Proširivi sustav za stvaranje CTF zadataka specifičnih za sustave upravljanja i zabave u automobilima

Grgurić Mileusnić, Lovro

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 656

PROŠIRIVI SUSTAV ZA STVARANJE CTF ZADATAKA SPECIFIČNIH ZA SUSTAVE UPRAVLJANJA I ZABAVE U AUTOMOBILIMA

Lovro Grgurić Mileusnić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 656

PROŠIRIVI SUSTAV ZA STVARANJE CTF ZADATAKA SPECIFIČNIH ZA SUSTAVE UPRAVLJANJA I ZABAVE U AUTOMOBILIMA

Lovro Grgurić Mileusnić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 656

Pristupnik:	**Lovro Grgurić Mileusnić (0036523763)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentor:	izv. prof. dr. sc. Stjepan Groš

Zadatak: **Proširivi sustav za stvaranje CTF zadataka specifičnih za sustave upravljanja i zabave u automobilima**

Opis zadatka:

Kompleksnost upravljačkih sustava, kao i sustava za zabavu u automobilima sve je veća. To omogućava niz pogodnosti, kao primjerice sve veća sigurnost za putnike, štedljivosti automobila, povećane udobnosti vožnje. Međutim, ta kompleksnost kao i stalna povezanost na Internet za sobom povlači i potencijalne izazove, prvenstveno po pitanju kibernetičke sigurnosti. Iz tog razloga sve veća pažnja posvećuje se pitanju kako spriječiti napadače da naštete vozilu ili putnicima. Međutim, kao i u svim drugim područjima istaknut je nedostatak kvalificiranog kadra te svijesti onih koji rade u automobilskoj industriji o tim problemima. U diplomskom radu potrebno je napraviti sustav koji će omogućiti upoznavanje sadašnjih, ali i budućih stručnjaka sigurnosti sa specifičnostima sustava u automobilima. To je potrebno napraviti na način da se definira niz zadataka tipa Capture the Flag koje u sebi uključuju specifičnosti sustava ugrađenih u automobile. Pri definiranju zadataka obratiti pozornost na postojeće napade i ranjivosti u automobilskim sustavima koje su otprije poznate. Dodatno, treba definirati metodu stvaranja novih zadataka kako bi se u budućnosti skup zadataka mogao znatno proširiti da bi se na taj način poboljšala kvaliteta edukacije.

Rok za predaju rada: 28. lipnja 2024.

SADRŽAJ

1. Uvod

Današnji automobili i ostala cestovna vozila značajno se razlikuju od njihovih prvih inačica, iako im glavna namjena ostala ista. Pojavom i širom primjenom elektronike, automobili više nisu samo mehanički strojevi, već sadrže određen broj međusobno povezanih elektroničkih upravljačkih jedinica (engl. *electronic control unit*, ECU) [40]. Elektroničke upravljačke jedinice svojom međusobnom suradnjom omogućavaju sigurno upravljanje vozilom, ali i dodatne funkcionalnosti poput klimatizacije te sustava za informacije i zabavu (engl. *in-vehicle infotainment*). Većina komunikacije između elektroničkih upravljačkih jedinica odvija se putem protokola poput *Controller Area Network* (CAN) protokola i pripadajućih sabirnica. Prvotno namijenjen za primjenu u automobilima, CAN ima svojstva prikladna za komunikaciju u stvarnom vremenu (engl. *realtime*), kao i sposobnosti detektiranja grešaka i propuštanja poruka višeg prioriteta [72]. Međutim, iako ga specifikacija druge inačice CAN protokola, CAN 2.0, opisuje kao protokol s visokom razinom sigurnosti, ni ta niti originalna specifikacija ne razmatraju osiguravanje osnovnih sigurnosnih svojstava povjerljivosti, integriteta i dostupnosti (engl. *confidentiality, integrity, availability*, CIA) [33, 45]. Zbog drastične prirode posljedica koje bi neispravan rad sustava automobila mogao imati na vozača i njegove suputnike te ostale sudionike prometa, pri razvoju sustava automobila uvijek je bila pridodana posebna pažnja njihovoj funkcionalnoj sigurnosti (engl. *safety*) i ispravnosti, ali ne nužno i kibernetičkoj sigurnosti [40]. Zbog tadašnje zatvorenosti komunikacije ovih sustava, za potencijalnog napadača nije postojala površina (engl. attack surface) koju bi mogao iskoristiti bez fizičkog pristupa CAN sabirnici.

Interne mreže današnjih vozila kao i programska podrška njihovih sustava znatno su kompleksnije i povezanije s vanjskim svijetom [37]. Mnogi novi modeli automobila su opremljeni SIM karticama odnosno mogućnošću povezivanja na mobilnu mrežu u svrhu komunikacije sa servisima proizvođača u oblaku (engl. cloud services) i mobilnim aplikacijama koje omogućuju udaljeno upravljanje vozilom. Uz nevedeno, mobilna mreža koristi se i u svrhu preuzimanja ažuriranja (engl. *Over-The-Air updates*,

OTA) za sustave zabave, ali i za kritične elektroničke upravljačke jedinice. *Bluetooth* tehnologija najčešće se koristi za povezivanje mobilnog telefona sa sustavom zabave kako bi se omogućilo pozivanje i reprodukcija glazbe. U svrhu bolje integracije mobilnih telefona u sustav zabave, mnogi proizvođači podržavaju platforme poput *Android Auto* ili *Apple Carplay*, kojima se sustav zabave proširuje mogućnostima preuzimanja i pokretanja raznovrsnih aplikacija iz izvora treće strane. Vozila opremljena mrežnim karticama s mogućnošću stvaranja i korištenja Wi-Fi bežičnih pristupnih točaka, mogu dijeliti pristup mobilnoj mreži koji ostvaruju s prethodno navedenim SIM karticama ili povezivati se na pristupne točke korisnika u svrhu preuzimanja OTA ažuriranja ili aplikacija. Za skoro svaku od navedenih funkcionalnosti istraživači automobilske sigurnosti pokazali su mogućnost njihvog iskorištavanja i ulančavanja otkrivenih ranjivosti u svrhu neautoriziranog pristupa i postizanja kontrole nad sustavima zabave, ali i kritičnim sustavima upravljanja automobila [54, 55, 16, 42, 50, 21]. Nadalje, već u izvješću o stanju automobilske sigurnosti tvrtke *Upstream Security* iz 2019. godine, predviđa se da će sva nova vozila proizvedena u 2025. godini imati navedene funkcionalnosti, ali i da će podržavati komunikaciju među vozilima (engl. *Vehicle-to-Vehicle, V2V*) i komunikaciju vozila s infrastrukturom (engl. *Vehicle-to-Infrastructure, V2I*), dodatno povećavajući dostupnu površinu napada [61].

Nagli rast povezanosti i kompleksnosti sustava vozila stvara potrebu za obrazovanjem određenog broja stručnjaka i inženjera koji će testirati i osiguravati kvalitetniju kibernetičku sigurnost tih sustava, od razine sklopovlja do *backend* servisa te V2V i V2I komunikacije. Osim formalnih sustava obrazovanja, popularan format za edukaciju stručnjaka kibernetičke sigurnosti su i *Capture the Flag* (CTF) zadaci. Međutim, Švábensky et al. ističu u [65] da se u analiziranom uzorku od 12 952 *Capture the Flag* zadataka, zadaci klasificirani u područje sigurnosti sklopovlja čine tek 8,19%, što je 6. najmanji udio od ukupno 8 analiziranih područja znanja. Većina današnjih *Capture the Flag* zadataka spada u razvijenije i poznatije grane kibernetičke sigurnosti poput *web* sigurnosti, sigurnosti operacijskih sustava i aplikacija te sigurnosti oblaka (engl. *cloud security*)[56]. U svrhu edukacije sigurnosnih stručnjaka u području kibernetičke sigurnosti u automobilskoj industriji potrebno je napraviti proširivi sustav koji će služiti kao temelj za stvaranje *Capture the Flag* zadataka koji sadrže specifičnosti sustava u automobilima.

Rad je strukturiran kroz 6 poglavlja. U drugom poglavlju, napravljen je pregled protokola te arhitekture modernog automobila. Kroz ovo poglavlje opisani su protokoli i sustavi modernog automobila, čime se postavlja teoretska podloga za razumijevanje

napada na iste, opisanih u trećem poglavlju. U četvrtom poglavlju razmatraju se postojeći sustavi za obuku stručnjaka sigurnosti u grani sigurnosti automobila i definiran je popis poželjnih svojstava takvog sustava. U petom poglavlju opisana je implementacija proširivog sustava za stvaranje *Capture the Flag* zadataka specifičnih za sustave upravljanja i zabave u automobilima. U zaključku je opisan konačan rezultat, prednosti i nedostatci implementiranog sustava te moguća unaprjeđenja i smjernice za daljnji razvoj.

2. Sustavi i protokoli modernih automobila

Kao temelj za razumijevanje iskoristivih površina i mogućih vektora napada na sustave automobila, kroz ovo poglavlje opisana je električna i elektronička arhitektura modernog automobila (engl. *Electrical/Electronic* architecture). Opisani su i protokoli specifični za automobile kroz više slojeva OSI modela.

2.1. E/E arhitektura modernog automobila

E/E arhitektura obuhvaća elektroničke komponente, električko ožičenje, tehnologije umrežvanja i programsku podršku jednog automobila [52]. Prema [52, 40, 39, 37, 9], sustavi vozila dijele se prema funkciji u domene:

- pogonskog sklopa (engl. *powertrain*)
- šasije (engl. *chassis*)
- kabine (engl. *interior cabin* ili *body*)
- zabave i povezivosti (engl. *infotainment and connectivity*)

Iako se sustavi mogu prema funkciji podijeliti u navedene 4 domene, domene nužno ne određuju topologiju mreže automobila.

Domene pogonskog sklopa, šasije i kabine sadrže ECU-ove koji najviše utječu na fizičko stanje automobila te je njihova komunikacija većinom ograničena na internu mrežu automobila. Domena informacija, zabave i povezivosti sadrži sustave koji pružaju dodatne informacije i sadržaje vozaču i putnicima te vrše komunikaciju i s okolinom automobila, primjerice s *backend* servisima, mobilnim uređajima i aplikacijama. Stoga je domena informacija, zabave i povezivosti obrađena u zasebnom potpoglavlju.

2.1.1. Domene pogonskog sklopa, šasije i kabine

Elektroničke upravljačke jedinice odnosno ECU-ovi, pripadaju domenama pogonskog sklopa, šasije i unutarnje kabine te služe za koordiniranje i upravljanje većinom funkcija automobila. ECU-ovi detektiraju trenutne operativne uvjete pomoću senzora, procesuiraju ih i aktiviraju aktuatore [34]. Sklopovlje ECU-a najčešće se nalazi u zatvorenom zaštitnom kućištu s izloženim priključcima za spajanje na ostatak mreže automobila, senzore i aktuatore. Uobičajeno je to tiskana pločica na kojoj se najčešće nalazi integrirani krug za upravljanje snagom (engl. *power management integrated circuit*, PMIC), primopredajnici protokola kojim komuniciraju, memorije te mikrokontroler ili sustav na čipu (engl. *System on a chip*), SoC) [52]. ECU-ovi ovih domena najčešće zbog zahtjeva za radom u stvarnom vremenu međusobno komuniciraju putem CAN ili FlexRay protokola, gdje specifična topologija mreže ovisi o modelu automobila [34].

Domeni pogonskog sklopa pripadaju ECU-ovi koji upravljaju i utječu na funkciju motora i prijenosa, a to su upravljački modul motora (engl. *engine control module*, ECM) i upravljački modul prijenosa (engl. *transmission control module*, TCM). U literaturi se često koriste i nazivi upravljačka jedinica motora (engl. *engine control unit*, ECU), odnosno upravljačka jedinica prijenosa (engl. *transmission control unit*, TCU) [52, 40]. Često automobili imaju jedan snažniji ECU koji obavlja obje funkcije, funkciju upravljanja motorom i upravljanja prijenosom te se takav ECU naziva upravljačkom jedinicom pogonskog sklopa (engl. *powertrain control unit*, PCU) ili upravljačkim modulom pogonskog sklopa (engl. *powertrain control module*, PCM) [34, 66]. U slučaju vozila na električni pogon, u domeni pogonskog sklopa pripadaju sustav upravljanja baterijom (engl. *battery managment system*, BMS), inverter (engl. *inverter*) i PCU. Uz navedene, u domenu pogonskog sklopa ulaze i druge upravljačke jedinice koje prikupljaju podatke sa senzora i upravljaju aktuatorima relevantnima pogonskom sklopu. S obzirom na to da navedeni ECU-ovi imaju izravan utjecaj na kretanje automobila, njihovo kompromitiranje može rezultirati fizičkim ozljedama vozača i putnika.

U domenu šasije svrstavaju se sustavi i senzori odgovorni za funkcije zaštite putnika koje zahtijevaju reakciju u stvarnom vremenu poput sustava kočenja, ovjesa i zračnih jastuka. Sukladno tome, kao i u slučaju sustava domene pogonskog sklopa, njihovo ispravno funkcioniranje je iznimno bitno za sigurnost vozača i putnika [52]. Jedna od upravljačkih jedinica koje pripadaju u domenu šasije je upravljački modul elektroničkog kočenja (engl. *electronic braking control module*, EBCM). EBCM je specijalizi-

rani modul koji upravlja aktivnim sigurnosnim mjerama poput ABS-a (njem. *Antiblockiersystem*), elektroničkom kontrolom stabilnosti (engl. *electronic stability control*, ESC) te automatskog hitnog kočenja (engl. *automated emergency breaking*, AEB). Uz EBCM, u domenu šasije svrstavaju se i napredni sustav za podršku vozaču pri upravljanju vozilom (engl. *Advanced driver-assistance system*, ADAS), koji je sve češće prisutan u novim vozilima [52]. ADAS je odgovoran za funkcije poput rada tempomata, održavanja razmaka između vozila, automatskog zadržavanja trenutne prometne trake, automatskog parkiranja te autonomne vožnje [34]. Domeni šasije pripada i upravljački modul zračnih jastuka (engl. *airbag control module*, ACM) te sustav servo upravljača (engl. *electronic power steering*, EPS). ADAS i EBCM prikupljaju podatke od niza sustava senzora, primjerice od radara za određivanje udaljenost objekata u svrhu aktivacije funkcije AEB, od ultrazvučnih senzora za potrebe sustava automatskog parkiranja te od sustava lidar i video senzora za potrebe autonomne vožnje.

U domenu kabine spadaju sustavi manje kritičnih funkcija. Uključuju sustave grijanja, ventilacije i klimatizacije (engl. *heating, ventilation and air conditioning*, HVAC), pripadajući upravljački modul klimatizacije (engl. *climate control module*, CCM) te ECU-ove za prikupljanje podataka sa senzora sigurnosnih pojasa i sjedala te ECU-ove za upravljanje ambijentalnim osvjetljenjem, prozorima, brisačima i ostalim dijelovima kabine. Uz navedene, kao sigurnosno bitni sustavi, ističu se sustav za ulaz bez ključa (engl. *keyless entry system*, KES) te ECU za upravljanje otključavanjem vrata vozila i pripadajući prijemnik za daljinsko zaključavanje (engl. *remote control door lock receiver*, RCDLR). KES i RCDL sustavi najčešće su iskorištavani sustavi u slučaju krađe vozila, često putem *relay* napada [52, 53].

2.1.2. Domena informacija, zabave i povezivosti

Suprotno domenama pogona, šasije i kabine, domena informacija, zabave i povezivosti sadrži sustave čije funkcionalnosti većinom nisu visokog prioriteta niti su uvjetovane *realtime* zahtjevima. Sustav za informacije i zabavu, odnosno IVI, obuhvaća niz podsustava koji vozaču i putnicima pružaju informacije o stanju vozila i dodatne sadržaje.

Ploča s instrumentima (engl. *instrument cluster*), u mehaničkoj izvedbi ili kao digitalni zaslon, pruža vozaču informacije o brzini vozila, broju okretaja motora i količini preostalog goriva te stanju pokazivača smjera, tempomata i kvarova kroz određen broj indikatorskih svjetlećih dioda. U digitalnoj izvedbi, ploča s instrumentima može pri-

kazivati i druge korisne informacije, kako vozač ne bi morao skretati pogled na središnji zaslon tijekom vožnje, poput trenutne radio postaje, parking kamere te navigacije [34, 52].

Navigacija se često prikazuje i na središnjem zaslonu u sklopu grafičkog sučelja operacijskog sustava središnje jedinice (engl. *head unit*). Središnji zaslon je često osjetljiv na dodir ili je njime moguće upravljati pomoću fizičkih tipki na središnjoj konzoli. Putem grafičkog sučelja vozač i putnici vozila mogu iskoristiti mogućnost povezivanja mobilnog telefona sa sustavom za informacije i zabavu putem *Bluetooth* tehnologije ili USB-a, u svrhu reprodukcije glazbe i telefoniranja.

Noviji modeli automobila podržavaju integraciju s platformama poput *Android Auto* i *Apple Carplay*, koji omogućavaju puno dublju integraciju funkcija mobilnog telefona u sustave informacija i zabave. Povezivanjem telefona s njemu pripadajućom platformom, korisnicima automobila omogućava korištenje aplikacija za navigaciju, reprodukciju glazbe i ostalih multimedija, korištenje *web* preglednika, telefoniranje i razmjenu poruka s njihovog mobilnog uređaja. Omogućavaju i integraciju obavijesti, kalendara te pametnih asistenata [10, 11].

