biztonsági kockázatot jelent. Titoktartás (confidentiality) az adatok védelmével foglalkozik, tehát a hozzáférés megtagadásával, vagy engedélyezésével autentikációs metódusok alkalmazásával. Titoktartás sokféle képpen támadható, például a jogosultság megszerzésével, vagy az adatok lehallgatásával. Ez utóbbi ellen kriptográfiai eljárások alkalmazása nyújthat védelmet, azonban nem minden esetben a techonlógiák hiánya okozza a sérükenységet, hanem emberi hibából adódnak (konfigurációs hibák).

Az integritás (integrity) az adatok megbízhatóságára, pontosságára, eredeti állapotaiknak megőrzésére törekszik. Fontos, hogy feldolgozás, közvetítés alatt ne sérüljön az adat, és akaratlan módosításokat ne lehessen végrehajtani. Ezt hozzáférések kezelésével, és rendszeres backupok készítésével lehet biztosítani.

A harmadik követelmény a rendelkezésre állás (availability), ami alapján az adatoknak mindig elérhetőnek kell lenniük az arra jogosult felek számára. Ezt a hardware és szoftver komponensek karbantartásával lehet elérni [13].

5. SÉRÜLÉKENYSÉGI TESZTELÉS

Habár a V2X kommunikáció, és a járműveket érő külső támadások fontos részét képzik az autonóm járművek biztonságának, a szakdolgozat a továbbiakban a járművek belső rendszereit kívánja vizsgálni, tehát feltételezzük, hogy már hozzáférést szereztünk a jármű fedélzeti rendszeréhez. Az autonóm platform tesztelését a fedélzeti szenzorok zavarása felől közelítettük meg.

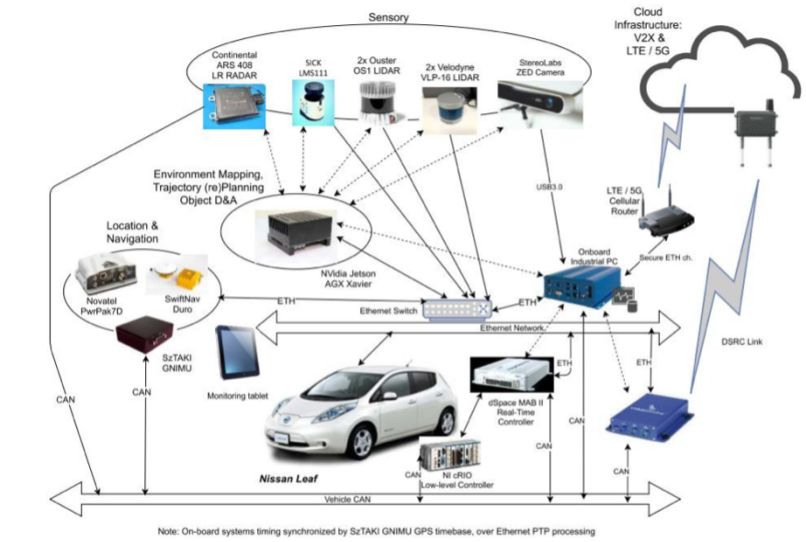
5.1. Tesztelés folyamata

A sérülékenységi tesztelés első lépéseként egy előkészületi elemzést hajtunk végre, aminek a célja potenciális sérülékenységek feltárása. Sajnos a működőképes autonóm platform Győrben, a Széchenyi István Egyetem birtokában van, így azon nem tudjuk elvégezni a vizsgálatot, csak a biztosított dokumentációk alapján elemezzük azt. Egy, a győrivel megegyező, második platform létrehozása is tervben van, habár ez még nem készült el, az egyes elemeit biztosították számunkra, amiket be tudtunk vonni az elemzésbe, és később a teszt környezetbe is. A feltárt sérülékenységek tudatában képesek leszünk megtervezni egy tesztelési környezetet, aminek támadásával igazoljuk (vagy megcáfoljuk) az elemzés során alkotott hipotézisünket. Továbbá, a kijátszható támadási vektorok ismeretében további eszköz, akár saját fejlesztésű malware szükségletek is körvonalazódnak. Végül, amennyiben sikerül kijátszani a rendszer biztonsági hibáit, javaslatokat teszünk azok orvoslására.

5.2. Referencia platform bemutatása

A teszteléshez használt platformnak egy Nissan Leaf elektrovoltaikus személygépjármű az alapja. Az autó a napjainkban az utakat járó járművekkel szemben nem egy önálló entitás, szenzorok segítségével képes feltérképezni a környezetét, rádiós kommunikációval képes más hasonló eszközökkel (járművekkel, infrastruktúrával) együttműködni, és a fedélzeti számítógépein a környezetről begyűjtött adatok alapján prediktív közlekedési stratégiákat alakít ki.

A járművön található gyári alkatrészek - szenzorok, aktuátorok, belső diagnosztikai rendszer - saját CAN-buszon kommunikálnak. Továbbá a járműre felszerelt külső érzékelők jelei (lokációs- és navigációs eszközök, LIDAR-ok, RADAR, IMU-k, és kamerák) a belső CAN-busztól független, saját adatátviteli hálózaton (Ethernet, soros kapcsolat, másodlagos CAN-busz) kommunikálnak, ezek feldolgozása közvetlenül a központi járműirányító rendszerben történik.



10. ábra. Az autonóm platform rendszerének felépítése

A fedélzeti rendszer további lényeges kiegészítése egy ITS V2X kommunikációs alrendszer, mely lehetővé teszi a forgalomban résztvevő járművek egymás, az infrastruktúra (például: jelzőlámpa), és a gyalogosok eszközei közötti adatátvitelt. Ez a V2X kommunikáció a jármű irányítása és prediktív pályatervezés szempontjából lényeges segítség, mivel így a fedélzeti szenzorok által gyűjtött adatok kiegészülnek más járművek által gyűjtöttekkel, ami lehetővé teszi például az útvonalak adaptív újratervezését, ha a jelenlegi tervezett haladási trajektória mentén baleset, vagy egyéb ok miatt dugó alakul ki. Az érzékelő és kommunikációs rendszereken túl egy újraprogramozható, konfigurálható HMI (Human Machine Interface) is található, mely a jármű teljes fedélzeti rendszerén mért jelek és adatok megjelenítését szolgálja. Ez a rendszer igény esetén kiegészíthető további információs és utasszórakoztató (infotainment) rendszerrel.

5.2.1. A jármű hálózati felosztása

A járműveket összekötő hálózatok - hasonlóan egy otthoni hálózathoz - két részre oszthatók: egyrészt belső hálózatra, amin a jármű belső alkatrészei kommunikálnak, illetve külvilágra, ahol a környezettel folyik a kommunikáció (V2X, 5G).

A Leaf képes a környezetével is kommunikálni. Erre a célra lett felszerelve egy CohdaWireless MK5 OBU (On Board Unit) V2X DSRC rádiókommunikációs egység, melyet egy RSU (Roadside Unit) egészít ki. A DSRC egy IEEE 802.11 szabványnak megfelelő kommunikációs protokoll, mely az OSI modell fizikai rétegében helyezkedik el. Ez az eszköz határfelületet képez a külvilág és a jármű fedélzeti rendszere között, így a biztonságban betöltött szerepe is nagy.

11. ábra. CohdaWireless V2X egység

A jármű komponensei között az információcsere CAN-buszokon történik. Az autó gyári szenzorait, aktuátorait, biztonsági rendszerét, és az alacsonyszintű járműirányítási rendszert a Leaf gyári CAN busza köti össze. A másodlagos CAN hálózat ettől elkülönülve működik, és az utólag felszerelt autonóm platform elemei használják. A két hálózat között létezik adatcsere, a bus gateway szerepet pedig a National Instruments cRIO9039 kontroller látja el. A járművön elhelyezkedő LIDAR szenzorok kommunikációja végett kialakításra került egy fedélzeti Ethernet hálózat is, ami két virtuális alhálózatra lett bontva. A virtuális hálózatokra azért volt szükség, mivel a LIDAR-okból érkező adatmennyiség egyéb eszközök számára nem hagyott volna maradék sávszélességet. A felsőszintű irányító rendszer USB interface-en is képes jeleket fogadni, azonban ezen módszerrel csak a Zed kamera van bekötve.

5.2.2. Fedélzeti alrendszerek felosztása

A Nissan Leaf fedélzetén az alacsonyszintű járműirányítást egy National Instuments cRIO real-time adatfeldolgozó FPGA alapú egység végzi. Ez a vezérlő fogadja a jármű irányításában résztvevő jeleket (féknyomás, gázpedál, kormánykerék nyomaték) a jármű gyári CAN buszán, és az irányításhoz szabályzóköröket (kerékszög, gyorsulás) is ez a hardver futtatja. Továbbá a jármű másodlagos CAN buszával is kommunikál, oda a magasabb szintű járműirányítási rendszerek számára publikál adatokat, illetve az autonóm működéshez használt jeleket is onnan kapja.