Primarna funkcija upravljačke jedinice za telematiku (engl. *telematics control unit*, TCU), je omogućavanje povezivosti automobila putem mobilne mreže, *Wi-Fi* i *Bluetooth* tehnologija te prijam GPS signala [52]. Povezivost putem *Wi-Fi* tehnologije i mobilne mreže koristi se u svrhu transmisije telemetrijskih podataka, preuzimanja OTA ažuriranja te komunikaciju s *backend* servisima proizvođača, ali i u svrhu zaštite putnika pozivanjem hitnih službi u slučaju prometne nesreće.

Operacijski sustav središnje jedinice najčešće je utemeljen na *Linux*, *QNX* i *Android* operacijskim sustavima ili vlastitom operacijskom sustavu proizvođača (engl. *proprietary*) [52]. Operacijski sustav objedinjuje sve navedene funkcionalnosti te omogućava upravljanje istima putem središnjeg zaslona, konzole i te putem glasa [34]. Zbog visoke razine kompleksnosti i povezivosti, središnja jedinica ima najveću površinu napada te je česta probojna točka u ostatak sustava automobila u mnogim istraživanjima sigurnosti automobila [9, 39, 64, 42, 50].

2.1.3. Ostale komponente

Od ostalih komponenti E/E mreže, bitno je spomenuti OBD-II priključak za dijagnostiku. OBD-II priključak omogućava serviserima brzo očitavanje dijagnostičkih kodova neispravnosti (engl. *diagnostic trouble code*, DTC) te se u osobnim automobilima naj-

češće nalazi ispod upravljača automobila [64]. Uz navedeno, OBD-II priključak omogućava pristup CAN-u automobila te ukoliko je mreža neispravno segmentirana moguće je putem OBD-II priključka komunicirati s ECU-ovima kritičnima za sigurnost putnika [39, 64]. Potencijalni napadač koji već ima pristup unutrašnjosti automobila može iskoristiti neispravnu segregaciju mreže. Primjerice, spajanjem zloćudnog uređaja na OBD-II priključak može onemogućiti upravljački modul elektroničkog kočenja tijekom vožnje, ugrožavajući fizičku sigurnost putika.

2.2. Tipovi E/E arhitektura

Tip E/E arhitekture može utjecati na površinu napada vozila te omogućiti ili onemogućiti određene vektore napada. U slučaju potpuno nesegmentiranih E/E arhitektura, što je karakteristično za starije modele automobila, kompromitiranjem jednog od sustava automobila moguće je komunicirati sa svim ostalim sustavima. Jedan od modela s nesegregiranom E/E arhitekturom je *Jeep Cherokeee* iz 2014. godine kojeg su uspjeli kompromitirati Miller i Valasek, nakon čega su proizvođači segregaciju E/E arhitektura krenuli provoditi i na jeftinijim modelima [50, 23]

U [34] autori razlikuju tri tipa E/E arhitektura, navedene povijesnim redoslijedom:

- Funkcionalno raspodijeljena E/E arhitektura (engl. *Functionally distributed E/E architecture*)
- E/E arhitektura centralizirana po domenama (engl. *Domain-centralized E/E architecture*)
- Zonalna E/E arhitektura (engl. *Zone E/E architecture*)

U današnjim osobnim automobilima je još uvijek je najčešća funkcionalno raspodijeljena arhitektura najčešća, dok se E/E arhitektura centralizirana po domenama sve češće pojavljuje u novijim modelima vozila. Zonalna E/E arhitektura je još uvijek u razvoju, ali se smatra da će zbog svoje skalabilnosti postati dominantna [34, 52, 23].

2.2.1. Funkcionalno raspodijeljena E/E arhitekture

U funkcionalno raspodijeljenim E/E arhitekturama, grupirani su ECU-ovi sličnih funkcionalnosti te su međusobno povezani CAN ili FlexRay sabirnicama. Pojedinačne sabirnice mogu biti povezane središnjim poveznikom (engl. *central gateway*). Središnji

poveznik vrši funkciju koordiniranja ponašanja između ECU-ova različitih sabirnica te segregiranja ECU-ova kritičnih za sigurnost putnika od ECU-ova manje kritičnih funkcionalnosti, a nekada i veće površine napada, primjerice ECU-ova iz domene informacija, zabave i povezivosti.

2.2.2. E/E arhitektura centralizirana po domenama

E/E arhitektura centralizirana po domenama uvodi domenske upravljačke jedinice (engl. *domain control unit*, DCU), odnosno ECU-ove s više procesorske snage, koje obnašaju funkciju više manjih ECU-ova iz iste domene. Opcionalno, DCU-ovi različitih domena mogu biti povezani središnjim poveznikom ili izravno, najčešće putem *Ethernet* [34, 52].

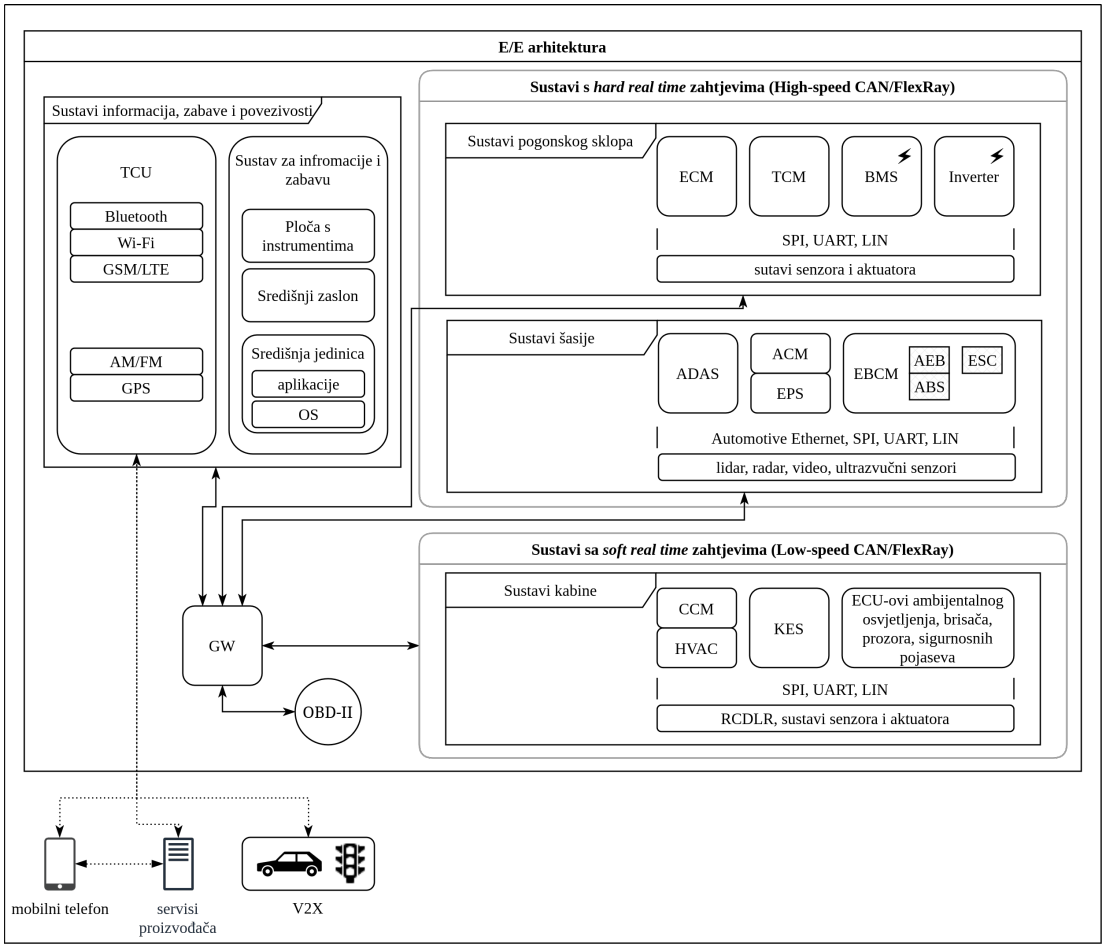
2.2.3. Zonalna E/E arhitektura

Zonalna E/E arhitektura centralizira logiku u središnjem računalu vozila (engl. *central vehicle computer*), uvodi zonalne upravljače te ih dijeli prema fizičkoj poziciji unutar vozila. Prednost zonalne arhitekture je dobra skalabilnost pri povećanju broja senzora [34]. S obzirom na to da se senzori mogu nalaziti u bilo kojem dijelu vozila, izravno ih povezivati na ECU-ove kojima pružaju podatke nije jednostavno izvedivo. Zonalni upravljači služe kao agregatori podataka sa senzora te ih prosljeđuju središnjem računalu vozila koje ih procesuira ili prosljeđuje u druge zone. Središnje računalo vozila sastoji se od više centralnih i grafičkih procesorskih jedinica te jezgara za zadatke s *realtime* vremenskim zahtjevima, kako bi podržale računalno intenzivne zadatke poput autonomne vožnje [52].

2.3. Generalizirana E/E arhitektura modernog automobila

Na slici 2.1 prikazana je generalizirana E/E arhitektura modernog automobila. Specifična toplogija mreže automobila je izostavljena, već je prikazana samo podjela sustava vozila. Sustavi različitih domena su grupirani prema vremenskim zahtjevima zadataka koje obavljaju. Sustavi iz domena pogonskog sklopa i šasije skupa su kategorizirani u sustave s *hard realtime* zahtjevima, jer je njihov rad kritičan za fizičku sigurnost vozača i putnika te ispravan rad automobila. Sustavi iz domene informacija, zabave i povezivosti odvojeni su zasebno jer zadaci koje obavljaju ne zahtjevaju rješavanje u

stvarnom vremenu. Unutar svake od domena navedeni su najbitniji ECU-ovi, a ispod njih senzori, aktuatori i ostale podkomponente s kojima komuniciraju. Najčešće korišteni protokoli za upravljanje i komunikaciju sa senzorima i aktuatorima navedeni su između paralelnih linija koje ih spajaju s pripadajućim ECU-ovima. Primjerice, od sustava šasije prikazani su upravljački modul elektroničkog kočenja (EBCM), napredni sustav za podršku vozaču pri upravljanju vozilom (ADAS), upravljački modul zračnih jastuka (ACM) te sustav servo upravljača (EPS). U sklopu EBCM-a prikazane su i njegove glavne funkcionalnosti automatskog hitnog kočenja (AEB), elektroničkom kontrolom stabilnosti (ESC) i ABS-a. S obzirom na to da ADAS vrlo često koristi Ethernet za prijenos video tokova, naveden je kao jedan od protokola za komunikaciju s podkomponentama uz protokole UART, SPI i LIN [52]. Sustavi specifični pojedinim tipovima E/E arhitektura poput središnjeg računala vozila, domenske upravljačke jedinice te zonalnih upravljača nisu prikazani, već je odabran središnji poveznik kao apstrakcija navedenih triju arhitektura, odnosno apstrakcija međusobne povezanosti tih sustava. Prikazana je i povezanost OBD-II dijagnostičkog priključka s centralnim poveznikom, jer je česta točka proboja u ostale sustave automobila [39, 64]. Prikazana je i razmjena podataka između upravljačke jedinice za telematiku i vanjskih sustava. Pod vanjske sustave spadaju mobilni telefoni, servisi proizvođača poput servisa za OTA ažuriranja i prikupljanja telemetrije te ostalih vozila i infrastrukture odnosno V2X sustava. S obzirom na to da proizvođači često korisnicima pružaju mogućnost upravljanja vozilom putem mobilne aplikacije, koja najčešće s vozilom komunicira putem internetskih aplikacijskih programskih sučelja proizvođača, naznačena je i komunikacija između mobilnog telefona i servisa proizvođača [21].



Slika 2.1: Generalizirana arhitektura modernog automobila

2.4. Protokoli fizičkog sloja i sloja podatkovne poveznice

2.4.1. CAN

Protokol *Controller area network* je serijski komunikacijski protokol sa svojstvima pogodnima za komunikaciju sustava u stvarnom vremenu [33]. Predstavili su ga inženjeri tvrtke *Robert Bosch GmbH* 1986. godine, na konferenciji *Society of Automotive Engineers* (SAE) u Detroitu, a prvu primjenu u automobilskoj industriji imao je u modelu *Mercedes-Benz W140*, puštenom na tržište 1991. godine te je danas standard za komunikaciju u motornim vozilima, ali se pojavljuje i u industrijskim upravljačkim sustavima [34, 4]. Specifikacija CAN-a tvrtke *Robert Bosch GmbH* standardizirana je u seriji standarda ISO 11898.

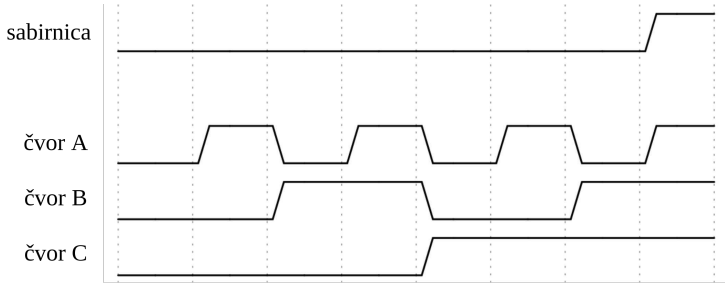
Prema [33] CAN se može podijeliti u 3 sloja:

- objektni sloj (engl. *object layer*)
- prijenosni sloj(engl. *transfer layer*)
- fizički sloj (engl. *physical layer*)

Prema OSI modelu, objektni i prijenosni sloj odgovaraju sloju podatkovne poveznice, a fizički sloj fizičkom. Objektni sloj definira filtriranje primljenih poruka te određivanje poruka koje treba prenijeti. Prijenosni sloj obuhvaća funkcije upravljanja okvirima, arbitraže, signaliziranja i provjeravanja grešaka. Fizički sloj definira kako se zapravo i kojim prijenosnim medijem signali prenose. Glavna svojstva protokola CAN su prioritizacija poruka putem nedestruktivne arbitraže, garancija vremena latencije, mogućnost postojanja više *master* čvorova, detekcija i signalizacija grešaka, razlikovanje privremenih grešaka i trajnih ispada te automatsko isključivanje neispravnih čvorova.

Na razni fizičkog sloja, razlikujemo CAN visoke brzine (engl. *high-speed CAN*) i CAN niske brzine (engl. *low-speed CAN*), definirani u ISO 11898-2 i ISO 11898-3. CAN visoke brzine može postići brzinu prijenosa podataka do 1 Mbit/s te se koristi za komunikaciju između kritičnijih čvorova, dok je CAN niske brzine tolerantan na greške te ima maksimalnu brzinu prijenosa podataka od 125kbit/s [34]. Prijenosni medij je parica, koja može biti upredena ili neupredena. Podaci se šalju putem dva komplementarna signala, *CAN Low* (CAN_L) i *CAN High* (CAN_H) koji čine diferencijalni par. Razlika između napona signala CAN_H i CAN_L definira logičko stanje sabirnice. Diferencijalni signal čini CAN otpornijim na interferenciju, jer će interferencija utjecati na oba signala, ali ne i na njihovu razliku[34].

Nisko logičko stanje odnosno logička „0“ je dominantno stanje, a visoko logičko stanje, logička „1“, je recesivno stanje. Logička razina sabirnice u određenom trenutku može se odrediti kao logički umnožak stanja koja na sabirnicu postavljaju čvorovi u tom trenutku. Odnosno, razina sabirnice bit će u dominantnom niskom logičkom stanju u slučaju da barem jedan od čvorova postavlja dominantnu logičku „0“, a u suprotnom biti će u recesivnom stanju, prikazano na primjeru s 3 čvora na slici 2.2.



Slika 2.2: Logička stanja CAN sabirnice

Opisano ponašanje omogućava sustav arbitraže i prioritizacije CAN poruka, kojim se određuje koji čvor ima prednost pri prenošenju podataka u slučaju istovremenog početka prijenosa poruka više čvorova. Svaki CAN okvir započinje bitom za početak okvira (engl. start-of-frame bit, SoF) te arbitražnog identifikatora (engl. *arbitration identifier*). Primjerice, kada dva čvora istovremeno krenu prenositi svoje poruke, odnosno prvo njihove identifikatore, oni će ih prenositi bit po bit sve dok prvi čvor ne zamijeti razliku između postavljenog i stvarnog stanja. Razlika se pojavljuje jer je drugi čvor postavio dominantnu logičku „0“ kada je prvi htio postaviti recesivnu logičku „1“. Nakon toga prvi čvor prepušta sabirnicu te se prebacuje u stanje čitanja, a drugi čvor prenosi svoju poruku do kraja. Nakon što sabirnica ponovno postane slobodna, prvi čvor će pokušati napraviti retransmisiju svoje poruke. Sukladno tome, poruke koje imaju manji identifikator, imaju veći prioritet, jer imaju duži neprekinuti niz nula na početku okvira. Primjer s razrješavanjem kolizije između dva čvora dalje se može proširiti na veći broj čvorova, gdje prioritet dobiva čvor čija poruka ima najmanji identifikator, a ostali čekaju završetak prijenosa. Kada se sabirnica ponovno oslobodi, ostali čvorovi započinju retransmisiju te se postupak arbitraže ponavlja.