A NI cRIO az elvárt gyorsulási, és kanyarodási értékeket - melyekből aztán a gáz, fék, és kerékszög értékeket generálja - a dSpace MAB II Real-Time kontrollerből érkeznek. A MAB II egy autóipari gyártók által közkedvelten alkalmazott ECU, mely egy prototípusok gyártására alkalmas real-time adatfeldolgozó fedélzeti egység. A valós idejű vezérlő jelek mellett - megfelelő hardveres kiegészítésekkel - képes szenzoradatok (IMU-k, LIDAR-ok, V2X) kezelésére is. A MAB II az autonóm platform agya, ide érkeznek be a szenzoradatok, melyek feldolgozásával jön létre az elvárt gyorsulási, és kanyarodási jel. Bizonyos feladatok elvégzésére, ahol nem fontos az adatok real-time feldolgozása, egy ipari pc lett beszerelve a járműbe. Ide érkeznek be a környezetérzékelő alrendszer adatai utófeldolgozásra (és logolásra), melyeket aztán a MAB II a pályatervezésben fel tud használni.

12. ábra. National Instuments cRIO, dSpace MAB II

Az autonóm járművek navigálásában fontos szerepet tölt be a gépi látás, a környezetben található objektumok felismerésére, megkülönböztetésére. A készített képek alapján felismerhetők az útburkolati jelek, közlekedési lámpák, táblák, és egyéb releváns objektumok (járművek, gyalogosok, domborzat). A képek feldolgozása a képfeldolgozó alrendszeren, neurális hálók segítségével történik. A platform odometrikus képfeldolgozáshoz egy StereoLabs Zed sztereó kamerával lett ellátva, ami USB 3.0 interface-szel csatlakozik a képfeldolgozó alrendszerbe.

13. ábra. StereoLabs Zed kamera

A környezetérzékelő alrendszer további fontos része még az autóra szerelt LIDAR szenzorok, ami a tervezett útvonal mentén több irányból infravörös fénynyalábokkal képes letapogatni a környezeti elemeket (útpadka, gyalogosok, járművek). A Leafen 5 darab LIDAR egység található: kettő darab, egy tetőre szerelt konzolon vízszintes orientációban helyezkedik el; ez a jármű előtt és mellett térképezi fel a környezetet. A tető konzolon további két egység található az út síkjára 45 fokos elhelyezéssel az útszélek, járdaszegélyek észlelésére. Az ötödik szenzor az autó első lökhárítója magasságában helyezkedik el, és az említett szenzorok holttereit fedi le. Interface-ek szempontjából valamennyi LIDAR érzékelő rendelkezik ipari Ethernet porttal.

14. ábra. Velodyne PUCK16, Ouster OS1 64, Slick LMS111

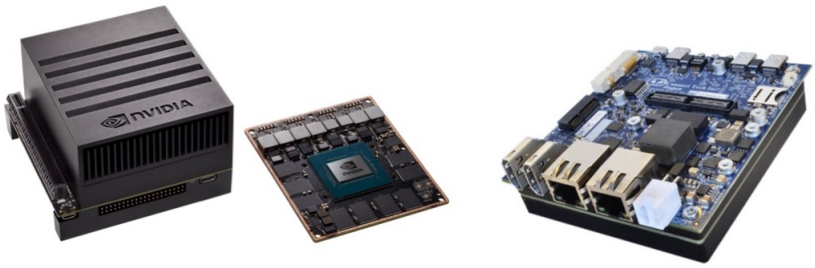
A platformon továbbá megtalálható egy Continental ASR-408 típusú, autóipari minősítésű RADAR szenzor. A RADAR technológia mikrohullámú rádióhullámok segítségével tapogatja le a környezetét, melyek segítségével meg lehet becsülni objektumok méretét, haladási irányát és sebességét. A szenzor adatok CAN buszon keresztül jutnak el a képiadat feldolgozó egységbe.

15. ábra. Continental ASR-408 RADAR szenzor

Pozícionálás és navigáció szempontból több szenzor is megtalálható a járművön: egy darab SwiftNavigation Duro GPS/GNSS, és egy darab Novatel PwrPak7D-E2 GNSS+IND modul. Ez a redundancia biztosítja a nagyobb hibatűrést, és a pontosabb szenzorfúziót. Az utóbbival USB, Ethernet és CAN buszon is lehet kommunikálni, míg az előbbi soros és Ethernet interface-ekkel rendelkezik.

16. ábra. SwiftNavigation Duro, Novatel PwrPak 7D-E2

A képfeldolgozó algoritmusok a LIDAR, RADAR, és a Zed kamera adatainak feldolgozása végett egy Nvidia Jetson AGX Xavier fejlesztő platformon futnak. A Jetson AGX Xavier egy Connecttech Rouge Carrier lappal lett bővítve, kiegészítve annak ki- és bemeneti interface-eit. A kialakított vezérlő modul képes az említett CAN, Ethernet, és USB kapcsolatok létrehozására is. A Jetson Xavier Ubuntu operációs rendszert futtat, melyen ROS, Autoware, és a SZTAKI által fejlesztett LIDAR pontfelhő feldolgozó szoftverek futnak.



17. ábra. Nvidia Jetson AGX Xavier, Connecttech Rouge Carrier

A biztonsági alrendszer a jármű rendszere és az alsószintű járműirányító rendszer közé épül be. Amennyiben a sofőr kívánja irányítani a járművet, a biztonsági rendszer a kezelőszervek (fék- és gázpedál, kormánykerék) jeleit veszi figyelembe. Autonóm üzemmód esetén ezeket figyelmen kívül hagyja, és az alacsonszintű járműirányító egység felől érkező jeleket engedi át. Továbbá képes még egy harmadik, vészállj funkcióra is, ami felmerülő hibák esetén biztonságos állapotba hozza az autót.

5.2.3. Feltárt sérülékenységi pontok

A bemutatott platform elemei között a kommunikáció titkosítatlan (CERT vulnerability), ebből kifolyólag nem csak az adatok kinyerése lehetséges, hanem a 3. pontban tárgyalt sérülékenységeknek szinte kivétel nélkül ki van szolgáltatva. Amennyiben a támadó fizikai eszköz beszerelésével, vagy egy fedélzeti ECU megfertőzésével képes saját kód futtatására, a jármű bizonyos részeit, vagy akár az egész rendszert az uralma alá vonhatja. Ebből a szempontból az autonóm platform leggyengébb láncszeme a másodlagos CAN hálózat, mivel azon az összes alegység adata áthalad.

A VLP16-os LIDAR különösképpen felkeltette a figyelmünket a biztonsági hiányosságaival. A szenzor egy webes interface segítségével, illetve curl utasításokkal is konfigurálható, azonban ezt a mechanizmust semmi nem védi, vele egy hálózaton bárki elérheti, és autentikáció nékül igénybe veheti. A későbbiekben ez a szenzor fogja a teszt környezet alapját képezni, részletesebben az 5.4.1 pontban lesz tárgyalva.

A platformon megtalálható ECU-kat erőforrások és beágyazott eszközök tesztelésének ismeretének hiányában részleteiben nem elemeztük, feketedobozként tekintve rájuk, csak a ki- és bemeneteiket vizsgáltuk.

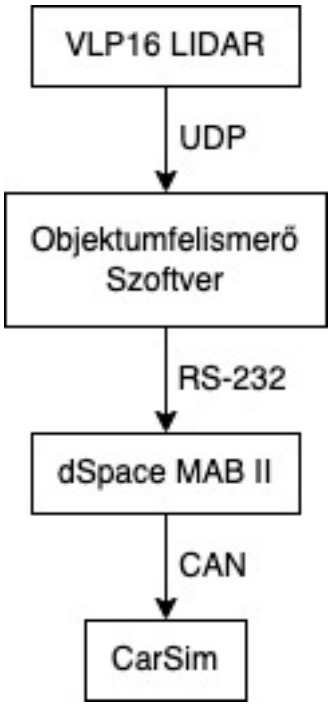
5.3. Támadás célja

A támadás célja az előző pontban kifejtett sérülékenységek alátámasztása: az elvárttól eltérő működés előidézése a rendszerben, DoS támadás kivitelezésével. A támadás célpontja egy LIDAR szenzor, és az általa gyűjtött pontfelhőt feldolgozó objektimfelismerő szoftvert összekötő hálózat, melyet egy saját fejlesztésű malware (5.5 fejezet) - a szenzor által küldött csomagokkal megegyező, de hamis adatokat tartalmazó - kártékony packetekkel elárasztva tesz megbízhatatlanná. Az említett titkosítás hiánya, a komponenseket összekötő IP alapú hálózatok ellen bevethető széles körben elterjet támadások, illetve az UDP alapú kommunikáció (a TCP handshake kikerülésére sincs szükség) ideális célpontá teszi azt.

5.4. Létrehozott teszt környezet

A feltárt sérülékenységek, és a megfogalmazott célok tudatában nekiláttunk a teszt környezet tervezésének és felépítésének, amiben a platform létrehozásában közreműködő SZTAKI-s kollégák segítettek bennünket.