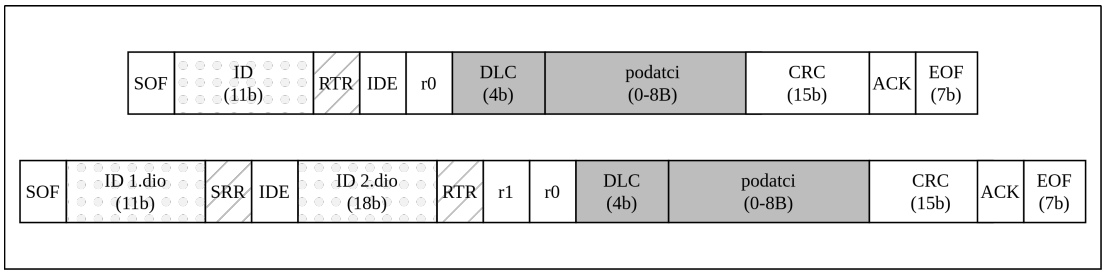
Boschova specifikacija inačice 2.0 CAN-a iz 1991. godine dijeli se na dijelove A i B, odnosno CAN 2.0A i CAN 2.0B [33]. Razlikuju se u podržanoj duljini identifikatora poruke, gdje CAN 2.0B podržava standardne poruke s 11-bitnim identifikatorima i proširene poruke s 29-bitnim identifikatorima, dok CAN 2.0B podržava samo standardne poruke. Razlikujemo 4 vrste CAN okvira:

- podatkovni okvir (engl. *data frame*)
- okvir greške (engl. *error frame*)
- okvir zahtjeva (engl. *remote frame*)
- okvir preopterećenja (engl. *overload frame*)

Glavni dijelovi podatkovnog okvira prikazanog na slici 2.3 su:

1. bit početka okvira (engl. *Start-of-Frame bit, SOF*)
2. arbitražni identifikator (engl. *arbitration ID*)
3. bit zahtjeva za prijenosom (engl. *remote transmission request*, RTR)
4. bit proširenja identifikatora (engl. *identifier extension bit*, IDE)
5. kod duljine podataka (engl. *data length code*, DLC)
6. podatkovno polje
7. kod cikličke provjere zalihosti (engl. cyclic redundancy check)
8. bit potvrde (*acknowledge slot*, ACK)
9. kraj okvira (engl. *End-of-Frame*, EOF)

Uz navedene, na slici 2.3 pojavljuju se substitucijski bit RTR bita (engl. *substitute remote request*, SSR) i bitovi r0 i r1 rezervirani za moguća proširenja protokola. Arbitražni identifikator poruke duljine je 11 bita za standardne poruke (CAN 2.0A) i 29 bita za proširene poruke. Poruke s nižom vrijednosti identifikatora imaju veći prioritet pri arbitraži. Skupa s RTR bitom čini arbitražno polje. RTR bit u recesivnoj „1“ označava okvir zahtjeva, a u dominantnoj „0“ podatkovni okvir. Ovakvim označavanjem okvira postiže se prednost podatkovnog okvira pri arbitraži nad okvirom zahtjeva s istim arbitražnim identifikatorom. Za specificiranje duljine podataka u oktetima koristi se 4-bitno DLC polje od kojih se zapravo koristi 3 bita, s obzirom na to da je duljina podataka ograničena na 8 okteta. Nakon polja s podacima nalaze se CRC kod i ACK bit. Čvor koji prenosi okvir postavlja ACK bit u recesivnu „1“ jer očekuje da će bar jedan čvor potvrditi ispravnost poruke postavljanjem stanja sabirnice u dominantnu „0“. Naposljetku se prenosi niz od 7 recesivnih „1“ kao oznaka kraja okvira (EOF).



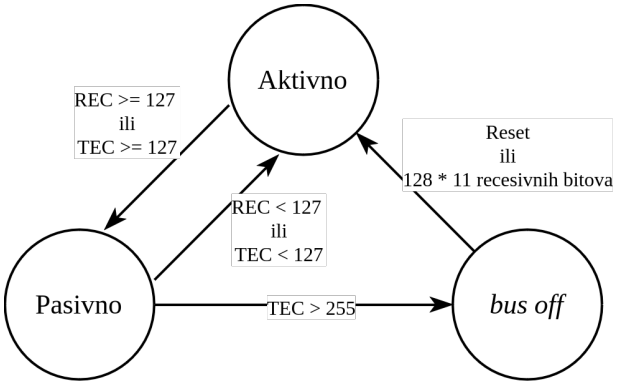
Slika 2.3: Standardni i prošireni CAN okvir

Prema [33] CAN protokol razlikuje 5 vrsta grešaka:

- greška bita (engl. *bit error*)
- greška umetnutog bita (engl. *stuff error*)
- CRC greška (engl. *CRC error*)
- greška formata (engl. *form error*)
- greška potvrde (engl. *acknowledgment error*)

Svaki vrsta greške od navedenih odgovara jednoj od metoda za detekciju grešaka koje su dio CAN protokola. Svaki CAN primopredajnik tijekom prijenosa poruke osluškuje sabirnicu bit po bit kako bi potvrdio da se njegova poruka ispravno prenosi. Ako se postavljeni bit i stvarni bit na sabirnici razlikuju, osim u periodu arbitraže, CAN primopredajnik će detektirati grešku bita. CAN protokol koristi i *bit stuffing* metodu detekcije grešaka, odnosno umetanje dodatnog bita suprotnog polariteta nakon 5 bitova istog polariteta. Dodatne bitove uklanja primopredajnik čvora primatelja. Greškom umetnutog bita smatra se pojavljivanje 6 bita istog polariteta. Svaki čvor primatelj izračunava CRC kod primljene poruke, a razlika u izračunatom i primljenom CRC kodu smatra se CRC greškom. CAN okviri sadrže nekoliko fiksnih bitova, čija neispravnost naznačava grešku formata. Naposljetku, grešku potvrde detektira čvor pošiljatelj kada niti jedan čvor primatelj nije postavio sabirnicu u dominantnu „0“ tijekom perioda ACK bita.

Svaki čvor interno održava dva brojača grešaka, brojač grešaka pri prijenosu (engl. *transmit error counter*, TEC) te brojač grešaka pri primitku (engl. *receive error counter*, REC). Brojači se koriste u sklopu CAN-ovog mehanizma lokalizacije grešaka (engl. *error confinment mechanism*). U kontekstu grešaka, čvorovi mogu biti u aktivnom, pasivnom i stanju ispada (engl. *bus off*). Trenutno stanje čvora ovisi o vrijednostima TEC i REC brojača, prikazano konačnim automatom stanja na slici 2.4. Kada čvor u aktivnom stanju detektira grešku postavlja aktivnu zastavicu greške u obliku 6 dominatnih bitova koji će sigurno trenutni okvir učiniti nevažećim zbog metode *bit stuffing* te će ostali čvorovi u aktivnom stanju analogno reagirati postavljanjem svojih zastavica. Ovisno na kojem od 6 bitova prve zastavice ostali čvorovi detektiraju grešku, duljina cjelokupnog okvira greške može biti između 6 i 12 bitova. Čvor u pasivnom stanju u slučaju detektiranja greške postavlja pasivnu zastavicu greške u obliku 6 recesivnih bitova, u svrhu samoprovjere čvora, što može rezultirati u promjeni stanja brojača.



Slika 2.4: Stanja CAN čvora

S obzirom na to da je maksimalna brzina prijenosa podataka klasičnog CAN-a 1 Mbit/s s ograničenjem od maksimalno 8 okteta podataka po poruci, tvrtka *Robert Bosch GmbH* uvodi CAN s fleksibilnom brzinom prijenosa podataka (engl. *CAN with Flexible Data-Rate*, CAN-FD) kao proširenje CAN-a. CAN-FD iskorištava sva četiri bita DLC polja za kodiranje duljine podataka, povećavajući maksimalnu duljinu polja podataka jednog okvira s 8 na 64 okteta. Uz navedeno, omogućava dinamično povećanje brzine prijenosa, uobičajeno do 2Mbit/s, isključivo za vrijeme trajanja prijenosa podataka nekog CAN okvira [34, 52].

2.4.2. FlexRay

Zbog potrebe za više deterministčkim komunikacijskim protokolom s garantiranom propusnosti za kritične poruke, stvoren je FlexRay [52]. FlexRay za razliku od CAN-a, koji koristi *Code Division Multiple Access* metodu dodjeljivanja pristupa sabirnici, FlexRay koristi *Time Division Multiple Access* metodu koja dodjeljuje pristup za prijenos određenih poruka po vremenskim odsječcima u dva kanala. Poruke mogu biti periodične, fiksne duljine te se za njih dodjeljuju statički vremenski odsječci ili neperiodične, varijabilne duljine te se onda prenose u dinamičkom vremenskom odsječku [52]. Maksimalna duljina polja podataka jedne poruke je 254 okteta, što omogućava brzinu prijenosa podataka do 10Mbit/s u slučaju konfiguracije s jednim redundantnim kanalom. Ukoliko komunikacija nije kritična, redundantni kanal moguće je iskoristiti za udvostručenje brzine prijenosa podataka do 20Mbit/s [52, 34]. FlexRay mreže mogu biti organizirane u sabirničku topologiju ili topologiju zvijezde.

2.4.3. Ethernet

Pojava sustava s potrebama za još većom brzinom prijenosa podataka, potaknula je proizvođače na uporabu Etherneta prilagođenog za primjenu u automobilskoj industriji (engl. *automotive Ethernet*) u internim mrežama svojih vozila [52, 34]. Brzine prijenosa podataka kreću se u rasponu od 10Mbit/s do 10Gbit/s, što Ethernet čini priladnim za vremenski osjetljivu komunikaciju, poput povezivanja domenskih upravljačkih jedinica te za primjene koje zahtjevaju veliku propusnost, primjerice prijenos video toka između ADAS-a i video senzora. Ethernet također nema sigurnosne probleme koje donosi CAN zbog kompatibilnosti s postojećim tehnologijama poput IPsec-a, MACsec-a te TLS-a [73].

Najveće prilagodbe za automobilsku industriju pojavljuju se na fizičkom sloju, koji mora biti prilagođen standardima elektromagnetske kompatibilnosti (engl. *electromagnetic compatibility*, EMC).

2.4.4. LIN

Local Interconnect Network sabirnica najčešću primjernu ima u povezivanju senzora i aktuatora s pripadajućim ECU-ovima . Primjerice, koristi se u sustavima domene šasije za upravljanje bravom vrata, podizanjem i spuštanjem prozora te namještanjem zrcala retrovizora [34, 52, 23]. U automobilima LIN komunikacija odvija se između *master* ECU-a, koji zadaje naredbe i *slave* senzora i aktuatora koji ih obrađuju i po potrebi odgovaraju. Komunikaciju pokreće *master* prijenosom zaglavlja koje sadrži sinkronizacijske bitove te identifikator naredbe. *Slave* odgovara prijenosom podataka maksimalne duljine 8 okteta i kontrolne sume. Iako je sporiji od CAN-a s brzinom prijenosa podataka do 20kbit/s te podržava manje fleksibilnu konfiguraciju od maksimalno 16 čvorova, prednost LIN-a je jeftin prijenosni medij u obliku jednožilnog kabela.

2.5. Dijagnostički i kalibracijski protokoli

Dijagnostički i kalibracijski protokoli su protokoli aplikacijskog sloja OSI modela te mogu funkcionirati povrh različitih protokola nižih slojeva, ali su u ovom radu razmatrani u kontekstu CAN-a. U ovom poglavlju opisani su dijagnostički protokoli *Unified Diagnostic Services* (UDS) i *On Board Diagnostics II* (OBD-II) te univerzalni mjerni i kalibracijski protokol (engl. *Universal Measurement and Calibration Proto-*

col, XCP).

2.5.1. Transportni protokol ISO-15765-2

Preduvjet za ispravnu komunikaciju putem protokola UDS i OBD-II je povećanje maksimalne duljine prenesenih podataka u jednom CAN okviru. U tu svrhu, kroz standard ISO 15765, uveden je ISO 15765 transportni protokol (ISO-TP) koji omogućava pouzdan prijenos podataka veličine do 4GB preko CAN-a ili CAN-FD-a [22]. ISO-TP uvodi 6 načina adresiranja čvorova, gdje se pri korištenju normalnog načina adresiranja svakom čvoru pridjeljuju dva arbitražna identifikatora, za primanje i slanje poruka.

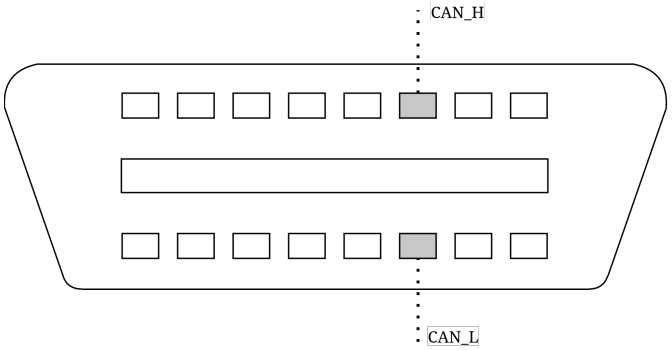
Podaci veći od maksimalne veličine polja podataka CAN ili CAN-FD okvira, fragmentiraju se u jedan *First Frame* (FF) okvir te u više uzastopnih *Consecutive Frame* (CF) okvira. Nakon FF okvira te prije primitka ostatka podataka kroz više CF okvira, primatelj mora prvo poslati *Flow Control* okvir, kojim definira parametre veličine bloka i vremenskog razmaka. U slučaju da su podaci manji od maksimalne veličine polja podataka CAN ili CAN-FD okvira, prenose se *Single Frame* okvirom.

2.5.2. OBD-II

Zahtjev za uvođenjem dijagnostike u svrhu kontrole emisija u svim novim automobilima izdao je *California Air Resources Board* u Kaliforniji 1991. godine. Navedena dijagnostika je nazvana *On Board Diagnostics* (OBD) te je 1994. standardizirana pod nazivom OBD-II. OBD-II definira standardizirani skup dijagnostičkih informacija koje svaki ECU mora čuvati te priključak i komunikacijski protokol za pristup navedenom skupu [34].

Od 2014. godine u Europskoj Uniji dozovoljeno je samo korištenje CAN-a na fizičkom sloju i sloju podatkovne poveznice te protokola ISO-TP na transportnom sloju, što je u skladu sa standardom ISO 15767 [34]. Međutim, priključak OBD-II uz CAN podržava i protokole *Keyword Protocol 2000*, VPW i PWM te protokol standarda ISO 9141-2. Shema priključka OBD-II s označenim CAN kontaktima prikazana je na slici 2.5 [29].

Zahtjevi u protokolu OBD-II šalju se s arbitražnim identifikatorom `0x7DF`, a odgovori na zahtjeve s identifikatorima `0x7E8` do `0x7EF`. Dijagnostičke informacije poput trenutne brzine, broja okretaja te temperature motora, moguće je dohvatiti korištenjem



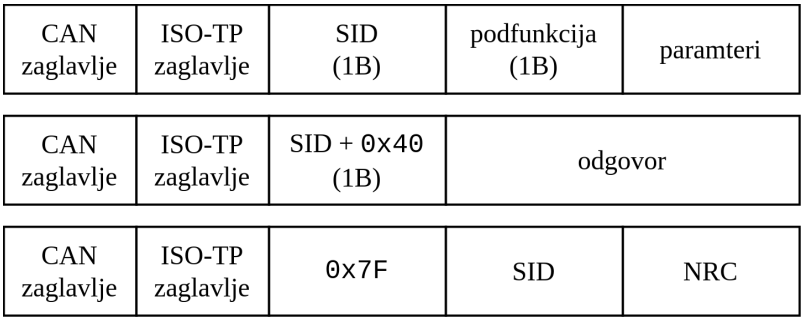
Slika 2.5: Shema OBD-II priključka

različitih servisa s njihovim paramtarskim identifikatorima (engl. *Parameter identificator*, PID). Specifičan je servis `03`, koji nudi isčitavanje dijagnostičkih kodova neispravnosti (engl. *diagnostic trouble code*, DTC) te servis `04` koji omogućava njihovo brisanje [31].

2.5.3. UDS

Unified Diagnostic Services (UDS) je protokol aplikacijskog sloja koji nudi niz dijagnostičkih servisa koje je moguće implementirati za pojedini ECU. Standardiziran je u ISO 14229 te povrh CAN-a koristi ISO-TP na transportnom sloju [25].

ECU se u UDS komunikaciji ponaša kao poslužitelj, dok je klijent najčešće dijagnostički alat servisera ili programska podrška za konfiguraciju ECU-ova u tvornici. Kroz niz UDS servisa moguće je upravljati stanjem ili ponovno pokrenuti ECU, čitati i brisati DTC-ove, isčitavati i modificirati parametre ECU-a, testirati značajke ECU-a korištenjem ugrađenih rutina te isčitavati i modificirati sadržaj memorije, prvenstveno u svrhu ažuriranja [30]. UDS zahtjev sastoji se od identifikatora servisa (engl. *service identifier*, SID), okteta podfunkcije odabranoga servisa, skupa s potrebnim parametrima, prikazano na slici 2.6. Ukoliko je odgovor pozitivan, sadržavat će SID uvećan za 0x40 te podatke odgovora specifične tom servisu. U slučaju negativnog odogvora, odgovor počinje s oketom `0x7F` te sadrži SID zahtjevanog servisa te kod negativnog odgovora (engl. *negative response code*, NRC), primjerice kod `0x11` u slučaju da ECU ne podržava zahtijevani servis.



Slika 2.6: UDS zahtjev, pozitivan i negativan odgovor

Između 27 dostupnih UDS servisa, kao sigurnosno kritične UDS servise treba izdvojiti:

- `0x10 DiagnosticSessionControl`
- `0x11 ECUReset`
- `0x28 CommunicationControl`

- `0x22 ReadDataByIdentifier`
- `0x23 ReadMemoryByAddress`
- `0x2E WriteDataByIdentifier`
- `0x3D WriteMemoryByAddress`

- `0x34 RequestDownload`
- `0x35 RequestUpload`
- `0x38 RequestFileTransfer`

- `0x2F InputOutputControlByIdentifier`
- `0x31 RoutineControl`

- `0x27 SecurityAccess`
- `0x29 Authentication`

Servis `DiagnosticSessionControl` koristi se za izmjenu vrste dijagnostičke sesije. Dijagnostička sesija određuje kontekst izvršavanja servisa, primjerice kojem dijelu memorije se pristupa i koji dio izvršnog koda se može ažurirati. Također, servisi mogu biti podijeljeni po različitim vrstama dijagnostičkih sesija, čime se omogućava ili onemogućava njihovo korištenje. Servis `ECUReset` koristi se za pokretanje više različitih vrsta resetova ECU-ova. Vrstu reseta određuje oktet podfunkcije. Zabrana primanja i slanja poruka nekom ECU-u može se uvesti korištenjem servisa `CommunicationControl`.

Servisi `ReadDataByIdentifier` i `WriteDataByIdentifier` koriste se u svrhu čitanja i pisanja podataka spremljenima pod određenim podatkovnim identifikatorima (engl. *data identifier*, DID). Značenje većine identifikatora određuje proizvođač. Za čitanje i pisanje u memoriju ECU-a koriste se servisi `ReadMemoryByAddress` i `WriteMemoryByAddress`, gdje dostupnost dijelova memorije ovisi o konkretnoj implementaciji [30, 25].

Skupini servisa `0x34` do `0x38` koji služe za preuzimanje i prijenos podataka pripadaju servisi `RequestUpload`, `RequestDownload` te `RequestFileTransfer`. Servisi `RequestUpload` i `RequestDownload` koriste se za započinjanje prijenosa izvršnog koda s testirane jedinice prema klijentu te obrnuto. Servis `RequestFileTransfer` koristi se za preuzimanje ili prijenos datoteka te dohvaćanje informacija o datotečnom sustavu ECU-a. Navedeni servisi često služe za distribuciju OTA ažuriranja pojedinim ECU-ovima nakon što je ažuriranje preuzeto putem mobilne ili Wi-Fi mreže.

Servis `InputOutputControlByIdentifier` omogućava klijentu upravljanje signalima koje ECU odašilje, njihovo zamrzavanje na trenutnoj vrijednosti ili resetiranje na pretpostavljenu vrijednost. Servis `RoutineControl` omogućava klijentu aktivaciju i deaktivaciju pretprogramiranih testnih rutina te dohvaćanje njihovih rezultata.

Naposljetku, za sve UDS servise moguće je omogućiti autorizaciju ili autentifikaciju s više razina pristupa putem servisa `SecurityAccess` ili `Authentication`. Servis `0x27 SecurityAccess` koristi nespecificirani *seed-and-key* algoritam. Pojam *seed-and-key* upotrebljava se u UDS specifikaciji umjesto pojma izazov-odgovor (engl. *challenge-response*) koji je čest u ostalim granama kibernetičke sigurnosti. Slijed autorizacije servisom `0x27` može se prikazati kroz nekoliko koraka:

1. Klijent šalje zahtjev za autorizacijom te dodatne parametre poput zahtjevane razine pristupa.
2. ECU generira niz okteta *seed* i šalje ga klijentu.
3. Klijent i ECU koriste *seed* kao ulaz u *seed-and-key* algoritam te izračunavaju ključ.
4. Klijent šalje *seed* ECU-u.
5. ECU uspoređuje primljeni ključ s izračunatim te odobrava ili odbija pristup klijentu.

Sigurnost autorizacije putem servisa `SecurityAccess` ovisi o odabranom *seed-and-key* algoritmu. Servis `Authentication` dodan je u reviziji standarda ISO 14229-1:2020 kao moderna alternativa servisu `SecurityAccess`. Temelji se na korištenju digitalnih certifikata i infrastrukture javnog ključa (engl. *public key infrastructure, PKI* za autentifikaciju [73]:

1. Klijent šalje zahtjev za autentifikacijom te svoj digitalni certifikat s javnim ključem.
2. ECU verificira certifikat provjeravajući njegov potpis s javnim ključem certifikacijskog autortiteta kojem vjeruje.
3. ECU generira niz okteta *challenge* i šalje ga klijentu.
4. Klijent izračunava digitalni potpis koristeći *challenge* te svoj privatni ključ te ga šalje ECU-u.
5. ECU verificira ispravnost potpisa koristeći javni ključ klijenta te odobrava ili odbija pristup klijentu.

2.5.4. XCP

Protokol XCP proširenje je CAN kalibracijskog protokola (engl. *CAN Calibration Protocol*, CCP) s podrškom za Ethernet, SPI, USB, FlexRay i CAN-FD kao nižim slojevima umjesto klasičnog CAN-a [31]. Koristi se za mjerenje i kalibraciju parametara i programiranje flash memorije ECU-ova. Najčešće se koristi samo za vrijeme razvoja i testiranja ECU-a [31].

Komunikacija je oblika *master-slave*, gdje je kalibracijski alat *master*, a ECU *slave*. *Master* s ECU-om komunicira putem zahtjeva nazvanih *Command Receive Object* (CRO) te od ECU-a dobiva *Data Transmission Object* (DTO) odgovore. DTO može biti poruka odgovora na naredbu (engl. *Command Response Message*, CRM), poruka događaja (engl. *Event Message*, EV) te poruka dohvaćanja podataka (engl. *Data Acquisition Message*, DAQ).

U slučajevima kada XCP treba ostati dostupan i nakon završetka razvoja automobila, moguće je kontrolirati pristup korištenjem *seed-and-key* autorizacije. Za autorizaciju se koriste CRO-ovi `GET_SEED` i `UNLOCK`, a koraci autorizacije isti su kao i kod UDS servisa `Security Access`.

3. Ranjivosti i napadi na protokole i sustave automobila

Pri izradi CTF zadataka s ciljem edukacije stručnjaka u određenom području, potrebno je razmotriti stvarne napade, incidente, tehnike i otkrivene ranjivosti kako bi se utvrdili najčešći vektori napada te potencijalno ranjive površine napada. Implementiranjem zadataka u skladu s navedenim razmatranjima, zadaci će biti bliži konfiguracijama sustava s kojima se budući stručnjak može susresti u praksi.

U nastavku su najviše razmatrani napadi i ranjivosti iz postojećih sigurnosnih istraživanja jer najdetaljnije prikazuju tehničku razinu koju je potrebno simulirati CTF zadacima.

3.1. Ranjivosti i napadi na protokole automobila

Prethodno razmatranju napada na sustave automobila, potrebno je razumjeti inherentne ranjivosti protokola kojima oni komuniciraju. Ranjivosti ovih protokola proizlaze iz njihove specifikacije, u kojima sigurnosne mjere nisu uopće razmatrane ili nisu u potpunosti definirane.

3.1.1. CAN

U specifikaciji inačice 2.0 protokola CAN, definiran je kao „serijski komunikacijski protokol koji efikasno podržava raspodijeljeno *realtime* upravljanje s visokom razinom sigurnosti“[33]. Raširenost njegove upotrebe u *realtime* sustavima potvrđuje prvi dio navedene definicije. Međutim, po pitanju sigurnosti specifikacija ne spominje nikakve mjere. Bozdal et al. ističu u [15] da protkol CAN ne osigurava niti jedno od svojstava povjerljivosti, integriteta i dostupnosti. CAN komunikacija je po prirodi *broadcast* odnosno svi čvorovi na sabirnici mogu čitati sve poruke. CAN specifikacija ne

definira mehanizam njihovog šifriranja, čime se narušava svojstvo povjerljivosti. Uz svaku CAN poruku šalje se i njen CRC u svrhu provjere njenog integriteta odnosno detekcije grešaka. Međutim, napadaču koji je presreo CAN poruku trivijalno je ponovno izračunati CRC u slučaju da ju želi modificirati, čime se svojstvo integriteta narušava. Svojstvo dostupnosti se također može trivijalno narušiti korištenjem mehanizma CAN-ovog prioritiziranja poruka ili drugim napadima opisanima u nastavku.

Najjednostavniji napad uskraćivanja usluge (engl. *denial of service*, DoS) je napad preplavljivanja sabirnice (engl. *bus flood attack*) koji iskorištava mehanizam prioritiziranja poruka s nižom vrijednosti arbitražnog identifikatora [67]. Napadač s pristupom sabirnici, izravno, putem OBD-II priključka ili kompromitiranog ECU-a, može ju preplaviti porukama s identifikatorom „0“. Ukoliko dodatne mjere nisu implementirane, takve poruke uvijek dobivaju prioritet u postupku arbitraže sabirnice te onemogućavaju komunikaciju ostalih uređaja na sabirnici.

Bus off napad je također napad uskraćivanja usluge, koji iskorištava CAN-ov mehanizam lokalizacije grešaka [20]. Napadač s pristupom sabirnici može bilo koji ECU prebaciti u *bus off* stanje postavljanjem dominantnih bitova na sabirnicu dok ciljani ECU prenosi svoje poruke. Ovim postupkom, ECU će akumulirati velik broj grešaka pri prijenosu i povećati TEC na vrijednost veću od 255, nakon čega ECU prelazi u *bus off* stanje i prestaje emitirati poruke dok se ne oporavi.

Tindell u [67] predstavlja novi napad uskraćivanja usluge nazvan *freeze doom loop* koji iskorištava sustav signalizacije preopterećenja sabirnice. U protokolu CAN, definiran je niz bitova koji mora proći između svakog podatkovnog okvira te se naziva *inter-frame space* (IFS). Postavljanje prvog od tih bitova u dominantnu „0“, pojedini ECU može signalizirati svoje preopterećenje. Nakon signalizacije, svi ECU-ovi na sabirnici ulaze u postupak oporavka od greške te prestaju prenositi podatkovne okvire. Napadač može periodički signalizirati preopterećenje nakon svakog perioda oporavka te efektivno zamrznuti sabirnicu na proizvoljno vrijeme. Uz navedeno, signalizacija preopterećenja ne inkrementira brojače grešaka, što može biti potencijalno korisno svojstvo.

Protokol CAN je podložan i napadima lažiranjem poruka (engl. *spoofing*). S obzirom na to da u protokol nije ugrađena provjera autentičnosti poruka, napadač može bilo koju poruku postaviti na sabirnicu, lažno se predstavljajući kao čvor koji ju inače prenosi. Primjerice, ako napadač dokuči format poruke za pokretanje motora, ne postoji mehanizam koji ga sprječava da ju pošalje ECM-u umjesto imobilizatora te neovla-

šteno pokrene motor.

Ukoliko napadač želi osigurati da komunikacija za vrijeme *spoofing* napada na sabirnici izgleda legitimno, time potencijalno izbjegavajući sustave za otkrivanje napada (engl. *intrusion detection system*, IDS), napadač može prvo natjerati ciljani čvor u *error passive* stanje uništavanjem njegovih poruka postavljanjem dominantnog bita na sabirnicu za vrijeme njihovog prijenosa. U *error passive* stanju čvor nastavlja prenositi poruke, ali ne može signalizirati grešku pri prijenosu. Napadač ovo ponašanje može iskoristiti za modifikaciju ciljanih poruka, bez da čvor koji ih šalje može signalizirati grešku. Opisani napad naziva se *error passive spoofing* napadom [67].

Napadač s fizičkim pristupom sabirnici može ju fizički prerezati te spojiti prerezane krajeve na svoj zloćudni uređaj. S tako podijeljenom sabirnicom napadač može filtrirati i modificirati promet između čvorova na suprotnim krajevima sabirnice te pokretati napade tipa čovjek u sredini (engl. *Man-in-the-middle attack*, MitM).

Međutim, programska podrška za ECU-ove, kao i u drugim granama programskog inženjerstva, se najčešće implementira korištenjem razvojnih okvira (engl. *software development framework*). Pritom se CAN poruke ne čitaju izravno sa sabirnice, već kroz apstrahirano aplikacijsko programsko sučelje (engl. *application programming interface*, API) koje odabrani radni okvir pruža. Među najraširenijim razvojnim okvirima za programsku podršku ECU-ova, ističe se *Automotive Open System Architecture* (AUTOSAR). AUTOSAR nudi standardiziranu sigurnosnu arhitekturu nazvanu *Security On-board Communication* (SecOC) [12]. Dio SecOC arhitekture je i sam protokol SecOC, kojim su osigurana svojstva autentičnosti i integriteta CAN poruka te se sprječavaju navedeni *spoofing* napadi. Protokol SecOC u polje podataka CAN poruka dodaje brojač te kod za provjeru autentičnosti poruke (engl. *message authentication code*, MAC). MAC je izračunat nad cijelim podatkovnim poljem ili samo dijelom polja, korištenjem tajnog ključa dijeljenog između svih ECU-ova. Brojač koji se inkrementira svakom prenesenom porukom, služi kao dodatna zaštita kako napadač ne bi mogao ponovno poslati poruku nekog ECU-a odnosno izvesti *replay* napad.

3.1.2. UDS

Protokol UDS kao i CAN nema ugrađene mjere koje bi osiguravale povjerljivost, integritet i dostupnost tokova podataka, konkretno sadržaja UDS zahtjeva i odgovora. Zbog toga je UDS podložan istim napadima kao i protokol CAN povrh kojeg funkcionira.

Međutim, UDS kroz servise `SecurityAccess` i `Authentication` omogućava implementaciju mjera autentifikacije, autorizacije i kontrole pristupa ostalim UDS servisima. Specifikacija servisa `Authentication` zahtjeva korištenje simetrične ili asimetrične kriptografije te korištenje sigurnih kriptografskih algoritama [73]. Za razliku od servisa `Authentication`, *seed-and-key* algoritam servisa `SecurityAccess` nije definiran specifikacijom, što je glavni razlog pojave ranjivih implementacija [62, 46]. Primjerice, ako se za generiranje *seed* vrijednosti ne koristi kriptografski siguran generator pseudoslučajnih brojeva, nego se pri generiranju koristi parametar na koji napadač može utjecati, `SecurityAccess` servis postaje ranjiv na *replay* napade ili napad grubom silom. U [62] testirani `SecurityAccess` servis koristi proteklo vrijeme od posljednjeg reseta ECU-a kao parametar pri generiranju pseudoslučajne *seed* vrijednosti. Autor iskorištava navedeno kako bi pokazao mogućnost *replay* napada resetiranjem ECU-a te pravovremenim `SecurityAccess` zahtjevom. Autor pretpostavlja postojanje prethodno snimljenog ključa za poznati *seed*. U [59], pronađeno je još 6 slučajeva koji koriste ponovljive *seed* vrijednosti. S druge strane, ranjivost u implementaciji može biti prisutna i u *seed-and-key* algoritmu. Primjerice, autori [28] pronašli su implementaciju servisa `SecurityAccess` u kojoj je za *seed-and-key* algoritam korišten jedinični komplement broja, što je u slučaju poznavanja jednog para *seed*-ključ, trivijalno jednostavno prepoznati. Proizvođači se pri implementaciji *seed-and-key* algoritama često odlučuju za pristup *security-by-obscurity* [28, 59, 62].

Potrebno je istaknuti servis `ECUReset`. Ukoliko je moguće aktivirati ovaj servis tijekom kretanja vozila, na sigurnosno kritičnom ECU-u, napadač s udaljenim pristupom sabirnici može napadom uskraćivanja usluge ugroziti sigurnost vozača i putnika. U [63], pokazano je korištenje servisa `ECUReset` s ciljem prebacivanja u *bootloader* UDS sesiju, koja je koristila ranjiviju implementaciju `SecurityAccess` servisa u odnosu na standardnu. Napad uskraćivanja usluge napadač može izvesti i korištenjem servisa `CommunicationControl`, kojim može potpuno onemogućiti komunikaciju ciljanog ECU-a s ostatkom mreže.

Servise `ReadMemoryByAddress` i `WriteMemoryByAddress`, u slučaju pogrešne konfiguracije, moguće je iskoristiti za neovlašteno pisanje i čitanje memorije ECU-a. Napadač ovim putem potencijalno može dohvatiti i modificirati *firmware* ECU-a te utjecati na njegov rad.

Vrijednosti spremljene pod podatkovnim identifikatorima potpuno ovise o proizvođaču te napadač ih može čitati i modificirati korištenjem servisa `ReadDataByIdentifier`

i `WriteDataByIdentifier`. Ovisno o njihovom značenju, ovim servisima napadač može saznati dodatne informacije o stanju ECU-a, čitati dnevničke zapise te u najgorem slučaju utjecati na ponašanje sigurnosno kritičnog ECU-a.

Korištenjem servisa `InputOutputControlByIdentifier` napadač može utjecati na signale ECU-a, utječući na druge sustave automobila [25]. Servis `RoutineControl` također ovisi o proizvođačevoj implementaciji rutina koje potencijalno mogu biti ugrožavajuće za sigurnost putnika ili posredno omogućiti pristup drugim sustavima automobila.

Naposljetku, servisi `RequestDownload`, `RequestUpload`, `RequestFileTransfer`, mogu se koristiti za modifikaciju i neovlašteno čitanje firmwarea te datoteka dostupnih preko navedenih servisa. Ukoliko su ovi servisi ispravno zaštićeni servisima `SecurityAccess` i `Authentication`, firmware se potencijalno može iščitati prisluškivanjem CAN sabirnice za vrijeme OTA ažuriranja, ako samo ažuriranje nije dodatno šifrirano.