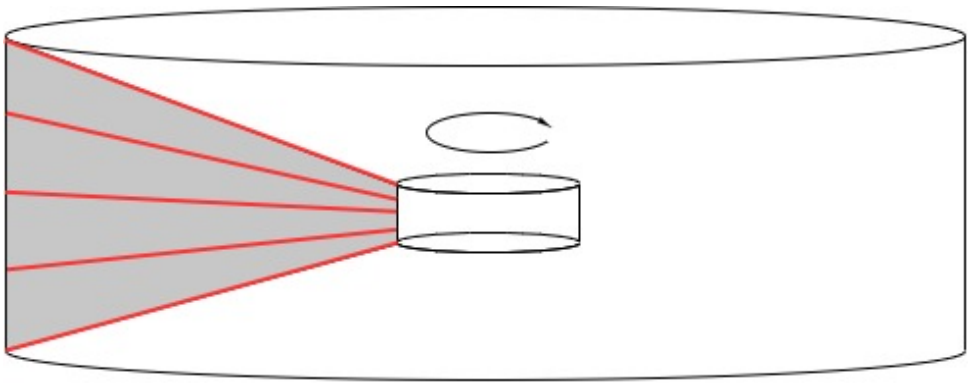
A sérülékenységi teszteket egy hardware-in-the-loop környezetben hajtjuk végre, melynek alapját a fentebb említett Velodyne VLP16 típusú LIDAR szenzor, és objektumdetektáló szoftver képzi, melyet egyben a tesztek validálására is felhasználunk. Habár a jelen tesztben nem töltenek be jelentős szerepet, a környezet részét képzi még egy dSapce MAB II vezérlő egység, ami soros porton fogadja a jármű környezetében fellelt objektumokat, amiket felhasználva - egyéb szenzoradatokkal kombinálva - beavatkozó jeleket generál és küld CAN hálózaton, amiket az alacsonyszintű irányítást végző NI cRIO dolgoz fel. A teszt környezetünkben azonban ezt felváltja egy járműszimulációs szoftver, a CarSim. Mivel a CarSim-et futtató számítógép nem képes CAN hálózatokoz csatlakozni, szükséges egy CAN-USB átalakító (34). Ezek a kiegészítő elemek további tesztek elvégzését segítik elő a jövőben.



18. ábra. Teszt környezet felépítése

5.4.1. LIDAR működése

Az általunk használt, Velodyne Puck 16 LIDAR szenzor környezetét henger alakban képes letapogatni. Ehhez tizenhat darab infravörös laser csatornát (laser modul és detektor pár) használ, melyek a vízszintes síkkal különböző szögeket zárnak be, és függőleges vonalban (annak 16 pontján) képes távolságot mérni. A laser oszlopot egy motor forgatja körbe, így képes a környezetét minden irányba mérni.



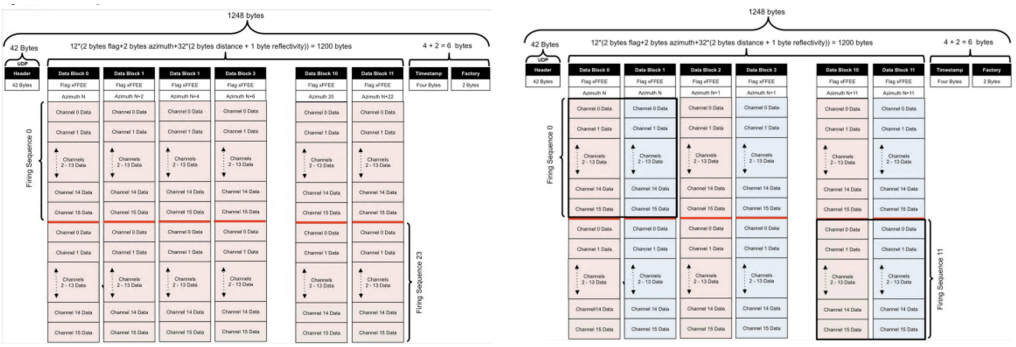
19. ábra. Laser szögek, lidar lefedettsége

A szenzor a time-of-flight elv alapján méri a távolságokat: a lasernyaláb kibocsájtása, és a visszaverődő fény detektálása közötti idő, illetve a fény sebességének tudatában számítható a nyaláb által megtett út. Mérés folyamán, ahogy a laser nyaláb távolodik a szenzortól, egyben annak keresztmetszete is nő. Távoli objektumok esetén ez azt is eredményezheti, hogy a nyaláb több tárgyról is visszaverődik

(37), ennek megfelelően a VLP16-os LIDAR többféle képpen is felkonfigurálható, annak megfelelően, hogy melyik visszaverődés értelmezendő pontként: legutolsó, legerősebb jel, vagy páros visszatérés esetén mindkét jelet megkapjuk.

A VLP16 IP alapú hálózaton terjeszti a mért adatokat, UDP csomagok formájában. Két féle csomag érkezhet a LIDAR felől: pozícióra vonatkozó (GPS), illetve 3D pontfelhő adatok. Számunkra csak a pontfelhő a lényeges, így csak ezt fogjuk részleteiben tárgyalni.

A szenzor a mért pontokat gömbi koordináta-rendszerben adjna meg sugár (R), azimut (α), és emelkedés (ω) koordinátákkal. Egy pont, egy laser csatorna egyszeri mérését jelzi, és három bájt-nak felel meg. Ebből két bájta távolságot (sugár), a harmadik pedig a mért felület kalibrált visszaverőképességét jelöli. Mérési sorozatként definiáljuk azt, amikor mind a 16 csatorna végez egy mérést. A pontok mérési sorozaton belüli elhelyezkedése határozza meg a koordináta emelkedését (ω), például: minden sorozat első pontjának az emelkedése -15°. Ezeket az állandó emelkedéseket a gyártó táblázatban definiálja. Egy adatblokk két mérési sorozatból áll, továbbá kiegészül 2 bájtnyi flaggel (0xFFEE), ami a blokkok kezdetét jelzi, illetve 2 bájt-azimut értékkel, ami előjel nélküli egész számként értelmezendő, és értéke század fokot reprezentál, tehát a 27245-ös érték 272.45°-nak felel meg. Minden adatcsomag 1248 bájthosszú, amiből az első 42 bájtprotokoll header, ezt követi 12 (12*100 bájt) adatblokk, 4 bájt hosszú timestamp, és két gyári bájt, ami segítségével detektálható a LIDAR típusa, vagyis az adatcsomagok felépítése. A timestamp 32-bites előjel nélküli egész szám micromásodpercben reprezentálja az első blokk, első mérési sorozatának első pontjának időpillanatát, az utolsó egész óra óta eltelt időként. A kiválasztott visszatérés konfigurációt az adatcsomagok felépítése is tükrözi (20). Páros visszatérés esetén az adatblokkok páronként értelmezendők, a páratlan számú blokkok tartalmazzák a legerősebb, míg a párosak a legutolsó észlelt jelet. Amennyiben csak egy jelet észlelt a szenzor, a blokk párok tartalma megegyezik.

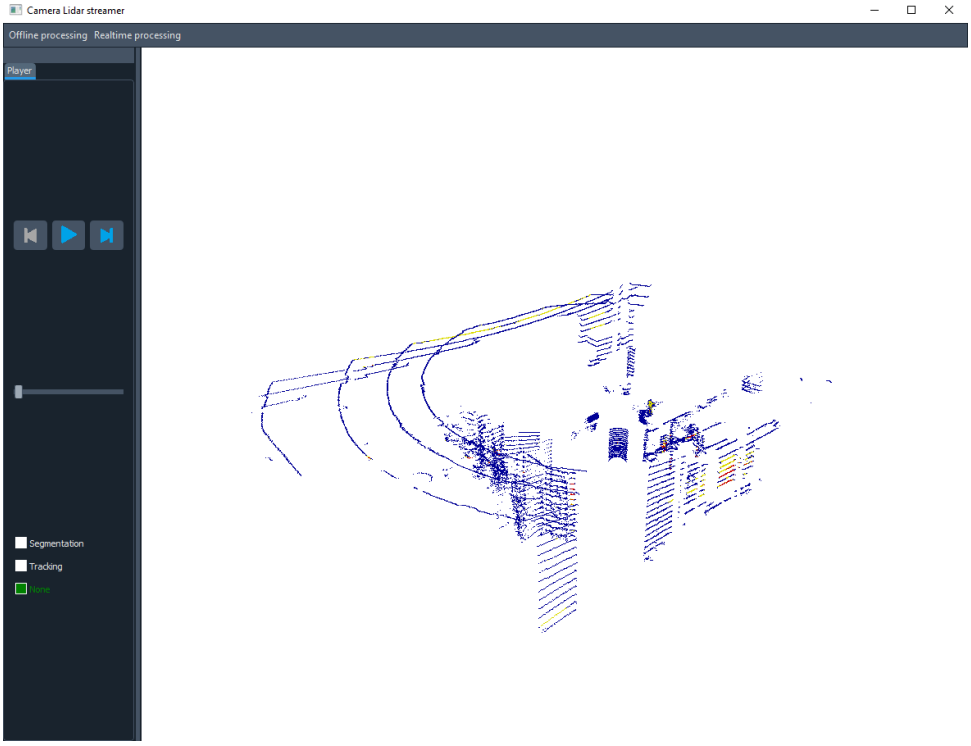


20. ábra. Adatcsomag struktúra egyes és páros visszatérés esetén [6]

A támadást a siker felé mozdítja az a tény is, hogy a fentebb tárgyalt specifikációk bárki számára ingyen elérhetők a gyártó weboldalán.

5.4.2. Objektumfelismerő szoftver

Az objektumfelismerő szoftvert a SZTAKI fejlesztette az autonóm platformhoz, és a LIDAR felől beérkező pontfelhő adatból fordít detektált objektumok relatív pozíciójára, illetve azok méretére, melyeket soros porton küld további használatra. Az felismert tárgyak adatait 26 bájthosszú csomagokban küldi, melyek szerkezetét a 5.6 pontban tágyalunk részletesebben. A szoftverhez grafikus felhasználói felület is társul, ahol a ki- és bemenetek konfigurációja, valamint a beérkező pontfelhő vizualizációja is megtalálható.



21. ábra. Objektumfelismerő szoftver grafikus felülete

A szoftver forráskódja számunkra ismeretlen, így fekete dobozként tekintve rá, első körben csak befolyásolni szeretnénk azt, habár a jövőben érdekes teszteset lehet a beérkező csomagokat feldolgozó modul, vagy a grafikus felület konfigurációs mezőinek kijátszása, így kényszerítve a programot futtató ECU-t saját kód futtatására (remote code execution).

5.5. Támadás kivitelezése - Malidar szoftver

A Malidar (Malicious LIDAR) a projekt keretein belül fejlesztett szoftver, aminek egyetlen célja a LIDAR és az objektumfelismerő szoftver közötti hálózat zavarása. A szoftver két fő osztályból épül fel:

A NetworkInjector osztály felelős a generált csomagok hálózatba juttatásáért (27). Dependency injection design pattern-t alkalmazva az osztály példányosításakor szükséges megadni a használandó packet generáló osztály példányát. A kom-

ponensek különválasztása lehetővé teszi többféle csomaggenerálási stratégia alkalmazását, például: DoS, illetve flooding funkció, ahol random adatokat tartalmazó csomagokkal árasztjuk el a hálózatot, ezzel blokkolva a megbízható működést, vagy "okos" beavatkozás is lehetséges, miszerint virtuális falat hazudunk az autó elé, így megállásra, vagy kanyarodásra kényszerítve azt.

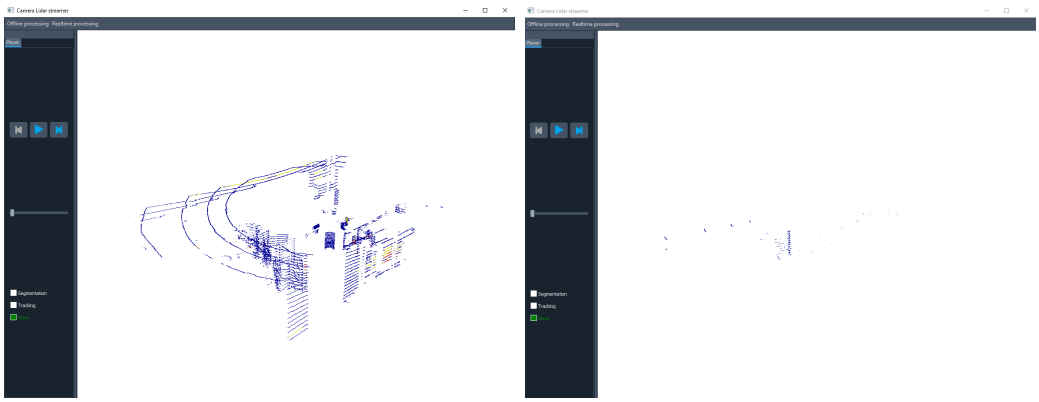
A PacketGenerator ősosztály egy interface-t definiál, amit minden csomag generáló osztálynak implementálnia kell, annak érdekében, hogy a NetworkInjector használni tudja azt (28). A leszármazott osztályoknak csak a _get_data függvényt kell implementálni, aminek az 5.4.1 pontban tárgyalt struktúrájú bájtstringet kell visszaadnia, amit aztán a get_packet függvény visszatérés előtt megfelelő hálózati protokoll headerekkel vesz körbe. A 29-es ábrán látható egy konkrét hálózat flooder implementáció, ami random adatokkal tölti fel a generált hálózati csomagokat.

Kiemelendő, hogy a kommunikáció természetéből fakadóan, a LIDAR adatokhoz a Malidar csak hozzáadni képes, kijelölt valós objektumokat nem tud eltűntetni, ez az objektumfelismerő szoftver kimenetének támadásával kivitelezhető.

A Malidar hálózati funkcióját, illetve részben a packetek generálását egy külső python modul, a Scapy látja el. A Scapy hálózati csomagok lehallgatására, küldésére, gyártására, és elemzésére fejlesztett szoftver. A fejlesztők szerint [7] a kíberbiztonságban használt szoftverek (nmap, tcpdump, stb. . .) nagy részét képes kiváltani. A Scapy kifejelezzen automotív környezetekben használt hálózatok - mint például CAN, LIN, vagy FlexRay - kezelésére is alkalmas, a jövőben a platform további sérülékenységeinek feltárásában is hasznos eszköz lesz.

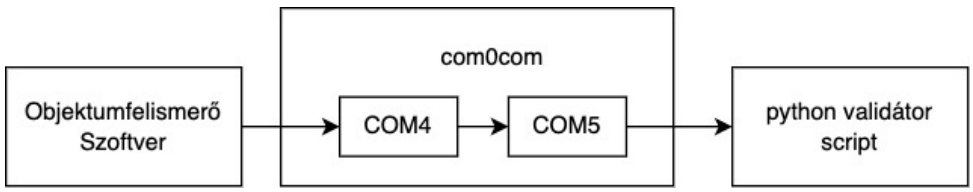
5.6. Támadás eredménye

Sikeresnek tekintjük a támadást, amennyiben az objektumfelismerő szoftvert, a tárgyalt sérülékenységeket kihasználva sikeresen befolyásoljuk. A kívánt hatást egyrészt a szoftver grafikus felületén látható pontfelhő vizualizációval validálhatjuk. A támadáshoz a 29-as ábrán látható flooder osztályt használtuk. A támadáshoz elindítottuk, majd felkonfiguráltuk az objektumfelismerő szoftvert, hogy melyik hálózati interfac-en várja a LIDAR adatokat, végül elindítottuk a Malidar szoftvert is. A 22. ábrán megfigyelhető, hogy támadás alatt a szoftver a valós adatok nagy részét eldobja, normál működéshez képest a kirajzolt pontfelhő jelentősen eltér, tehát a vizuális validáció sikeresnek tekinthető.



22. ábra. Objektumfelismerő szoftver normál üzemmódban (bal), és támadás alatt (jobb)

Azonban a szoftver pontos működésének ismeretének hiányában, ez az eredmény nem tekinthető száz százalékig pontosnak, mivel nem lehetünk biztosak benne, hogy a vizualizáció összhangban van a soros portra írt adatokkal, tehát szükség van egy másodlagos validációra, ami a soros portra írt adatokat figyeli. Ehhez használhatnánk külön számítógépet, vagy mikrokontrollert (pl.: Arduino), azonban a legegyszerűbb a szoftverrel azonos gépen a használt COM portot vizsgálni, viszont egy COM portot egyszerre csak egy process képes igénybe venni, tehát nem lennénk képesek egyszerre írni, és másik szoftverrel olvasni azt. Ennek kiküszöbölésére a com0com nevű szoftvert alkalmaztuk, ami képes virtuális COM port párokat létrehozni. Az objektumfelismerő szoftver, és a validációs script is rá tud csatlakozni egy-egy COM portra, amik között a com0com képzi az átmenetet.



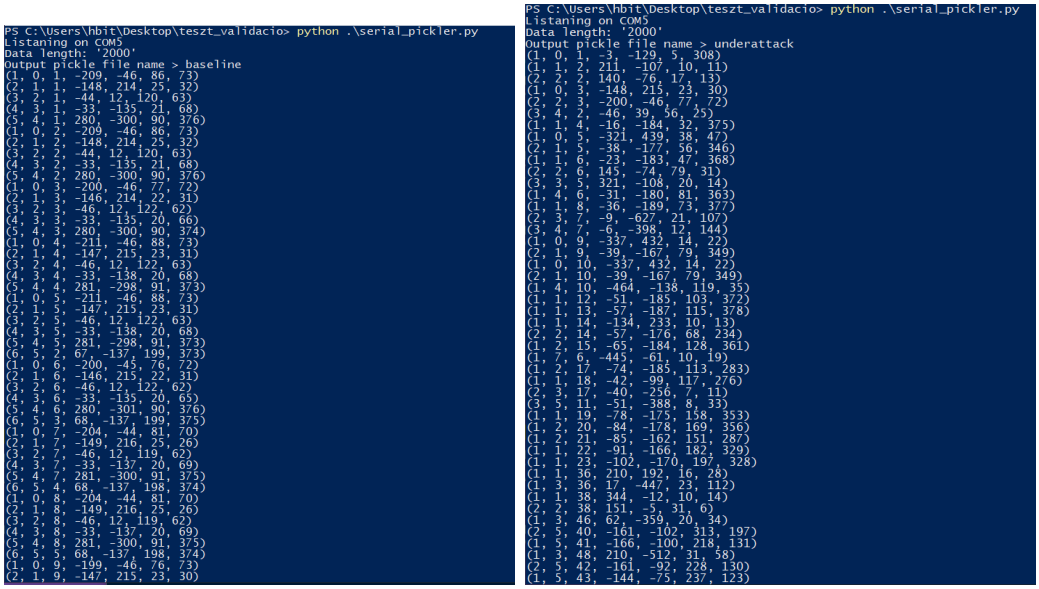
23. ábra. Kimenet validációs pipeline

A validátor egy egyszerű python script (30), ami soros portról képes adatokat olvasni a PySerial csomag segítségével. Egyszerre kétezer bájtnyi adatot olvas be, amit egy nagy tömbben tárol el. Ez azonban emberek számára nehezen olvasható, ezért némi feldolgozásra van szükség. Az objektumfelismerő szoftver jól elkülöníthető, 26 bájthosszú csomagokban, egyesével küldi a felismert tárgyak adatait, tehát az első lépés ezek kinyerése az egy nagy tömbből. A határokat két-két bájtjelzi a csomagok elején (0xFF) és végén (0xF0), a közöttük található adat első két bájtja az objektum identifikálását szolgálja. A maradék 20 bájt - amit a struct modul alakít 4 bájtos egész számokká - az objektum relatív pozícióját és méreteit jelzi, illetve egy számláló mutatja, hogy hány frame-en keresztül sikerült követni. A feldolgozott csomagokat megfigyelhetjük a 24. ábrán, ahol egy sorban egy csomag látható. Ezeket a script későbbi feldolgozáshoz egy file-ba menti el a pickle modul segítségével,

ami lehetővé teszi python struktúrák mentését, scriptek közötti átvitelét.