3.1.3. XCP

Protokol XCP podložan je sličnim napadima kao i ekvivalentni UDS servisi. Primjerice, implementacija `GET_SEED` i `UNLOCK` mehanizma može imati iste ranjivosti kao i ekvivalentna implementacija `SecurityAccess` servisa. Protokol XCP također podržava čitanje i modificiranje memorije kao što je to moguće s UDS-ovim servisima, ali i modifikaciju parametara koji često utječu na ponašanje ECU-a.

3.1.4. FlexRay

Kašnjenja i kolizije u FlexRay komunikaciji, zlonamjeran čvor može postići uzrokovanjem poremećaja u procesu vremenske sinkronizacije čvorova. Uz to, zlonamjeran čvor u FlexRay mreži može neprestanim zahtijevanjem dinamičkog odsječka onemogućiti njegovo korištenje drugim čvorovima. Za razliku od CAN-a, FlexRay sprječava potpuno uskraćivanje usluge alociranjem vremena za kritične poruke u statičkim odsječcima.

3.2. Napadi na automobile i sustave automobila

Prvu značajnu eksperimentalnu sigurnosnu analizu sustava automobila proveli su 2010. godine Koscher et al. [40]. Rad pretpostavlja fizički pristup CAN sabirnici. De-

monstritali su inherente ranjivosti CAN protokola te nespecificiranog dijagnostičkog protokola opisom sličnim UDS-u. Uspjeli su upravljati svim kritičnim ECU-ovima, kroz obične CAN poruke koje su identificirali neizrazitim testiranjem (engl. *fuzzy testing*, *fuzzing*) te dijagnostičke servise na stacionarnom vozilu i na vozilu u kretanju. Proveli su napade uskraćivanjem usluge korištenjem servisa sličnog UDS servisu `CommunicationControl` te pokrenuli učitavanje novog koda u *flash* memoriju ECM-a u vožnji putem servisa sličnog UDS servisu `RequestDownload`. Uz navedno, uspjeli su zaobići autorizaciju prije korištenja dijagnostičkih servisa, napadom grubom silom na *seed-and-key* algoritam te čitanjem fiksnih *seed-and-key* parova iz memorije ECU-a. Uspjeli su i upravljati pločom s instrumentima te funkcijama radija. Naposljetku, uspjeli su upravljati svim funkcijama sustava iz domene šasije poput upravljanja bravom vrata, otvaranjem prtljažnika te prozora.

Nadalje, slične napade demonstrirali su Miller i Valasek u [48], na modelima automobila *Ford Escape* te *Toyota Prius* iz 2010. godine. Analizom komunikacije na CAN sabirnici *Forda*, uspjeli su upravljati brzinomjerom, odometrom i navigacijom te upravljačem automobila. Skretanje upravljača automobila postigli su analizom komunikacije između modula za pomoć pri parkiranju (engl. *parking assist module*, PAM) i EPS-a. Uspjeli su i kompromitirati UDS servis `SecurityAccess`, reverznim inženjerstvom *Ford* dijagnostičkog alata. Nakon kompromitacije `SecurityAccess` servisa, korištenjem `RoutineControl` servisa uspjeli su isključiti motor automobila pri kretanju. Naposljetku, korištenjem `RequestDownload` servisa uspjeli su učitati vlastiti izvršni kod na jedan od ECU-ova i postavljati proizvoljne CAN okvire na sabirnicu. U slučaju *Toyote*, uspjeli su upravljati kočenjem, ubrzanjem te upravljačem automobila. Navedeno su postigli analizom CAN okvira sustava za sprječavanje sudara (engl. *Pre-Collision system*, PCS), PCM-a te sustava za automatsko praćenje prometne trake (engl. *Lane Keep Assist*, LKA).

Primjer stvarnog napada putem CAN sabirnice analizirali su Tabor i Tindell [68]. Autori su prepoznali novi tip napada na CAN sabirnicu korištenjem modificiranog CAN primopredajnika. Napadač koristi uređaj s modificiranim CAN primopredajnikom, skriven unutar prijenosnog zvučnika te ga povezuje na CAN sabirnicu iza prednjeg svjetla automobila, pritom nanoseći štetu karoseriji kako bi joj fizički pristupio. Uređaj osluškuje sabirnicu čekajući određenu CAN poruku. Nakon primitka određene CAN poruke, šalje niz CAN poruka značenja „ključ je ispravan“ te se motor automobila pokreće. Maliciozni uređaj također ima sposobnost prijenosa drugog niza poruka za otključavanje vrata. U normalnim okolnostima, poruke malicioznog uređaja poten-

cijalno bi mogle izazvati koliziju s porukama legitimnih ECU-ova zbog istovremenog prijenosa poruka istih arbitražnih identifikatora [67, 68]. Međutim, CAN primopredajnik ovog malicioznog uređaja modificiran je na način da može postaviti recesivnu „1“ na sabirnicu, iako istovremeno jedan ili više drugih čvorova postavljaju dominantnu „0“. Primopredajnik modificiran na ovaj način ima potpunu kontrolu nad sabirnicom te se ne mora ponašati u skladu s procesom arbitraže sabirnice.

Checkoway, Koscher et al. proveli su 2011. godine sigurnosnu analizu površina napada automobila, konkretnije onih površina koje bi napadačima mogle omogućiti udaljen pristup sustavima automobila [19]. Klasificirali su površine napada u kratkometne, dalekometne te neizravne kanale napada (tablica 3.1). Nad navedenim sustavima provedeno je sigurnosno testiranje te su pronađene ranjivosti koje omogućavaju niz napada poput ostvarivanja pristupa *Linux* središnjoj jedinici putem Wi-Fi-a te udaljenog izvršavanja proizvoljnog koda kroz ranjivost preljeva spremnika Bluetooth implementacije. Također su demonstrirali napad kroz reprogramirani sustav za praćenje tlaka guma (engl. *tire pressure monitoring system*, TPMS), koji je moguće udaljeno pokrenuti odašiljanjem niza specifičnih lažiranih signala koje inače odašilja senzor pritiska u gumama. Sličan napad proveli su Rouf et al. u [60] te također demonstrirali napad koji omogućava udaljenu identifikaciju vozila na temelju poruka TPMS senzora.

Tablica 3.1: Podjela kanala napada prema [19]

Neizravni	Kratkometni	Dalekometni
CD	Bluetooth	AM/FM, RDS
USB	KES	GPS
OBD-II	TPMS	mobilna mreža
	Wi-Fi	

Miller i Valasek su također napravili pregled udaljenih površina napada u [49] te na temelju njega izradili i publicirali [50]. U navedenom radu demonstrirali su udaljeni napad na nemodificirano vozilo, u ovom slučaju model *Jeep Cherokee* iz 2014. godine. Inicijalni pristup središnjoj jedinici vozila ostvarili su prvo putem Wi-Fi pristupne točke vozila, a potom putem mobilne mreže, iskorištavajući *D-Bus* servis dostupan na svim mrežnim sučeljima središnje jedinice. Već nakon ostvarenja inicijalnog pristupa, bilo je moguće upravljati radio, HVAC i GPS sustavima putem *Lua* skripti dostupnih na jedinici, bez podizanja ovlasti. Nakon ostvarivanja incijalnog pristupa, reprogramirali su *Renesas V850* procesor koji se koristi za *realtime* komunikaciju te ima pristup CAN

sabirnici. Reprogramiran je s pomoću ugrađene skripte modificiranim *firmwareom* kako bi prosljeđivao CAN okvire primljene s kompromitirane središnje jedinice putem SPI-a. Nakon reprogramiranja ostvarili su neograničen pristup CAN sabirnicama vozila te mogućnost upravljanja kritičnim sustavima automobila. Korištenjem približno istih metoda, uključujući reprogramiranje *Renesas V850* procesora, kompromitirani su i modeli *Audi* i *Volkswagen* automobila [38].

Nakon Millerovog i Valasekovog napada, koji je popularizirao ovu granu kibernetičke sigurnosti, sve su češće publikacije koje demonstriraju napade na specifične modele automobila [24]. Od 2016. godine demonstriran je niz napada na razne modele automobila tvrtke *Tesla*. Autori iz istraživačke tvrtke *Keen Security Lab* pokazali su prvi takav napad, u kojem su inicijalni pristup središnjoj jedinici ostvarili putem specifično konfigurirane Wi-Fi pristupne točke na koju se automobil automatski povezuje, kroz prethodno poznatu *WebKit* ranjivost (CVE-2011-3928) [41]. Nakon eskalacije privilegija iskorištavanjem ranjivosti u jezgri *Linux*, dobili su mogućnost reprogramiranja svih ECU-ova. Postupak reprogramiranja nije uključivao provjeru digitalnog potpisa, što im je omogućilo reprogramiranje centralnog poveznika s malicioznim *firmwareom*, kojim su ostvarili pristup svim sabirnicama automobila. Sličan napad demonstrirali su 2017. godine, ponovno iskorištavajući *WebKit* i *Linux* ranjivosti te zaobilazeći novu provjeru digitalnog potpisa kod reprogramiranja iskorištavajući ranjivost u implementaciji datotečnog sustava [54]. Iskorištavajući isti niz ranjivosti, 2019. godine demonstrirali su mogućnost udaljenog upravljanja upravljačem automobila, kompromitirajući dio komponenti ADAS-a [43].

Istraživači sigurnosti iz *Keen Security Lab* demonstrirali su slične napade i na modelima automobila *BMW* i *Mercedes-Benz*, koji se mogu generalizirati na inicijalno kompromitiranje središnje jedinice putem Wi-Fi pristupne točke ili mobilne mreže te eskalacije privilegija u svrhu omogućavanja komunikacije s kritičnim ECU-ovima [42, 16, 44].

Jedan od jednostavnijih napada na sustave za ulaz bez ključa, *relay* napad, pojavio se u posljednjem desetljeću [35]. Sustavi za ulaz bez ključa automatski otključavaju vozilo kada se odašiljač ključa nalazi u određenom dometu. Navedeno ponašanje iskorištavaju napadači, koji korištenjem pojačivača signala pojačavaju signal ključa koji inače nije dovoljno blizu automobilu da bi se automobil automatski otključao. Primjerice, pojačavanjem signala ključa koji se nalazi u domu vlasnika automobila, neovlašteno otključavaju i pokreću motor automobila. Sofistirciraniji napad na ove sustave opisan

je u [75], gdje autori iskorištavaju zastarjeli DST40 algoritam koji su koristili sustavi za ulaz bez ključa automobila tvrtke *Tesla*.

Na sklopovlje ECU-ova primjenjivi su svi uobičajeni napadi na sklopovlje ostalih ugradbenih računalnih sustava, poput napada ubacivanja greške (engl. *fault injection*) i *glitching* napada. Primjerice, Weiss i Pozzobon u [74] demonstriraju *fault injeciton* napad sa svrhom zaobilaženja `SecurityAccess` servisa i provjere potpisa *firmwarea* u *bootloader* kodu.

Naposljetku, potrebno je spomenuti napade na javno dostupne API-je proizvođača, koji omogućavaju vlasnicima upravljanje dijelom funkcija vozila putem mobilne aplikacije. Curry je demonstrirao i opisao napade na API-je i servise 12 proizvođača automobila kojima je uspio udaljeno upravljati stanjem motora, otključavanjem vrata te trubom i svjetlima [21]. Uz navedeno, uspio je i precizno locirati vozila, učitati video tok kamera automobila te sakupljati osobne podatke vlasnika.

4. Analiza postojećih edukativnih materijala

Kibernetička sigurnost automobila, kao i zajednica njenih istraživača nije toliko popularizirana i raširena kao ostale grane sigurnosti, što je moguće objasniti kroz nekoliko razloga: skuplje inicijalno ulaganje u sklopovlje i alate potrebne za istraživanje, zatvorenost koda i sklopovlja te potreba za predznanjima i iz područja računalnog inženjerstva i iz elektrotehnike. Za razliku od ove grane sigurnosti automobila, osoba koja želi postati stručnjak iz raširenije grane kibernetičke sigurnosti poput *web* sigurnosti ima dostupan niz praktičnih edukativnih materijala. Primjerice, platforme *Hack The Box*, *TryHackMe* i *Portswigger Web Security Academy* nude niz virtualnih strojeva na kojima je moguće vježbati tehnike iskorištavanja čestih *web* ranjivosti. U nastavku, razmotreni su postojeći edukativni materijali u području sigurnosti automobila.

4.1. Capture The Flag natjecanja i zadaci

4.1.1. Blockharbour VSEC platforma

Blockharbour je tvrtka koja nudi usluge penetracijskog testiranja i konzultantske usluge proizvođačima automobila i ECU-ova. Njihova VSEC platforma služi za demonstraciju njihovih *DevSecOps* proizvoda, ali i za pristup virtualnim vozilima za potrebe godišnjih CTF-ova koje održavaju [14]. Svako vozilo reprezentirano je jednim *SocketCAN* mrežnim sučeljem kojem se može pristupiti putem *web* naredbenog retka na kojem su dostupni *can-utils* alati namijenjeni za rješavanje CTF zadataka.

Na njihovoj *Proving Grounds* aplikaciji u sklopu VSEC platforme, ponuđeno je 37 stalnih CTF zadataka [13]. Od 37 zadataka, 5 ih spada u „*Getting Started*“ kategoriju, koja je namijenjena upoznavanju natjecatelja sa *SocketCAN* mrežnim sučeljem i osnovnim *can-utils* alatima. Primjerice, jedan od zadataka zahtjeva da natjecatelj koristi *can-*

dump alat kako bi pročitao ponavljajuću CAN poruku sa *SocketCAN* mrežnog sučelja. Kategorije „VSEC *HarborBay*“ i „UDS“ nude 13 zadataka u kojima natjecatelj mora iskoristiti UDS servise kako bi došao do zastavice. Do zastavice natjecatelj može doći isčitavanjem DTC-ova, resetiranjem ECU-a, čitanjem određenog dijela memorije ili pokretanjem `RoutineControl` servisa. Napredniji zadaci zahtijevaju i zaobilaženje ranjivih implementacija `SecurityAccess` servisa putem `RequestDownload` servisa. Ostale kategorije nisu usko vezane uz sigurnost sustava automobila.

4.1.2. CloudCar

CloudCar je CTF platforma izvorno postavljena u sklopu *Car Hacking Village* dijela *DEFCON30* konferencije [8, 3]. Nakon prijave, platforma učitava interaktivnu trodimenzionalnu simulaciju razvijenu u pogonskom sustavu *Unity*. Glavni ekran prikazuje automobil na trkaćoj stazi, ploču s instrumentima i dugmad za upravljanje funkcijama automobila. Dostupne funkcije najviše pripadaju domeni šasije, poput upravljanja bravom vrata, farovima, prtljažnikom, brisačima i pokazivačima smjera (slika 4.1. Kao i na platformi VSEC, natjecatelj dobiva pristup naredbenom retku virtualnog stroja s *can-utils* alatima i *SocketCAN* mrežnim sučeljem, ali putem protokola *Secure Shell* (SSH).

Slika 4.1: Snimka zaslona - platforma CloudCar

Tijekom *DEFCON30* konferencije održavalo se CTF natjecanje koje se bodovalo na dva načina:

- brojem odvoženih krugova na traci
- sakupljanjem zastavica

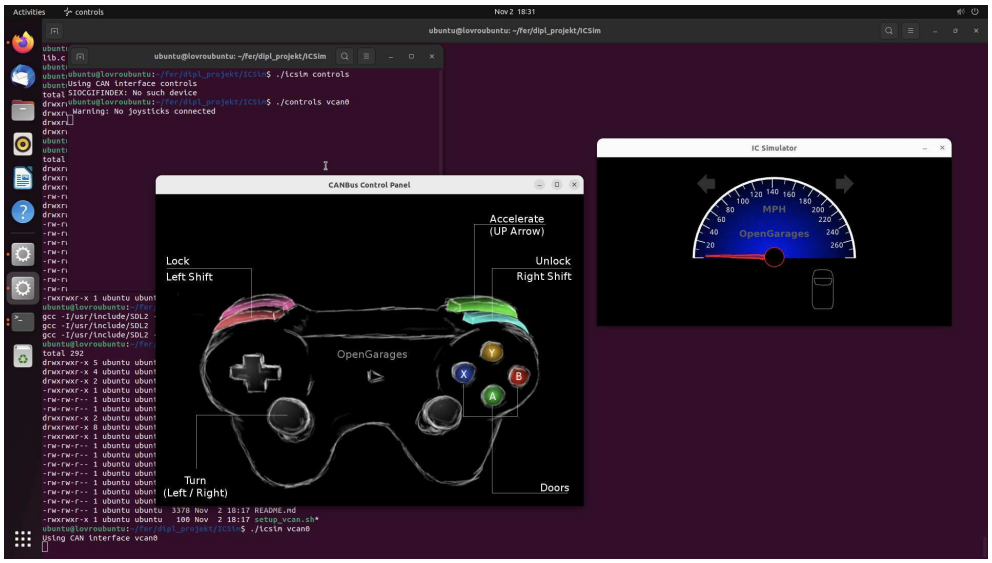
Analizom CAN prometa moguće je bilo utvrditi koje CAN poruke određuju brzinu automobila, što su natjecatelji mogli iskoristiti kako bi odvozili veći broj krugova. Pojedinačne zastavice natjecatelji su mogli sakupiti analizom CAN prometa vezanih uz druge funkcije automobila.

Konkretni zadaci na platformi više ne postoje, ali je još uvijek moguće pristupiti simulaciji i analizirati CAN poruke određenih funkcija automobila.

4.2. Simulatori

4.2.1. ICSim

ICSim je simulator ploče s instrumentima koji radi povrh jednog *SocketCAN* sučelja. Sastoji se od programa koji prikazuje ploču s instrumentima i upravljačkog programa pomoću kojeg korisnik može upravljati prikazanom brzinom, vratima i pokazivačima smjera (slika 4.2). Korisnik simulatora potom može koristiti alate kompatibilne sa *SocketCAN* sučeljima kako bi analizirao generirani promet između upravljačkog programa i programa ploče s instrumentima. Umjesto korištenja grafičkog korisničkog sučelja upravljačkog programa, korisnik može koristiti i igraći upravljač.



Slika 4.2: Snimka zaslona - ICSim

4.2.2. Toyota PASTA

Portable Automotive Security Testbed with Adaptability (PASTA) razvili su Toyama et al. s podrškom korporacije *Toyota Motor* [69, 71]. PASTA je fizički ispitni sustav otvorenog koda i sklopovlja, razvijen u svrhu omogućavanja edukacije i istraživanja sigurnosti na stvarnom sklopovlju prisutnom u automobilima. PASTA se sastoji od tri ECU-a odgovornih za domene pogonskog sklopa, šasije i kabine. ECU-ovi su povezani preko jednog središnjeg poveznika, a na središnji poveznik povezan je i OBD-II priključak. Svaki ECU je povezan na vlastiti zaslon koji prikazuje odgovarajuće informacije. Primjerice, ECU pogonskog sklopa na zaslonu prikazuje stanje prijenosa, kočenja, ubrzanja i orijentacije kotača. Ulaze ECU-ova čine fizička dugmad i sklopke kao i minijaturni upravljač automobila. Iako se može koristiti samostalno, PASTA je proširiva te moguće spojiti dodatne uređaje putem izloženog dijela CAN sabirnice. Navedena funkcionalnost demonstrirana je povezivanjem simulatora CARLA [69]. CARLA je simulator vožnje otvorenog koda izrađen u pogonskom sustavu *Unreal Engine 4*, prvenstveno namijenjen za istraživanje pristupa autonomnoj vožnji [27]. Simulator CARLA i sustav PASTA povezani su zasebnom *Python* skriptom koja pretvara CAN poruke iz sustava PASTA u ulaze za simulator CARLA [70].

5. Implementacija sustava za obuku stručnjaka sigurnosti

U sklopu ovog rada implementiran je proširivi sustav za stvaranje CTF zadataka, koji omogućava upoznavanje sadašnjih i budućih stručnjaka sigurnosti sa specifičnostima sustava u automobilima. Sustav je osmišljen i implementiran prema sljedećim zahtjevima:

1. Sustav mora imati sposobnost simulacije komunikacijskih protokola i sustava specifičnih automobilima.
2. Sustav mora biti implementiran programski, bez zahtjeva za dodatnim sklopovljem osim osobnog računala.
3. Sustav mora omogućavati stvaranje novih CTF zadataka.
4. Sustav mora biti programski proširiv.
5. Sustav mora biti poveziv sa stvarnim sklopovljem.
6. Sustav mora imati mogućnost pokretanja na osobnom računalu, ali i biti dostupan s udaljenog računala.

Zahtjevi sustava definirani su prema prednostima i nedostatcima postojećih sustava opisanih u četvrtom poglavlju. Platforme VSEC i *CloudCar* podržavaju pristup s udaljenog računala u svrhu CTF natjecanja, ali nije ih moguće pokretati lokalno, kao ni proširivati ih niti programski niti sa stvarnim sklopovljem te ne olakšavaju stvaranje novih zadataka. Sustav PASTA ispunjava većinu zahtjeva, ali zahtjeva posjedovanje sklopovlja kao i predznanja za sastavljanje sklopovlja sustava, što ga čini manje pristupačnim u odnosu na ostale sustave. Glavni doprinos implementiranog sustava je robustan način simuliranja sustava specifičnih automobilima i stvaranja CTF zadataka. Implementirani sustav namijenjen je za simulaciju CAN-a automobila, od sloja podatkovne poveznice do aplikacijskog sloja, s proizvoljnim brojem CAN sabirnica i

ECU-ova.

5.1. Korištene tehnologije

5.1.1. SocketCAN

SocketCAN je implementacija protokola CAN, CAN-FD i ISO-TP za *Linux*, koja omogućava korištenje CAN sklopovlja putem *Berkley socket* API-ja[36]. Prije *SocketCAN* paketa, komunikacija između korisničkih programa i CAN sklopovlja odvijala se izravno kroz upravljačke programe. Pritom je pri razvoju korisničkih programa bilo potrebno pisati više puta funkcionalno isti kod, za svaku podržanu vrstu CAN sklopovlja te voditi računa o specifičnostima njihovih upravljačkih programa. *SocketCAN* apstrahira korištenje upravljačkih programa kroz *Berkley socket* API. Ovakva implementacija donosi i prednosti korištenja *Linux* mrežnog stoga, poput redova okvira (engl. *queueing of frames*) i mogućnosti implementacije transportnih protokola u jezgri umjesto u korisničkom aplikacijskom (engl. *userspace*) kodu.

Temeljna ideja za ostvarivanje CAN komunikacije procesa je korištenje virtualnih *SocketCAN* sučelja kao svojevrsnih CAN sabirnica. Virtualna CAN sučelja omogućena su upravljačkim programom *vcan* te se ponašaju kao uobičajena *loopback* sučelja te se zbog tog svojstva mogu koristiti za komunikaciju između procesa (engl. *inter-process communication*, IPC). Prednost ovog pristupa je i kompatibilnost s ostalim alatima kompatibilnima sa *SocketCAN* sučeljima, poput cijelog *can-utils* paketa te alata za istraživanje sigurnosti automobila *caringcaribou* [2].

Komunikaciju između procesa putem *vcan* sučelja, moguće je demonstrirati alatima *cansend* i *candump* iz paketa *can-utils*. Alat *candump* služi za ispis svih poruka na specificiranom *SocketCAN* sučelju, a *cansend* za slanje CAN poruka putem istog. Prvo je potrebno stvoriti i podići *vcan* sučelje korištenjem alata *ip* (ispis 5.1). Potom je potrebno pokrenuti prvo alat *candump* pa alat *cansend* i predati im proizvoljnu poruku i podignuto *vcan* sučelje (ispis 5.2).

```
$ ip link add vcan0 type vcan
$ ip link set dev vcan0 up
$ ip a

(...)

7: vcan0: <NOARP,UP,LOWER_UP> mtu 72 qdisc noqueue state UNKNOWN
   group default qlen 1000
    link/can
```

Ispis 5.1: Stvaranje i podizanje *vcan* sučelja

```
// Naredbeni redak 1
$ cansend vcan0 123#ABCD
$ cansend vcan0 123#EF01

// Naredbeni redak 2
$ candump vcan0
  vcan0  123   [2]  AB CD
  vcan0  123   [2]  EF 01
```

Ispis 5.2: IPC putem *vcan* sučelja

Isti pristup komunikaciji između procesa koristi i prethodno opisani sustav *ICSim* te je skalabilan na veći broj procesa odnosno ECU programa.

5.1.2. Docker i Docker Compose

Docker je platforma otvorenog koda koja omogućava pakiranje aplikacija u kontejnere, kako bi bile prenosive i izolirane od sustava na kojem se pokreću. Ključna komponenta *Dockera* je *Docker Engine* koja upravlja kontejnerima, slikama, mrežama i volumenima pohrane.

Kontejnerizacija aplikacije vrši se kroz *Dockerfile* tekstualne datoteke, koje definiraju potrebne korake za izgradnju izvršne datoteke, instalaciju potrebnih biblioteka te pokretanje aplikacije. Potom se iz *Dockerfile* datoteke gradi *Docker* slika koja sadrži sve potrebno za pokretanje aplikacije na sustavu za koji je izgrađena. Iz slike stvara se *Docker* kontejner te se aplikacija pokreće u izoliranom okruženju, skoro potpuno odvojenom od sustava domaćina kroz mehanizme jezgre. *Docker Engine* upravlja životnim ciklusom kontejnera te oslobađa korištene resurse nakon njihova uklanjanja.

Uz navedeno, *Docker Engine* upravlja umrežavanjem kontejnera stvaranjem potrebnih sučelja i dodavanjem pravila usmjeravanja paketa.

Docker Compose je alat za definiranje i pokretanje aplikacija koje se sastoje od više kontejnera. *Docker Compose* omogućava korisnicima korištenje YAML datoteku za definiranje usluga, mreža i volumena potrebnih za aplikaciju.

5.2. Opis sustava

Sustav za stvaranje CTF zadataka, implementiran je kroz 4 komponente:

- generatorsku skriptu
- dodatak Docker mrežnom upravljačkom programu *dockercan*
- predložak programa ECU-a
- vizualnu komponentu *ic-tui*

Kao temelj sustava odabrane su tehnologije *Docker* i *Docker Compose*. Svaki ECU program izoliran je u zasebni kontejner, a međusobno su povezani kroz *Docker* mreže, gdje svaka mreža predstavlja jednu sabirnicu. Korištenje *Docker* kontejnera i mreža, omogućava definiranje CTF zadataka kroz YAML konfiguracije *Docker Composea*, što osigurava njihovo lakše pokretanje, kao i robusnost upravljanja mrežama kontejnera i resursima domaćinskog sutava neizravno putem *Docker Enginea*.

S obzirom na to da Docker podržava samo komunikaciju putem protokola TCP i UDP, napisan je dodatak Docker mrežnom upravljačkom programu (engl. *Docker network driver plugin*), nazvan *dockercan*. Dodatak *dockercan* koristi se za umrežavanje kontejnera pomoću *SocketCAN* sučelja i alata te omogućava CAN i CAN-FD *broadcast* komunikaciju između kontejnera u istoj mreži. Mreže stvorene dodatkom *dockercan* simuliraju CAN sabirnice.

Napisan je i predložak programa ECU-a, koji omogućava lakše stvaranje višedretvenog programa koji može asinkrono komunicirati i odgovarati na poruke putem protokola CAN, CAN-FD, UDS i XCP. Uz navedeno, predložak ima mogućnost definiranja komunikacije prema drugim kontejnerima putem uobičajenih *Docker* mreža. Primjerice, prema kontejneru koji sadrži GUI za vizualizaciju nekog sustava automobila ili kontejneru s posredničkim programom za integraciju s simulatorom poput simulatora CARLA.

Primjer programa koji je kompatibilan s predloškom programa ECU-a je vizualna komponenta *ic-tui*. Komponenta *ic-tui* je program koji prikazuje glavne elemente ploče s

instrumentima putem korisničkog sučelja naredbenog retka (engl. *terminal user interface*, TUI). Pruža HTTP API za ažuriranje prikazanih vrijednosti s kojim ECU predložak može komunicirati. Uz to, može se pokretati i u SSH načinu rada, što omogućava udaljeno spajanje na TUI.

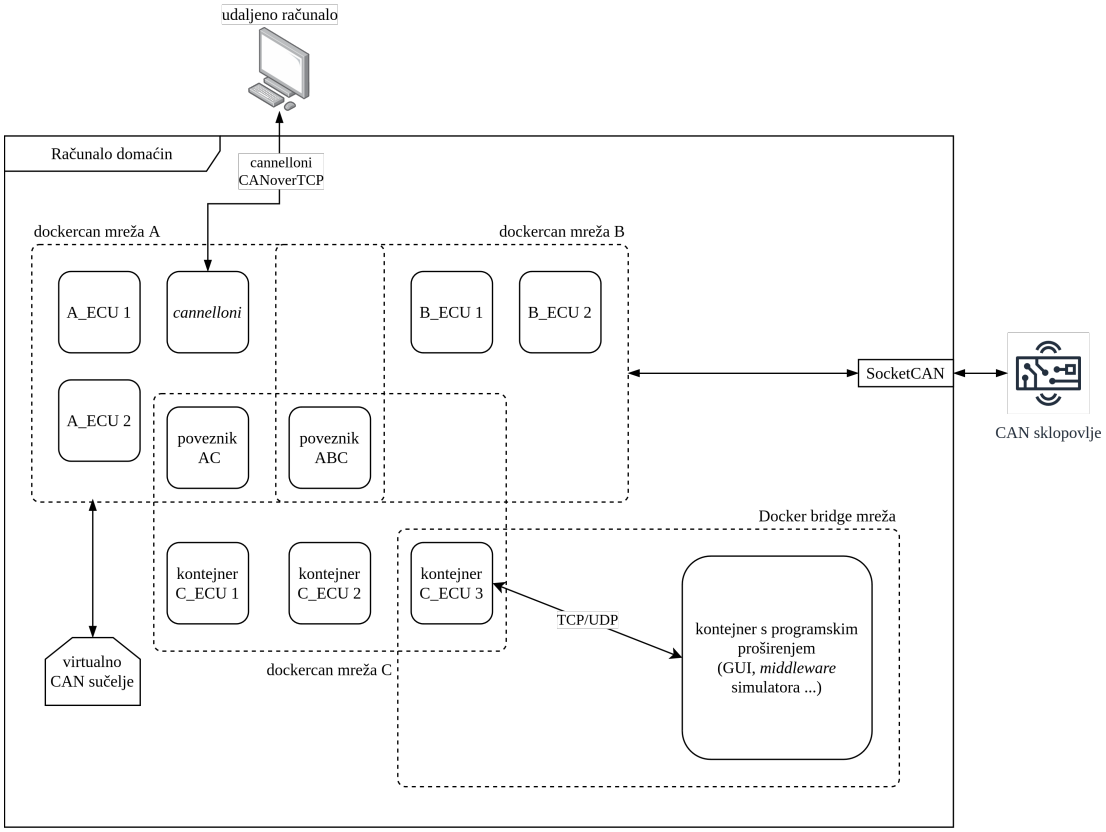
Sustav je objedinjen generatorskom skriptom, koja služi kao vodič za inicijalno postavljanje projekta CTF zadatka, kojeg korisnik potom može modificirati svojim kodom. Skripta generira i *Docker Compose* YAML datoteku, koja definira *dockercan* mreže te ECU i druge kontejnere koji su dio zadatka.

5.2.1. Primjer arhitekture generiranog zadatka

Slika 5.1 prikazuje arhitekturu s osam ECU-ova povezanih na 3 CAN sabirnice, odnosno 8 kontejnera povezanih u 3 *dockercan* mreže. Dva ECU-a poveznika nalaze se u više mreža, odnosno poveznik AC u mrežama A i C te poveznik ABC u sve tri definirane mreže, ali njihovo konkretno ponašanje definira korisnik. Primjerice, mreže A, B i C bi u svrhu CTF zadatka mogle predstavljati CAN sabirnice domene šasije, kabine i pogonskog sklopa, a poveznici imaju svrhu prosljeđivanja određenog skupa poruka između sabirnica.

Konkretna tehnička izvedba dodatka *dockercan* opisana je u poglavlju 5.5, ali potrebno je naglasiti da ima opciju stvaranja virtualnog CAN sučelja na domaćinskom računalu, kako bi se osobi koja rješava zadatak omogućio pristup određenoj CAN sabirnici. Navedeno je prikazano na *dockercan* mreži A spajanjem vanjskog CAN sučelja. Uz to, izvedba dodatka *dockercan* s pomoću *SocketCAN* mrežnih sučelja i alata, omogućava spajanje vanjskog CAN sklopovlja na mrežu. Navedeno je prikazano na primjeru *dockercan* mreže B.

Ukoliko je potrebno omogućiti udaljeno spajanje na *dockercan* mrežu, moguće je koristiti alat *cannellonni*. Alat *cannellonni* omogućava udaljeno spajanje dvaju *SocketCAN* sučelja putem TCP, UDP i SCTP protokola [58]. U generatorskoj skripti je predviđena mogućnost dodavanja prethodno konfiguriranog *cannellonni* kontejnera u svrhu udaljenog spajanja na neku *dockercan* mrežu.



Slika 5.1: Arhitektura sustava

Naposljetku, za komunikaciju s potencijalnim programskim proširenjima predviđeno je povezivanje ECU kontejnera s kontejnerima proširenja putem uobičajenih *Docker bridge* mreža. Programsko proširenje može biti *web* aplikacija koja prikazuje status rješavanja zadatka, interaktivno grafičko korisničko sučelje nalik programu upravljaču iz sustava *ICSim* ili program posrednik za povezivanje zadatka na simulator poput simulatora CARLA.

5.3. Predložak programa ECU-a

U svrhu olakšavanja izrade programa ECU-a za CTF zadatke stvoren je predložak koji služi kao početna točka. Predložak korisniku omogućava definiranje ponašanja ECU-a i odgovora na UDS, XCP i CAN poruke kroz 5 *Python* datoteka.

Predložak se sastoji od direktorija `ecu_template`, `impl` i datoteke `main.py`. Direktorij `ecu_template` sadrži interni kod predloška potrebnog za njegovu normalnu funkciju. Direktorij `impl` sadrži datoteke `can_handler.py`, `uds_handler.py` i `xcp_handler.py` u kojima korisnik može definirati željeno ponašanje u pogledu

navedena tri protokola. Uz njih, u direktoriju `impl`, nalazi se datoteka `ecu_model.py` koja služi za održavanje stanja ECU-a između prethodne tri datoteke, kao i za definiranje ponašanja ovisnog o promjeni stanja. Predložak objedinjuje datoteka `main.py` koja postavlja i pokreće program ECU-a.