A validálás három lépésben történik: az első mérést normál működés alatt végezzül el, amit viszonyítási alapként használunk. Ezt követi egy második mérés, ez már a támadás alatt álló rendszer kimenetét rögzíti, végső sorban a két mérés összehasonlítását végezzük el, ami alapján igazoljuk (vagy megcáfoljuk) a támadás sikerességét.

A mérések menetét az alábbiak szerint hajtottuk végre: elindítottuk, majd felkonfiguráltuk az objektumfelismető szoftvert. A 21. ábra bal oldalán megfigyelhető egy play gomb, ami az üzenetek feldolgozását indítja/állítja meg, továbbá három checkbox, amikkel kiválaszható az objektumok detektálásának módja. A szoftver csak akkor írja ki az adatokat a soros portra, ha a középső "tracking" opció van kijelölve. A konfigurációs felületeken (33) kiválasztottuk a megfelelő COM portot, illetve az szenzoradatok forrását, kiválasztottuk a legalsó checkboxot (ez nem hajt végre objektumfelismerést, csak a pontfelhőt mutatja), valamint elindítottuk a beérkező adatok feldolgozását. Ezekkel a lépésekkel az objektumdetektáló szoftver kész, elindíthatjuk a validátor scriptet is. Amennyiben mindkét szoftver fut, az objektumfelismerő szoftver felületén elindíthatjuk a detekráló algoritmust (checkbox), ami határása a validátor kimenetén megjelennek a beolvasott adatok. Az alábbi procedúra a baseline mérést írta le, azonban a támadás során is hasonló az eljárás, az egyetlen különbség, hogy az objektumfelismerés aktiválását a támadó script indítása előzi meg.



24. ábra. Soros portról kiolvasott adatok normál működés (bal), és támadás alatt (jobb)

Első ránézésre a két ábra között nem lelhető fel lényegesebb különbség, azonban részletesebb vizsgálat után bebizonyosodik, hogy a támadás sikeres volt. Számunkra az utolsó 4 oszlopból leolvasható pozíció és méret adatok, illetve a második oszlopban található számok lényegesek, ezek az egyes felismert objektumok egyedi azonosítói.

A bal oldali képen megfigyelhetjük, hogy az azonos azonosítóval ellátott sorok pozíció és méret adatai szinte megegyeznek. Az autonóm platformon mozgás közben nem ez az elvárt működés, azonban a teszt környezet statikus, a szenzorhoz képest a környezete nem mozog, tehát az értékek a valóságot tükrözik. Ezzel ellentétben a jobb oldali kép sokkal kaotikusabb, azonos tárgyakhoz jelentősen eltérő tulajdonságokat rendel, a valós rendszerben továbbhaladva ezekez az adatok definiálatlan magatartáshoz vezetnének, az autó failsafe üzemmódba, hiba esetén előírt állapotba kerülne, az út szélére húzódna.

6. EREDMÉNYEK ÉRTÉKELÉSE

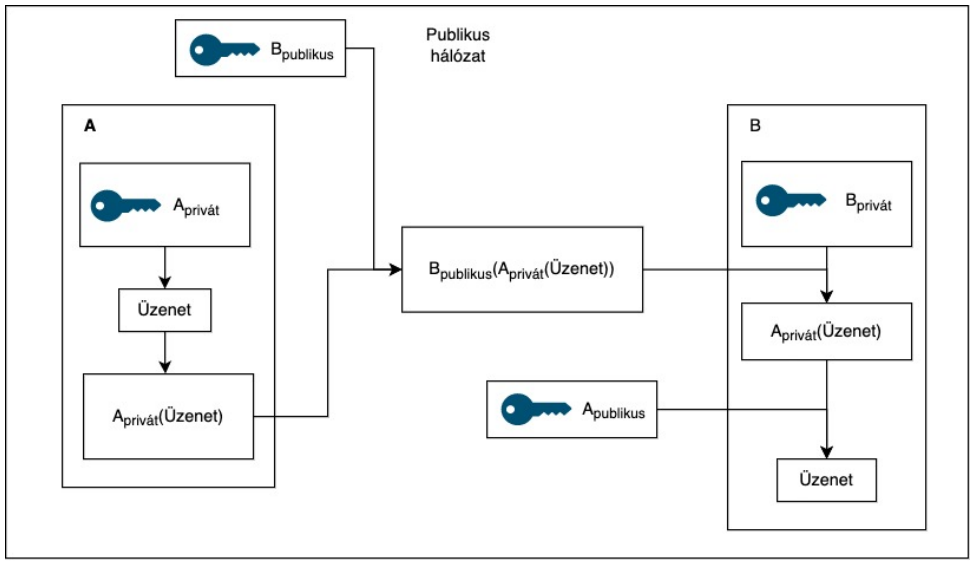
6.1. Javaslatok a sérülékenységek javítására

6.1.1. Titkosítás és autentikáció

Napjainkban a kíberbiztonság a beépített elektronikai eszközök növekvő komplexitás miatt egyre nagyobb jelentőséget kap a járműiparban. A bemutatott sérülékenységi teszt rámutatott arra, hogy amennyiben a gyártók nem szentelnek kellő figyelmet a biztonságnak, a rendszereik viszonylag egyszerű támadásoknak is ki vannak téve. A vizsgált autonóm platform nem gyártásra, illetve piacra szánt, hanem egy kutatást elősegítő rendszer, így a biztonsági szempontok nem kerültek szem elé annak tervezésekor. Habár önvezető funkciók kutatására, fejlesztésére lett létrehozva, a platform biztonságbeli hiányosságai kiváló környezetet nyújt autóipari biztonság kutatására és fejlesztésére.

Jelenlegi felépítésben a legnagyobb biztonsági rést a titkosítás (confidentiality) jelenti, melynek pótlása nagy mértékben megnehezítené a támadók dolgát. Titkosító algoritmusok egyrészt az adatok olvasás elleni védelmét, másrészt a küldő fél azonosítását is biztosítaná. Sharaf és társa [24] a jármű környezetbe felhasználható autentikációval egybekötött cypher algoritmusokat vizsgálták, többek között gyorsaság, energiafogyasztás, és implementálhatóság szempontok alapján.

Wolf és társai [28] szimmetrikus és aszimmetrikus kulcsú titkosítással oldaná meg a problémát. Mindkét módszer kulcsok segítségével alakítja át a titkosítandó szöveget. A szimmetrikus kulcsú rejtjelezésben a küldő, és a fogadó fél is ugyanazt a kulcsot használja az üzenet kódolásához és megfejtéséhez, aminek előnye, hogy az eljárás gyorsan elvégezhető, és nem erőforrás igényesek, ezért ezt alkalmaznák a kommunikáció titkosítására. A legismertebb ilyen rejtjelezések az AES és a DES. A probléma a kulcsok disztribúciójakor merül fel, azaz: hogyan egyeztek meg a felek a kulcsról? Ennek áthidalására a nyílt kulcsú, másnéven aszimmetrikus kulcsú titkosítást alkalmaznák, melynek lényege, hogy a szimmetrikus kulccsal ellentétben minden fél saját kulcspárral rendelkezik, melyek közül egyet titokban tartanak, a másik viszont szabadon terjeszthető bárki számára. A kulcsok különleges tulajdonsága, hogy matematikailag összefüggenek, egyiket sem lehet deriválni a másikból, viszont az egyikkel titkosított üzenet csak a másik kulccsal oldható fel. Tételezzük fel, hogy ECU A üzenetet szeretne küldeni ECU B-nek. Ha A titkosítja az üzenetet B publikus kulcsával, akkor azt csakis B tudja értelmezni, azonban ha előbb A a saját privát kulcsával titkosít, és csak utána B publikus kulcsával, B-nek a saját privát kulcsán kívül A publikus kulcsát is fell kell használnia, tehát biztos lehet benne, hogy valóban A-tól érkezett az üzenet. A legismertebb nyilvános kulcsú algoritmus a Diffie-Hellmann.