5.3.1. Biblioteka Scapy i SocketCAN

Pri implementaciji predloška programa ECU-a, odabrana je biblioteka *Scapy* [6]. Odabrana je zbog mogućnosti parsiranja poruka niza komunikacijskih protokola specifičnih automobilima, kao i zbog kompatibilnosti sa *SocketCAN* sučeljima. Razmatrane su i biblioteke *python-can* te *can-isotp*, zbog mogućnosti definiranja *callback* funkcija koje se pozivaju po primitku *CAN* poruka [5, 1]. Međutim, *Scapy* podržava parsiranje poruka većeg broja protokola, poput UDS-a i XCP-a, koji u navedenim bibliotekama nisu podržani. Uz to, *Scapy* u potpunosti iskorištava mogućnosti *SocketCANa*, poput podrške za transportni protokol ISO-TP u *Linux* jezgri. Navedeno svojstvo *Scapy* čini kompatibilnijim s ostalim *SocketCAN* alatima, u odnosu na biblioteku *can-isotp*, koja ISO-TP implementira programski u *Pythonu*, povrh jezgre.

Scapy nudi način korištenja kroz naredbeni redak, kroz vlastiti omotač oko *Python* ljuske. Primjerice, za stvaranje objekta razreda `CANSocket` u svrhu slanja i primanja poruka putem *SocketCAN* sučelja dovoljno je pokrenuti naredbe s linija 1 do 3, iz ispisa 5.3. Linijom 1 omogućava se korištenje *SocketCANa* izravno, umjesto putem biblioteke *python-can*, a linijom 2 učitava se modul `cansocket`. Linijom 3 stvoren je `CANSocket` objekt za slanje i primanje poruka putem virtualnog CAN sučelja `vcan0`. Parametrom `can_filters` definirano je filtriranje svih CAN paketa osim onih s arbitražnim identifikatorom `0x344`. Filtriranje se odvija na razini *SocketCAN socketa* te je interno konfigurirano kroz *setsockopt* pozive [36]. Slanje jedne CAN poruke i blokirajuće čitanje iduće poruke moguće je ostvariti kroz pozive metoda `send` i `recv` ili kroz jedan poziv metode `sr1`, prikazano na linijama 5 do 8.

```
conf.contribs['CANSocket'] = {'use-python-can': False}
load_contrib('cansocket')
socket = CANSocket(channel="vcan0", can_filters=[{'can_id': 0x344, '
    can_mask': 0x7FF}])

socket.sr1(CAN(identifier=0x345, data=b'\x01\x02\x03'))
# ili
socket.send(CAN(identifier=0x345, data=b'\x01\x02\x03'))
socket.recv()
```

Ispis 5.3: Korištenje *SocketCAN* sučelja iz ljuske alata *Scapy*

U praksi, metoda `sr1` korisnija je za protokole oblika *zahtjev-odgovor*, primjerice za protokol UDS. Razlog tomu je što Scapy rekurzivno provjerava `answers` metodu implementiranu na razredima pojedinačnih tipova paketa, od nižih prema višim slojevima OSI modela. Primjerice, u slučaju poslanog `UDS_DSC` paketa, odnosno zahtjeva za servisom `DiagnosticSessionControl`, *Scapy* će pozvati `answers` metodu svakog idućeg primljenog paketa. Metodi `answers` će predati poslani paket kao parametar, kako bi mogla usporediti sadržaje oba paketa i zaključiti odgovara li primljeni paket na poslani. U slučaju da je primiljen paket `UDS_DSCPR`, koji je odgovor na poslani `UDS_DSC`, pozvana će biti metoda `answers` paketa `UDS_DSCPR` koja provjerava je li tip dijagnostičke sesije isti u poslanom i primljenom paketu (ispis 5.4). Ukoliko je, *Scapy*, primljeni paket će biti vraćen iz poziva `sr1` metode. Potrebno je napomenuti da je poziv metode `UDS_DSCPR.answers`, posljednji u nizu rekurzivnih poziva te da joj je prethodio poziv `UDS.answers`.

```
class UDS_DSCPR(Packet):
    name = 'DiagnosticSessionControlPositiveResponse'
    fields_desc = [
        ByteEnumField('diagnosticSessionType', 0,
                      UDS_DSC.diagnosticSessionTypes),
        StrField('sessionParameterRecord', b"")
    ]

    def answers(self, other):
        return isinstance(other, UDS_DSC) and \
            other.diagnosticSessionType == self.
                diagnosticSessionType
```

Ispis 5.4: `UDS_DSCPR.answers`metoda

Scapy UDS paketi formiraju se korištenjem operatora „/“ za slaganje slojeva. U slučaju UDS-a, kao temeljni sloj koristi se razred `UDS` povrh kojeg se nadodaju specifični tipovi UDS zahtjeva ili odgovora. Sastavljanje i slanje pakta zahtjeva UDS `ReadMemoryByAddress` koji čita `0xFF` okteta početka memorije prikazano je linijama 2 i 3 u ispisu 5.5, a linijom 7 poslan je pozitivan odgovor na zahtjev iz druge ljuske. Primljeni paket je instanca temeljnog `UDS` razreda, a konkretnom UDS odgovoru moguće je pristupiti putem člana `payload`. Specifičnim poljima odgovora također je moguće pristupiti putem članova razreda. U slučaju servisa `ReadMemoryByAddress` vraćenom dijelu memorije pristupa se putem člana `dataRecord`.

```
1  # ========= Scapy ljuska A =========
2  >>> pkt = UDS()/UDS_RMBA(memoryAddressLen=4,memorySizeLen=1,
       memoryAddress4=0x0, memorySize1=0xFF)
3  >>> res = s.sr1(pkt)
4
5  # ========= Scapy ljuska B =========
6
7  >>> s.send(UDS()/UDS_RMBAPR(dataRecord=b"diplomski"))
8  10
9
10 # ========= Scapy ljuska A =========
11 >>> res.payload
12 <UDS_RMBAPR  dataRecord='diplomski' |>
13 >>> res.dataRecord
14 b'diplomski'
```

Ispis 5.5: Sastavljane *Scapy* paketa

5.3.2. Definiranje CAN, UDS i XCP aplikacijske logike

Svaka od datoteka `can_handler.py`, `uds_handler.py` i `xcp_handler.py`, sadrži odgovarajući razred (engl. *class*) koji korisnik može implementirati, ako želi definirati aplikacijsku logiku u kontekstu nekog od navedenih protokola.

Primjerice, za definiranje CAN aplikacijske logike, korisnik treba implementirati razred `CANHandlerImpl`. Razred `CANHandlerImpl` nasljeđuje apstraktni razred `CanHandler` koji sadrži neimplementiranu metodu `handle_msg`. Navedena metoda je *callback* metoda koja se poziva pri svakom primitku CAN poruke. Primljena CAN poruka prosljeđuje se metodi `handle_msg` kroz argument `msg`. Apstraktni razred `CanHandler` sadrži i referencu na `NativeCANSocket` instancu, kako bi

svaki razred koji ga nasljeđuje mogao odgovarati na ili slati CAN poruke. Primjer jednostavnog `CanHandlerImpl` razreda koji sve primljene CAN poruke ponovno šalje s arbitražnim identifikatorom uvećanim za 1, prikazan je ispisom 5.6. Opisana struktura razreda vrijedi i za datoteke `uds_handler.py` i `xcp_handler.py`, odnosno razrede `UDSHandlerImpl` i `XCPHandlerImpl`.

```
1  class CanHandlerImpl(CanHandler):
2      def __init__(self, ecu: ECUModelImpl):
3          super().__init__(ecu)
4
5      def handle_msg(self, msg: CAN):
6          print(msg.identifier)
7          msg.identifier += 1
8          self.socket.send(msg)
```

Ispis 5.6: `CanHandlerImpl` primjer

U slučaju implementacije razreda `UDSHandlerImpl`, preporučeni obrazac je korištenje *Python* `match-case` mehanizma za razlikovanje različitih vrsta UDS zahtjeva (ispis 5.7). Prikazani primjer odgovara pozitivnim odgovorom na svaki zahtjev za servisima `ECUReset` te `DiagnosticSessionControl`.

```
1  class UDSHandlerImpl(UDSHandler):
2      def __init__(self, ecu: ECUModel):
3          super().__init__(
4              ecu,
5          )
6
7      def handle_msg(self, msg: UDS):
8          match msg.payload:
9              case UDS_ER():
10                 self.isotp.send(UDS() / UDS_ERPR(resetType=msg.
                       resetType))
11             case UDS_DSC():
12                 self.isotp.send(
13                     UDS() / UDS_DSCPR(diagnosticSessionType=msg.
                           diagnosticSessionType)
14                 )
```

Ispis 5.7: `UDSHandlerImpl` primjer

Kako bi nepotrebni pozivi *callback* metoda bili izbjegnuti u `config.py` datoteci moguće je definirati filtriranje CAN poruka prema arbitražnom identifikatoru za metode `handle_msg` razreda `CANHandlerImpl` i `XCPHandlerImpl` (ispis 5.8). Parametar `can_mask` sadržava masku kojom se određuju bitovi identifikatora relevantni za filtriranje. Postavljanjem parametra na `0x7FF` ili `0b11111111` binarno, odabiru se svi bitovi 11-bitnog identifikatora. Uz navedeno, u `config.py` datoteci potrebno je definirati ISO-TP adresu UDS poslužitelja u obliku *Python* rječnika s vrijednostima `rx_id` i `tx_id`. Onemogućavanje protokola moguće je postavljanjem varijabli `CAN_FILTERS`, `XCP` i `UDS` na vrijednost `None`.

```
1
2  CAN_FILTERS = [
3      {"can_id": 0x456, "can_mask": 0x7FF},
4  ]
5
6  XCP = [
7      {"can_id": 0x701, "can_mask": 0x7FF},
8  ]
9
10 UDS = dict(rx_id=0x100, tx_id=0x101)
```

Ispis 5.8: Konfiguracija filtriranja i ISO-TP adrese

5.3.3. Definiranje stanja ECU-a

Datoteka `ecu_model.py` i razred `ECUModelImpl` koji definira, namijenjen je pohranjivanju stanja. Stoga svaki od navedenih razreda za definiranje aplikacijske logike protokola CAN, UDS i XCP, čuva i referencu na instancu razreda `ECUModelImpl`.

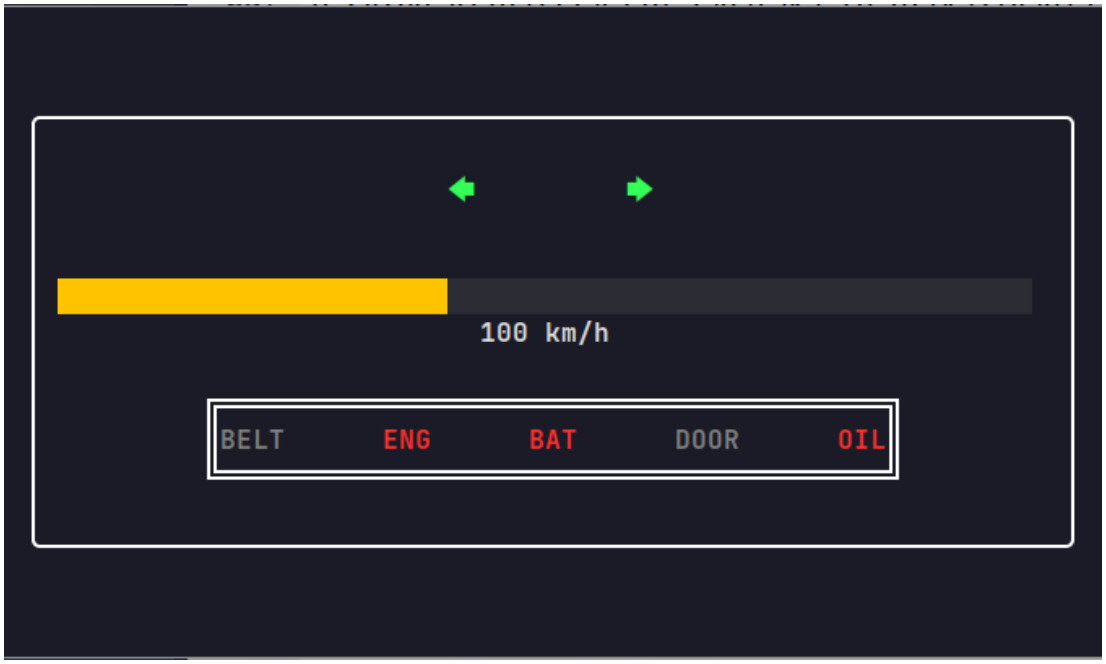
Korisniku je prepušteno koje će konkretno podatke o stanju čuvati u instanci razreda `ECUModelImpl`. Dodatno apstraktni razred `ECUModel` omogućava dodavanje i uklanjanje promatrača metodama `attach_listener` i `detach_listener` te njihovo obavještavanje metodom `notify`. Prikazan je primjer implementacije jednostavnog ECU-a, koji čita brzinu vozila sa sabirnice te ju šalje HTTP POST zahtjevom na nedefinirani API (ispis 5.9). Promatrače je potrebno postaviti u predefiniranoj funkciji `setup_ecu_model`, koju će izvršiti glavni program prilikom pokretanja.

```
class CanHandlerImpl(CanHandler):
    def __init__(self, ecu: ECUModelImpl):
        super().__init__(ecu)

    class SpeedData:
        def __init__(self, data: bytes):
            self.speed = int.from_bytes(data[1:3])

    def handle_msg(self, msg: CAN):
        if msg.data[0] == 0xAB and msg.length == 3:
            speed_data = SpeedData(data=msg.data)
            if 0 < speed_data.speed <= 250:
                self.ecu.set_speed(speed_data.speed)

class ECUModelImpl(ECUModel):
    class APIListener(Listener):
        def update(self, model: ECUModelImpl):
            data = {
                "speed": model.speed,
            }

            try:
                print("sending req")
                requests.post("http://api:8080/update", json=data)
            except Exception as e:
                print(e)
    def __init__(self):
        super().__init__()
        self.speed = 0
    def set_speed(self, speed: int):
        self.speed = speed
        self.notify()

def setup_ecu_model():
    model = ECUModelImpl()
    model.attach_listener(APIListener())
    return model
```

Ispis 5.9: Primjer implementacije `ECUModelImpl`

5.4. Vizualna komponenta zadataka

U svrhu vizualizacije zadataka te prikazivanja povratne informacije natjecatelju, izrađena je vizualna komponenta nazvana *ic-tui*. Za vizualnu komponentu odabrana je ploča s instrumentima te je implementirana kao korisničko sučelje naredbenog retka (engl. *terminal user interface*, TUI) (slika 5.2).



Slika 5.2: TUI ploče s instrumentima

5.4.1. Implementacija

Pri izradi vizualne komponente korištena je biblioteka *Wish* i razvojni okvir *Bubbletea* programskog jezika *Go* [18, 17]. Razvojni okvir *Bubbletea* koristi se za stvaranje TUI aplikacija te definira niz jednostavnih često korištenih TUI komponenti. Primjerice, za prikaz brzinomjera na ploči s instrumentima, korištena je komponenta `progress`.

U sklopu *Bubbletea* arhitekture, vizualna komponenta koristi *Go* dretve (engl. *gorutines*) za pokretanje jednostavnog HTTP servera koji očekuje POST zahtjev s stanjem ploče koje treba prikazati. Prikazan je primjer zahtjeva upućenog alatom *curl*, pod pretpostavkom da je komponenta pokrenuta na istom računalu (ispis 5.10). U slučaju da ECU program treba signalizirati da je ostvaren uvjet za pobjedu te da je potrebno prikazati zastavicu, može to učiniti korištenjem `winCondition` parametra, nakon

čega će se umjesto ploče s instrumentima prikazati zastavica.

```
$ curl -X POST -H "Content-Type: application/json" --data \
'{"winCondition":false, \
  "speed":100 , \
  "blinkers":true, \
  "seatbelt":false, \
  "engine":true, \
  "battery":true, \
  "doors":false, \
  "oil":true}' localhost:8080/update
```

Ispis 5.10: Primjer POST zahtjeva sa stanjem ploče s instrumentima

Udaljeni prikaz TUI-ja ostvaren je kroz biblioteku *Wish*, koja omogućava prikaz *Bubbletea* TUI aplikacija putem SSH-a te u tu svrhu implementira vlastiti SSH server. Implementirana vizualna komponenta podržava zastavice „-ssh“ i „-a“, a upute je moguće ispisati postavljanjem zastavice „-h“ (ispis 5.11). Zastavica „-ssh“ koristi se za pokretanje vizualne komponente u sklopu *Wish* servera, a zastavica „-a“ omogućava specificiranje adrese i pristupa na kojima će HTTP poslužitelj slušati.

```
$ ./ic-tui -h
Usage of ./ic-tui:
  -a string
        Address and port for HTTP API in format <ip_address>:<port>
            (default ":8080")
  -ssh
        Set to run as a Wish SSH server

$ ./ic-tui -a "0.0.0.0:4000" -ssh
```

Ispis 5.11: Korištenje vizualne komponente

5.5. Dodatak Docker mrežnom upravljačkom programu

Pri implementaciji sustava za stvaranje CTF zadataka, bilo je potrebno ostvariti pouzdan način umrežavanja ECU programa odnosno ECU kontejnera te su razmotrene sljedeće opcije:

- umetanje skripti za umrežavanje u svaki kontejner
- umrežavanje putem klasičnih *Docker* mreža te tuneliranje CAN prometa putem TCP-a
- zasebni program koji brine o umrežavanju i stanju kontejnera

Opcija umetanja skripti kroz *Dockerfile*, koje se pokreću na početku rada kontejnera te ga povezuju putem *SocketCAN* mrežnih sučelja i pravila usmjeravanja, pokazala se nepouzdanom. Nedostatak centralnog entiteta, dovodio je do problema u sinkronizaciji između kontejnera pri stvaranju mrežnih sučelja i pravila usmjeravanja. Uz navedeno, sam način umrežavanja nije pouzdan te ostavlja više prostora za korisničke pogreške.