25. ábra. Aszimmetrikus rejtjelezés

6.1.2. Behatolásdetektálási- és védelmi rendszerek

A titkosító algoritmusok hátulütője, hogy a kommunikáló felek mindegyikének implementálnia kell azt. Amennyiben a rendszer valamennyi elemét egy csoport fejleszti ez nem jelent nagy problémát, azonban - mint az autonóm platform esetében - ha harmadik féltől származó hardware-t, illetve szoftvert alkalmaznak, amikben ez a funkció nem elérhető, más módszert kell alkalmazni a rendszer védelmére. Az ilyesfajta problémákra hálózat alapú behatolásdetektáló (IDS), vagy behatolásvédelmi (IPS) rendszerek nyújthatják a megoldást. A két technológia működésében kicsi az eltérés, mindkettő illetéktelen hálózati forgalom észlelésére alkalmas, azonban amíg az IDS csak észeleli azt, és jelentést tesz (például a rendszergazdának), addig az IPS aktívan igyekszik blokkolni a kártékony forgalmat.

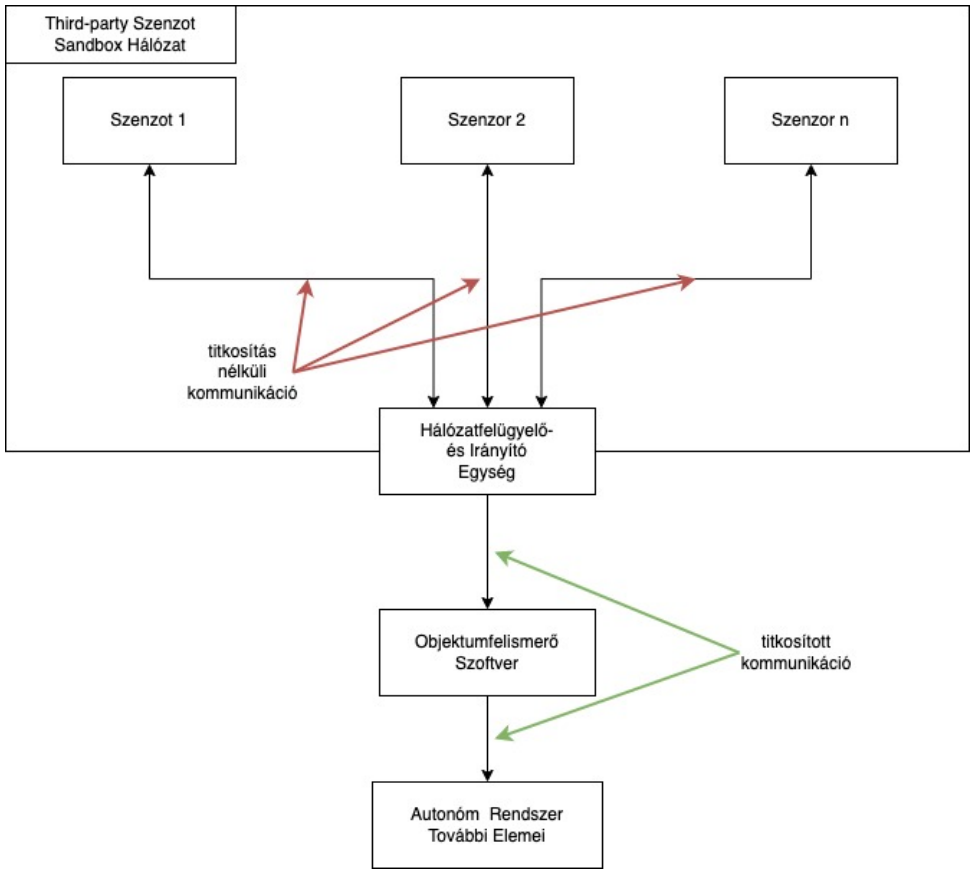
A behatolások detektálásának két fő fajtáját különböztetjük meg: a régóta alkalmazott szignatúra alapú detektálás, ami előre meghatározott minták felismerésére, és ismert sérükénységeket kihasználó támadások észlelésére képes. Előfodulhat az eset, hogy a rendszer auditálásakor egy támadási vektort nem észleltünk, és így az IDS-t sem tudtuk felkészíteni annak észlelésére. Ezekre az esetekre fejlesztették ki az anomália alapú behatolásészlelő rendszereket. Ezek, habár sokkal komplexebb rendszerek, képesek eddig fel nem tárt sérülékenységek észlelésére is, gyakran valamiféle gépi tanulás módszert alkalmazva.

6.1.3. Javasolt javítások

Az első javaslat a 6.1.1 pontban tárgyalt titkosítási eljárás integrálása a rendszerbe. Ennek előkészületeként érdemes lehet egy rövid tanulmány elvégzése, mely többek között pontosan felméri az egyes komponensek között áthaladó adatmennyiséget, valamint az egyes node-okban rendelkezésre álló számítási kapacitást, mely alapján kiválaszható a célra megfelelő titkosítási algoritmus. Ez azonban csak az autonóm

platform azon elemei között lehetséges, melyeket teljes mértékben a rendszer fejlesztői kezelnek, képesek az ECU-kon futó szoftver módosítására.

Harmadik féltől származó hardware esetén, ahol firmware módosítás nem lehetséges (például a vizsgált VLP 16-os LIDAR szenzor), azok elszigetelt, homokozó hálózatba költöztetését javasoljuk. A hálózatban egy központi hálózatfelügyelő egység látná el a router/switch feladatát, ezen futna a hálózatot felügyelő IDS/IPS rendszer, valamint implementálja a platform többi részével történő kommunikációhoz szükséges titkosító algoritmusokat, és egyfajta tűzfalat képezne közöttük. Amennyiben a központi egység kártékony tevékenységet észlel a hálózaton, meggátolja az adatok továbbítását az objektumfelismerő szoftver felé, valamint jelzi a probléma felmerültét.



26. ábra. Javasolt hálózati felépítés

A javasolt felépítésben a központi egység kulcsfontosságú elemét képzi a rendszernek, így annak a védelme különösen nagy prioritást élvez, hardveres manipuláció, szoftveres támadás rengeteg formája ellen fel kell készíteni azt.

6.2. Jövőben elvégezhető további tesztek

Említettük, hogy habár a felépített teszt környezet olyan komponenseket is tartalmaz, amiket a szakdolgozatban tárgyalt sérülékenységi tesztek nem érintettek, a jövőben viszont kiváló alapot képeznek további vizsgálatok elvégzésére.

A Malidar szoftver moduláris architektúrája lehetővé teszi újabb támadási stratégiák, például spoofing kidolgozását, ahol nem random adatokkal, hanem a rendszer számára valósnak tűnő virtuális akadályok tévesztjük meg a rendszert. A hálózatok támadása terén, a dSpace ECU-t és a szimulációs szoftvert összekötő másodlagos CAN hálózat is érdekes lehet. A titkosítás hiánya itt is lehetővé teszi a DoS és spoofing támadásokat, továbbá a szakirodalmakban kifejtett CAN sérülékenységek implementációja is érdekes kihívásokat rejthet.

A tárgyalt LIDAR biztonsági problémákból arra lehet következtetni, hogy a szenzor egyéb sérülékenységeket is rejteget. A konfigurációs weboldalt szolgáltató hardware megismerése, esetleges egyéb nyitott kommunikációs portok feltárása nmap szoftverrel, a webes felület input mezőinek a tesztelése az OWASP top 10 webapplikációs sérülékenységek ellenőrzésével gyakran alkalmazott tesztelési eljárások, amik érdekes eredményekhez vezethetnek. Ugyanúgy az objektumfelismerő szoftver konfigurációs felületének tesztelése buffer overflow-val, vagy a LIDAR packeteket feldolgozó modul kijátszása, a megszokott struktúrától eltérő kártékony csomagokkal akár RCE-hez is vezethet (Remote Code Execution).

Habár a projekt keretein belül nem tettünk ilyesmire vállalást, szakmai fejlődés szempontjából kiváló feladat lenne a javaslatok implementálása, és a tesztek újbóli elvégzése, hogy megbizonyosodjuk az általunk felvetett orvoslatok validságáról. Ha a titkosítás részét nem is lennénk képesek kivitelezni, hisz az ECU-kon futó szoftverek forráskódjai nem állnak rendelkezésünkre, a szenzorokat körbefonó homokozó környezet megvalósítható lenne számunkra. Számos nyílt forráskódú behatolásdetektáló szoftver létezik, és hálózati osztályként a hardver oldali megközelítés sem állna távol tőlünk. Habár a felvetett izolációs környezet kidolgozottság hiányában tartogathat meglepetéseket a biztonság terén, annak sikeres implementációja egy piacképes megoldáshoz vezethet, amelyet nem csak az autóiparban, de akár gyártósori, vagy egyéb automatizált környezetben is alkalmazható.

A felépített teszt környezet kiválóan teljesítette a feladatát, a jövőben viszont fontos lenne visszajelzés, és szakmai fejlődése szempontjából, hogy a tesztjeinket ne csak szimulált környezetben, de a valós autonóm járművön is el tudjuk végezni. Amennyiben hozzáférést kapnánk az autonóm járműhöz, azt komplexebb tesztelésnek is alá tudnánk vetni.

7. ÖSSZEFOGLALÁS

A szakdolgozat egy SZTAKI-s projekt keretein belül jött létre, aminek a célja egy önvezető funckiók fejlesztésére létrehozott autonóm platform tesztelése biztonsági szempontok alapján. Először bemutatásra kerültek a napjainkban gyakran alkalmazott járműipari alkatrészek: a vezérlő elektronikák, illetve az őket összekötő hálózati típusok, miután korábbi cikkek, és könyvek segítségével ezen komponensek támadási vektoraikat kielemeztük, és megismertük az ellenük bevethető támadási metódusokat is. Bemutatásra kerültek a platform egyes elemei is, melyeket a megismert sérülékenységek alapján vizsgáltunk. Létrehoztunk egy hardware-in-the-loop kötnyezetet a sérülékenységi tesztek végrehajtására, melyek részben a platformon megtalálható elemekkel megegyező komponensekből épültek fel, továbbá a támadások végrehajtására alkalmas szoftvert is lefejlesztettük. Sikeres támadással igazoltuk a rendszer sérülékenységét, valamint javaslatokat is tettünk ezek javítására, és a jövőben elvégezhető további mérésekre.

Habár a szakdolgozat az autóipari kíberbiztonság témáját korán sem merítette ki, remélem hozzájárul annak fejlődéséhez, és jó kiindulási pontot biztosít a téma iránt érdeklődők számára.

Hivatkozások

[1] URL: https://primatec.tn/expertises/in-vehicle-networks/.

[2] „LIN – Local Interconnect Network”. *Multiplexed Networks for Embedded Systems*. John Wiley & Sons, Ltd, 2007. 7. fej., 285–311. old. ISBN: 9780470511770. DOI: https://doi.org/10.1002/9780470511770.ch7. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470511770.ch7. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470511770.ch7.

[3] „MOST - The Automotive Multimedia Network”. Szerk. Prof. Dr. Ing. Andreas Grzemba. Franzis Verlag GmbH, 2011.

[4] URL: https://www.dsf.my/2019/02/obd2-guide/.

[5] URL: https://en.wikipedia.org/wiki/On-board_diagnostics#/media/File:MaxScan_OE509_collage.jpg.

[6] URL: https://velodynelidar.com/products/puck/.

[7] URL: https://scapy.net.

[8] C. Bisdikian. „An overview of the Bluetooth wireless technology”. *IEEE Communications Magazine* 39.12 (2001), 86–94. old. DOI: 10.1109/35.968817.

[9] Marshall Wilensky Candace Leiden. „TCP/IP for Dummies, 6th Edition”. Wiley Publishing, Inc., 2009.

[10] Robert N. Charette. „This Car Runs on Code”. (2009).

[11] Stephen Checkoway, Damon McCoy, Brian Kantor és tsai. „Comprehensive Experimental Analyses of Automotive Attack Surfaces”. *20th USENIX Security Symposium (USENIX Security 11)*. San Francisco, CA: USENIX Association, 2011. aug. URL: https://www.usenix.org/conference/usenix-security-11/comprehensive-experimental-analyses-automotive-attack-surfaces.

[12] Steve Corrigan. *Introduction to the Controller Area Network (CAN)*. SLOA101B. Rev. B. Texas Instruments. 2022.

[13] David Coss. „The CIA strikes back: Redefining confidentiality, integrity and availability in security”. *Journal of Information System Security* 10 (2014. jan.).

[14] Kenneth Hacker, Scott Graham és Stephen Dunlap. „Vehicle Identification and Route Reconstruction via TPMS Data Leakage”. *Critical Infrastructure Protection XIII*. Szerk. Jason Staggs és Sujeet Shenoi. Cham: Springer International Publishing, 2019, 123–136. old. ISBN: 978-3-030-34647-8.

[15] Tobias Hoppe, Stefan Kiltz és Jana Dittmann. „Security Threats to Automotive CAN Networks – Practical Examples and Selected Short-Term Countermeasures”. *Computer Safety, Reliability, and Security*. Szerk. Michael D. Harrison és Mark-Alexander Sujan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, 235–248. old. ISBN: 978-3-540-87698-4.

[16] John D Howard és Thomas A Longstaff. „A common language for computer security incidents”. (1998. okt.). DOI: 10.2172/751004. URL: https://www.osti.gov/biblio/751004.

[17] Karl Koscher, Alexei Czeskis, Franziska Roesner és tsai. „Experimental Security Analysis of a Modern Automobile”. (2010), 447–462. old. DOI: 10.1109/SP.2010.34.

[18] Yunxin (Jeff) Li. „An Overview of the DSRC/WAVE Technology”. *Quality, Reliability, Security and Robustness in Heterogeneous Networks*. Szerk. Xi Zhang és Daji Qiao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 544–558. old. ISBN: 978-3-642-29222-4.

[19] Charlie Miller és Chris Valasek. „A survey of remote automotive attack surfaces”. *black hat USA* 2014 (2014), 94. old.

[20] R. Nusser és R.M. Pelz. „Bluetooth-based wireless connectivity in an automotive environment”. *Vehicular Technology Conference Fall 2000. IEEE VTS Fall VTC2000. 52nd Vehicular Technology Conference (Cat. No.00CH37152)*. 4. köt. 2000, 1935–1942 vol.4. DOI: 10.1109/VETECF.2000.886152.

[21] Jonathan Petit és Steven E. Shladover. „Potential Cyberattacks on Automated Vehicles”. *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), 546–556. old. DOI: 10.1109/TITS.2014.2342271.

[22] Florian Sagstetter, Martin Lukasiewycz, Sebastian Steinhorst és tsai. „Security challenges in automotive hardware/software architecture design”. *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2013, 458–463. old. DOI: 10.7873/DATE.2013.102.

[23] Fatih Sakiz és Sevil Sen. „A survey of attacks and detection mechanisms on intelligent transportation systems: VANETs and IoV”. *Ad Hoc Networks* 61 (2017), 33–50. old. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2017.03.006. URL: https://www.sciencedirect.com/science/article/pii/S1570870517300562.

[24] Sahar Sharaf és Hassan Mostafa. „A study of Authentication Encryption Algorithms (POET, Deoxys, AEZ, MORUS, ACORN, AEGIS, AES-GCM) For Automotive Security”. *2018 30th International Conference on Microelectronics (ICM)*. 2018, 303–306. old. DOI: 10.1109/ICM.2018.8704025.

[25] Robert Shaw és Brendan Jackman. „An introduction to FlexRay as an industrial network”. *2008 IEEE International Symposium on Industrial Electronics*. 2008, 1849–1854. old. DOI: 10.1109/ISIE.2008.4676987.

[26] Rainer Steffen, Richard Bogenberger, Joachim Hillebrand és tsai. „Design and Realization of an IP-based In-car Network Architecture”. 2010. máj. DOI: 10.4108/ICST.ISVCS2008.3543.

[27] Stephen Turner és Suleyman Uludag. „Towards Smart Cities: Interaction and Synergy of the Smart Grid and Intelligent Transportation Systems”. 2015. júl.

[28] Marko Wolf, André Weimerskirch és Christof Paar. „Security in automotive bus systems”. (2004).

A. KÓDOK

A.1. Malidar

```
59 from scapy.all import *
60
61 class NetworkInjector:
62
63     def __init__(self, network_iface, packet_generator):
64         self.network_interface = network_iface
65         self.generator = packet_generator
66
67     def run(self, total_packets=1000, buff_size=10, continuous=False):
68         #buff_size - 10 packets should contain (12(block)*32(point)[packet])*100=3840 points
69         print(f"Starting injection on interface: '{self.network_interface}'")
70
71         while True:
72             for _ in range(0, total_packets, buff_size):
73                 sendp(
74                     list(self.generator.get_packets(buff_size)),
75                     iface=self.network_interface
76                 )
77
78             if not continuous:
79                 print("Continouos mode was not defined, finishing injection")
80                 break
81
82 def main():
83     packet_gen = PacketGenerator()
84     injector = NetworkInjector("eth0", packet_gen)
85     injector.run()
86
87 if __name__ == "__main__":
88     main()
89
```

27. ábra. NetworkInjector osztály

```python
from scapy.all import *

class PacketGenerator:
    """
    Base class for generating malicious packets
    """
    def __init__(self, sender_addr, sender_mac, dest_addr, dest_port):
        self._sender_addr = sender_addr
        self._sender_mac = sender_mac

        self._dest_addr = dest_addr
        self._dest_port = dest_port

    def get_packets(self, num):
        for _ in range(num):
            yield self.get_packet()

    def get_packet(self):
        # Create one packet with scapy
        pkt = Ether(src=self._sender_mac, dst="ff:ff:ff:ff:ff:ff")/ \
            IP(src=self._sender_addr, dst=self._dest_addr)/ \
            UDP(sport=self._dest_port, dport=self._dest_port)/ \
            self._get_data()

        return pkt

    def _get_data(self):
        raise NotImplemented("Subclasses should implement this function")

```

28. ábra. PacketGenerator ősosztály

```python
from packet_generator import PacketGenerator

class FloogingPacketGenerator(PacketGenerator):
    """
    This class generates packets for network flooding applications.
    That means packet data is generated randomly.
    """

    AZIMUT = 0

    def __init__(self):
        super().__init__(
            "10.111.1.2", #source ip
            "60:76:88:10:2f:08", # source mac
            "10.111.1.111", # destination ip
            2368) # destination port

    def _get_data(self):
        """
        Generate random, 1200 bytes long data
        """
        t = datetime.now()
        timestamp = int.to_bytes(t.minute * 60000000 + t.second * 1000000 + t.microsecond, 4, "little")
        factory_bytes = b'7"'
        data = timestamp + factory_bytes

        for _ in range(12):
            data = self._create_block() + data

        return data

    def _create_block(self):

        flag = b"\xff\xee"
        azimuth = int.to_bytes(self._get_azimut(), 2, "little") # angle * 100

        block = flag + azimuth

        while len(block) < 100:
            dist = int.to_bytes(random.randint(0, 1000), 2, "little")
            refl = random.randbytes(1)
            block += dist + refl # one point = distance + reflectivity

        return block

    def _get_azimut(self):
        ret = JammingPacketGenerator.AZIMUT % 36000
        JammingPacketGenerator.AZIMUT += 1
        return ret
```

29. ábra. PacketGenerator flooder implementáció

A.2. Soros kimenet validátor

```python
if __name__ == "__main__":
    read_data = b""
    print("Listaning on COM5")
    with serial.Serial(port="COM5", timeout=5) as port:
        try:
            while True:
                data = port.read(100)
                read_data = read_data + data
                print(" "*20, end="\r") # clear previous line
                print(f"Data length: '{len(read_data)}'", end="\r")

                if len(read_data) == 2000:
                    break
        except KeyboardInterrupt:
            pass
        finally:
            print("") # fix any printing disturbances from line 40 and 41
            output_file_name = input("Output pickle file name > ")
            output_file_name = output_file_name + ".pickle"

            parsed_data = parse_data(read_data)
            for packet in parsed_data:
                print(packet)

            with open(output_file_name, "wb") as f:
                pickle.dump(parsed_data, f)

            print(f"Done! \nPickled output > {output_file_name}")
```

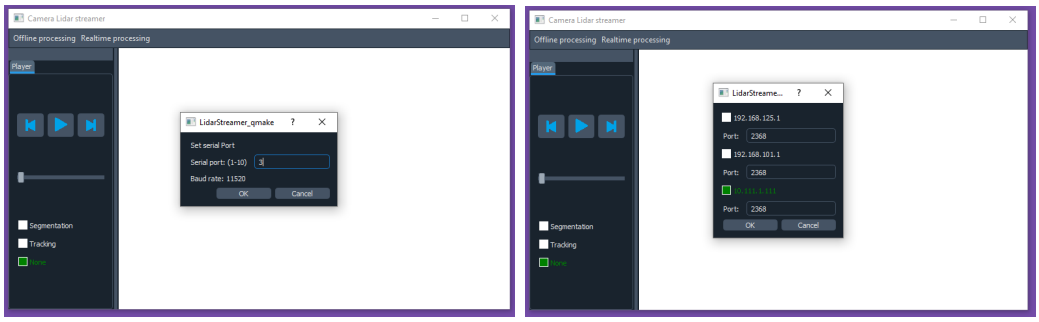
30. ábra. Validációs python script

```python
def parse_data(data):
    # 1 packet = 26 bytes
    # -> 2 bytes header, 2 bytes tail
    # -> 1 byte frame object ID, 1 byte tracking object ID
    # -> 4 byte counter
    # -> 16 byte positional data

    split_data = []
    for i in range(0, len(data), 26):
        split_data.append(data[i:i+26])

    # last packet might not be full length
    if len(split_data[-1]) < 26:
        split_data.pop()

    parsed = []
    for raw in split_data:
        naked = raw[2:24] # removed header and tail

        frame_obj_id = naked[0]
        tracking_obj_id = naked[1]
        counter, dx, dy, rx, ry = struct.unpack(">iiiii", naked[2:22]) # '>' = read as big endian

        parsed.append((frame_obj_id, tracking_obj_id, counter, dx, dy, rx, ry))

    return parsed
```

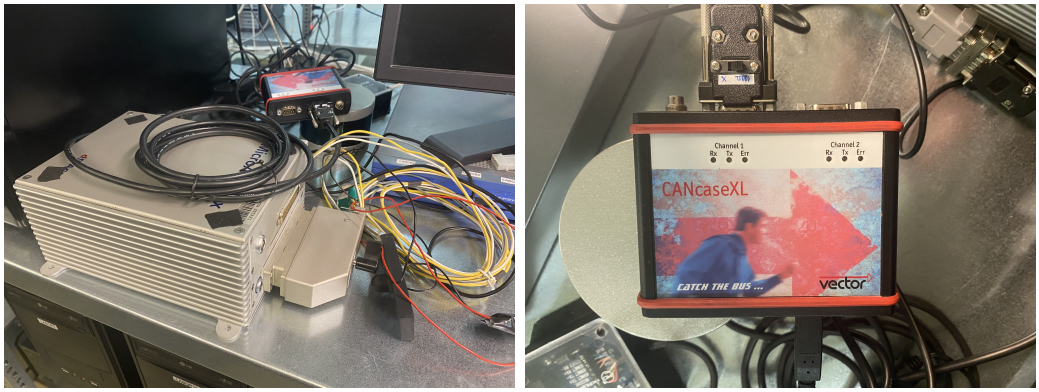
31. ábra. Soros portról olvasott adatfeldolgozó függvény

B. KÉPEK A TESZTKÖRNYEZETRŐL

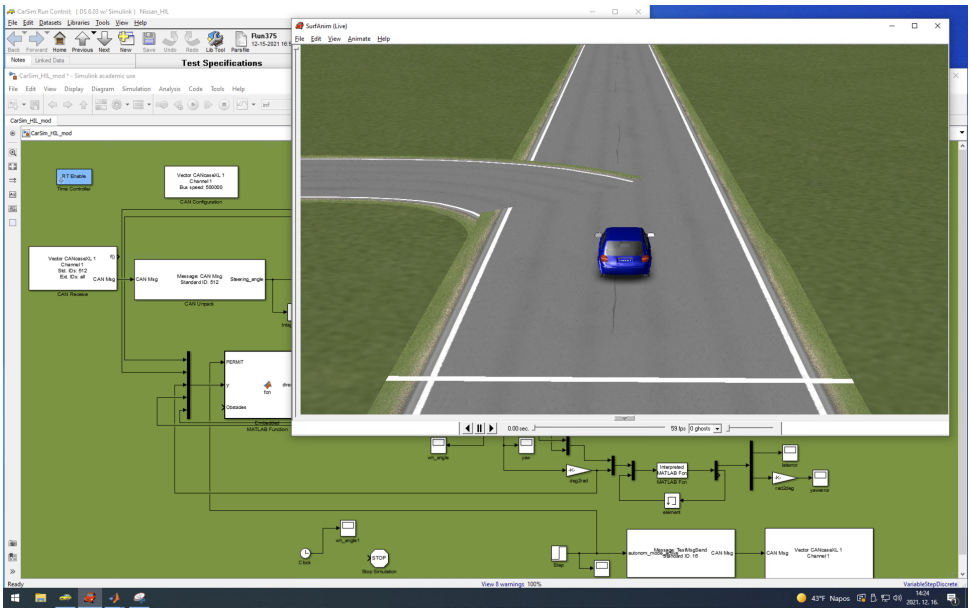
32. ábra. LIDAR szenzor és vezérlője



33. ábra. Az objektumfelismerő szoftver konfigurációs felületei

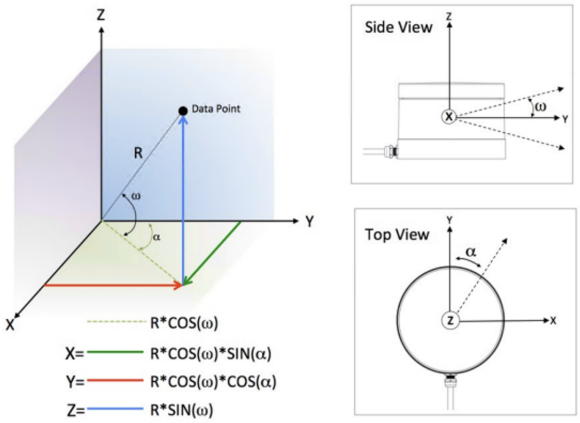


34. ábra. dSpace MAB II, és a hozzá szükséges CAN átalakító

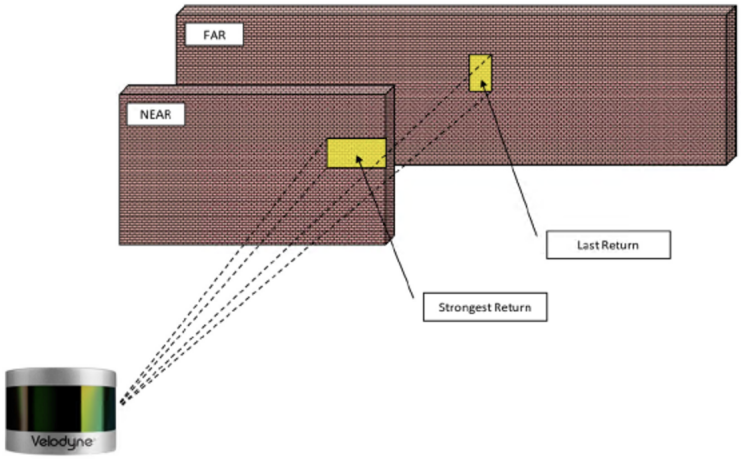


35. ábra. CarSim szimulációs szoftver

C. LIDAR



36. ábra. LIDAR koordináta rendszer [6]



37. ábra. Visszaverődési módok [6]