Pouzdanija opcija bila je umrežavanje kontejnera putem klasičnih *Docker* mreža te tuneliranje CAN prometa između kontejnera putem TCP-a. Međutim, ova opcija zahtijeva pokretanje dodatnog programa za tuneliranje u svakom kontejneru, što kao i prethodna opcija ostavlja više prostora za korisničke pogreške. Uz navedeno, potrebno bi bilo osmisliti proces kojim će kontejneri međusobno iskomunicirati podatke poput TCP pristupa na kojima će tuneli biti postavljeni. Za više sabirnica ovaj postupak postaje višestruko kompleksniji.

Korištenje zasebnog programa koji brine o umrežavanju i stanju kontejnera najpouzdanija je opcija od prethodno navedenih. Međutim, navedeni program bi morao voditi računa i o uklanjanju stvorenih mrežnih sučelja i ostalih artefakata. Dodatno, istu funkciju već obavlja *Docker Engine* te su naposljetku istražene opcije za njegovo proširenje.

Docker Engine moguće je proširiti pisanjem i korištenjem mrežnih dodataka, dodataka za bilježenje dnevničkih zapisa te volumnih dodataka. Za potrebe omogućavanja novih vrsta umrežavanja, koriste se mrežni dodatci. Stoga je ova opcija odabrana kao pouzdan način omogućavanja umrežavanja ECU kontejnera putem protokola CAN.

5.5.1. Linux mrežni imenski prostori

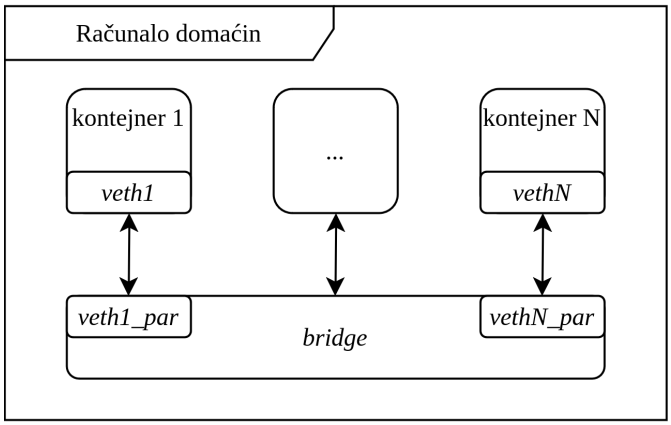
Linux mrežni imenski prostori (engl. *network namespaces*), pružaju mogućnost izolacije mrežnih resursa operacijskog sustava poput mrežnih sučelja i uređaja, mrežnih stogova, tablica usmjeravanja i sigurnosnih stijena [7]. Mrežni imenski prostor služi kao potpuno izolirano mrežno okruženje za procese koji mu pripadaju. Dodavanje dvaju mrežnih imenskih prostora korištenjem alata *ip*, prikazano je ispisom 5.12.

```
1 $ ip netns add ns1
2 $ ip netns add ns2
3
4 $ ip netns
5 ns1
6 ns2
```

Ispis 5.12: Dodavanje mrežnih imenskih prostora

Navedena svojstva čine mrežne imenske prostore pogodnima za kontejnerizaciju aplikacija te je takav pristup mrežnoj izolaciji koristi *Docker* [57]. Konkretnije, pri pokretanju svakog novog kontejnera, *Docker* mu dodjeljuje novi mrežni imenski prostor. Ovim postupkom, izbjegavaju se potencijalne kolizije s drugim aplikacijama pri zauzimanju TCP i UDP pristupa te pri korištenju ostalih mrežnih resursa.

S obzirom na to da su kontejneri mrežno izolirani, za omogućavanje njihove međusobne komunikacije *Docker* ih automatski dodaje u pretpostavljenu *bridge* mrežu. Kako bi *bridge* mreža ispravno funkcionirala, *Docker* koristi dvije vrste mrežnih sučelja: *bridge* i *veth*. *Linux* mrežno sučelje *bridge* ponaša se kao virtualni preklopnik, prosljeđujući pakete između sučelja spojenih na njega. Mrežno sučelje *veth* je lokalni *Ethernet* tunel te služi za međusobno povezivanje mrežnih sučelja. Krajevi tunela su *veth* parovi, gdje se svaki nalazi u zasebnom mrežnom imenskom prostoru. Korištenje samo sučelja *veth* za međusobno povezivanje imenskih prostora svih kontejnera, rezultiralo bi u $\binom{n}{2}$ *veth* tunela, gdje je n broj kontejnera. Stoga *Docker* koristi sučelje *bridge* na računalu domaćinu, za usmjeravanje paketa između *veth* tunela, što smanjuje broj tunela na n (slika 5.3)



Slika 5.3: *Docker* bridge mreža

5.5.2. Umrežavanje kontejnera korištenjem paketa SocketCAN

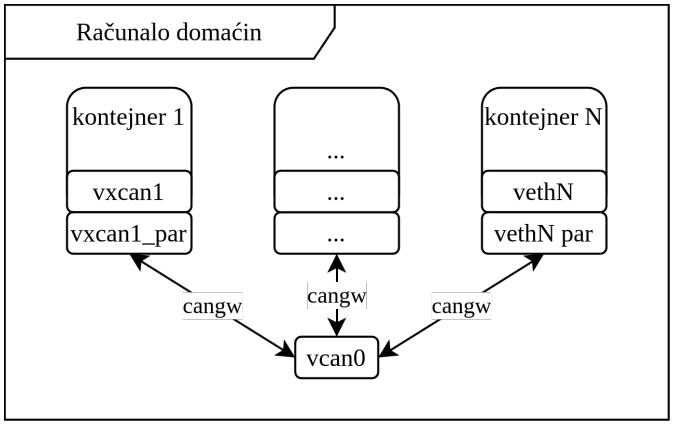
SocketCAN ekvivalent *veth* tunelima su *vxcan* tuneli, koja omogućavaju povezivanje *Linux* mrežnih imenskih prostora putem protokola CAN, a time i povezivanje *Docker* kontejnera [47]. Za prosljeđivanje CAN poruka između *vcan* i *vxcan* sučelja, koristi se jezgreni modul *can_gw* i pripadajući alat *cangw*. Povezivanje dvaju imenskih prostora `ns1` i `ns2` *vxcan* tunelom, prikazano je ispisom 5.13. Povezivanjem mrežnih imenskih prostora n kontejnera na ovaj način rezultira u $\binom{n}{2}$ *vxcan* tunela, kao i u slučaju *veth* tunela.

```
1 $ ip link add vxcan1 type vxcan peer name vxcan1_par
2
3 $ ip link set vxcan1 netns ns1
4 $ ip link set vxcan1_par netns ns2
```

Ispis 5.13: Povezivanje mrežnih imenskih prostora *vxcan* tunelom

SocketCAN ekvivalent sučelju *bridge* ne postoji te izravna preslika optimizacija iz slučaja *veth* tunela nije moguća. Međutim, zbog *loopback* svojstva virtualnih CAN sučelja *vcan*, moguće ih je iskoristiti kao zamjenu za sučelje *bridge*. Sučelje *vcan* nije izvorno namijenjeno za ovu primjenu te nije moguće dodijeliti mu sučelja *vxcan*, kao što je to moguće u slučaju sučelja *bridge* i *veth*. Opisano ograničenje moguće je zaobići korištenjem alata *cangw* za dodavanje pravila usmjeravanja CAN poruka između *vcan* i *vxcan* sučelja (slika 5.4). Naredbe za ručno umrežavanje imenskih prostora putem naredbenog retka prikazane su ispisom 5.14. Naredbama na linijama 1 do 3 dodana

su potrebna *vcan* i *vxcan* sučelja, a naredbama na linijama 5 i 6 prebačeni su krajevi oba tunela u odgovarajuće mrežne imenske prostore. Naredbom *modprobe* na liniji 17 učitan je modul jezgre *can_gw*, što omogućava korištenje alata *cangw* za povezivanje sučelja *vxcan* i *vcan*, prikazano na linijama 19 do 22. Potrebno je napomenuti da je modul *can_gw* učitan s paramterom `max_hops` postavljenim na 2, čime se dopušta da CAN poruke budu proslijeđene više od jednog puta. Navedeni parametar je potreban jer je broj skokova od sučelja *vxcan* jednog kontejnera do sučelja *vxcan* drugog kontejnera uvijek jednak 2.



Slika 5.4: Umrežavanje kontejnera korištenjem alata *cangw*

```
$ ip link add vxcan1 type vxcan peer name vxcan1_par
$ ip link add vxcan2 type vxcan peer name vxcan2_par
$ ip link add vcan0 type vcan

$ ip link set vxcan1 netns ns1
$ ip link set vxcan2 netns ns2

$ ip a
(...)
9: vxcan1_par@if10: <NOARP> mtu 72 qdisc noop state DOWN group
   default qlen 1000
    link/can  link-netns ns1
11: vxcan2_par@if12: <NOARP> mtu 72 qdisc noop state DOWN group
   default qlen 1000
    link/can  link-netns ns2
13: vcan0: <NOARP> mtu 72 qdisc noop state DOWN group default qlen
   1000
    link/can

$ modprobe can_gw max_hops=2

$ cangw -A -s vcan0 -d vxcan1_par -e
$ cangw -A -s vcan0 -d vxcan2_par -e
$ cangw -A -d vcan0 -s vxcan1_par -e
$ cangw -A -d vcan0 -s vxcan2_par -e
$ cangw -L
cangw -A -s vcan0 -d vxcan2_par -e # 0 handled 0 dropped 0 deleted
cangw -A -s vcan0 -d vxcan1_par -e # 0 handled 0 dropped 0 deleted
```

Ispis 5.14: Povezivanje mrežnih imenskih prostora alatom *cangw*

5.5.3. LibNetwork i udaljeni upravljački programi

Docker Engine omogućava pisanje mrežnih programskih dodataka, kojima je moguće definirati dodatne načine umrežavanja kontejnera. *Docker* podržava nekoliko ugrađenih načina umrežavanja kontejnera, primjerice *Docker bridge*. Ugrađeni načini umrežavanja odnosno njihovi upravljački programi za upravljanje umrežavanjem kontejnera izravno koriste biblioteku *LibNetwork* [51].

Pri pisanju mrežnih programskih dodataka, biblioteku nije moguće koristiti izravno, već pisanjem HTTP poslužitelja koji implementira *LibNetwork* protokol za udaljene

upravljačke programe (engl. *remote driver protocol*). Smatra se da je HTTP poslužitelj implementirao navedeni protokol, ako prihvaća određeni skup HTTP POST zahtjeva s argumentima predanima u JSON formatu. Konkretnije, za definiranje umrežavanja kontejnera bitni su POST zahtjevi na putanje:

- `/NetworkDriver.CreateNetwork`
- `/NetworkDriver.DeleteNetwork`
- `/NetworkDriver.CreateEndpoint`
- `/NetworkDriver.DeleteEndpoint`
- `/NetworkDriver.Join`
- `/NetworkDriver.Leave`

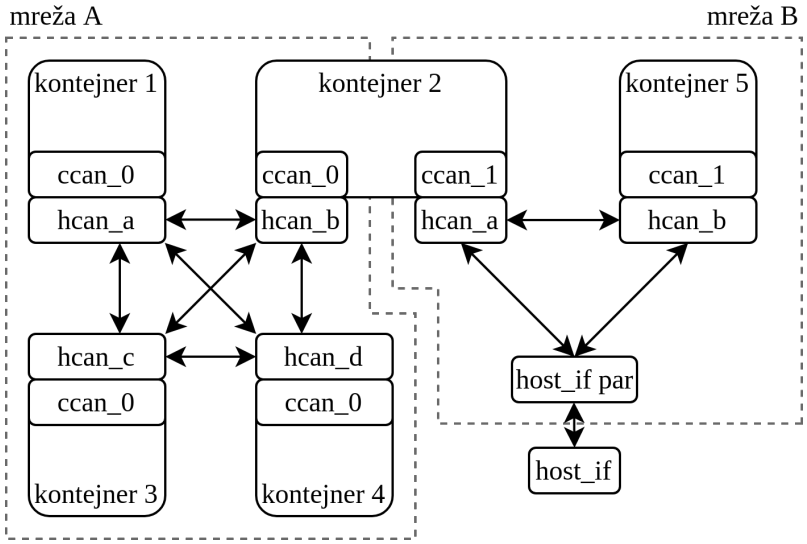
Navedene zahtjeve šalje *Docker Engine* na HTTP poslužitelj dodatka. Zahtjeve `/NetworkDriver.CreateNetwork` i `/NetworkDriver.DeleteNetwork` šalje prilikom stvaranja i brisanja mreža kontejnera, zahtjeve `/NetworkDriver.CreateEndpoint` i `/NetworkDriver.DeleteEndpoint` šalje prije dodavanja i uklanjanja kontejnera iz mreže. Zahtjeve `/NetworkDriver.Join` i `/NetworkDriver.Leave` šalje prilikom dodavanja i uklanjanja kontejnera iz mreže.

5.5.4. Implementacija

Dodatak, nazvan *dockercan*, implementiran je u programskom jeziku *Go*, korištenjem službenog *Docker* pomoćnog koda za pisanje dodataka [26]. Dodatno, ovisi o alatima *cangw* i *ip* te modulima jezgre *can_gw* i *vxcan* za ispravan rad. Dodatak podržava dva načina umrežavanja: centralizirani i *peer-to-peer*. Neovisno o odabranom načinu umrežavanja, dodatak stvara zasebni mrežni imenski prostor za svaku mrežu, koji koristi za skrivanje *cangw* pravila i stvorenih sučelja od korisnika ili natjecatelja. Pri opisivanju oba načina umrežavanja koristi se isti primjer s dvije *dockercan* mreže: mreža A sadrži 5 kontejnera, a mreža B sadrži 2 kontejnera te ima uključenu opciju stvaranja sučelja za pristup mreži na računalu domaćina.

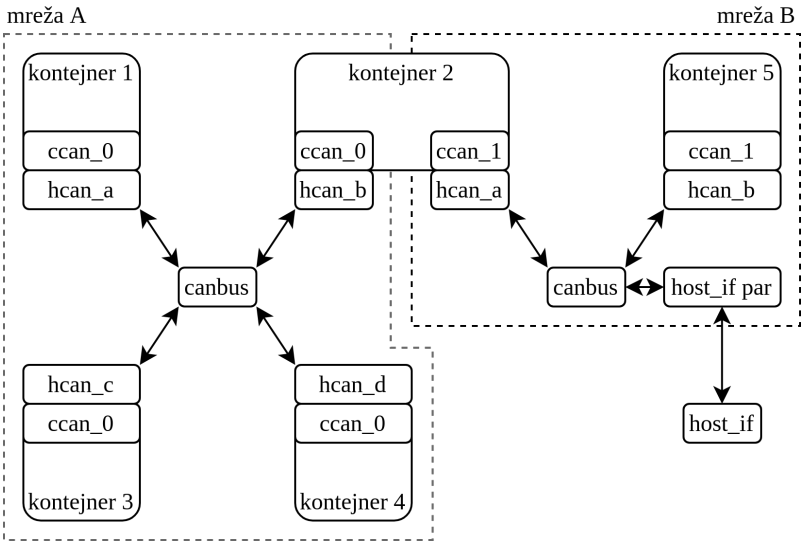
Peer-to-peer način umrežavanja je jednostavniji za implementaciju, ali koristi $2 * \binom{n}{2}$ *cangw* pravila za povezivanje kontejnera. Prototip *Docker* dodatka s ovim načinom umrežavanja opisao je Gagneraud u [32]. U ovom načinu umrežavanja, dodatak svakom kontejneru dodjeljuje jedan *vxcan* tunel, gdje je jedan kraj tunela u mrežnom imenskom prostoru kontejnera, a drugi u mrežnom imenskom prostoru njegove mreže (slika 5.5). Krajevi *vxcan* tunela koji se nalaze u mrežnim prostorima kontejnera, formata su `ccan_X`, gdje je `ccan_` prefiks, a `X` neki cijeli broj. *LibNetwork* vodi

računa o jedinstvenosti naziva krajeva tunela u istom kontejneru. Primjerice, prema primjeru sa slike 5.5, kada je kontejner 2 bio dodan u mrežu A, stvoreno je *vxcan* sučelje `ccan_0` u njegovom mrežnom imenskom prostoru. Kada je naknadno dodan u mrežu B, *LibNetwork* je stvoreno *vxcan* sučelje nazvao `ccan_1`, kako bi se izbjegla kolizija u imenima sučelja. Krajevi *vxcan* tunela u mrežnom imenskom prostoru mreža A i B imenovani su prema formatu `hcan_ID`, gdje je `hcan_` prefiks, a `ID` identifikator krajnje točke (engl. *endpoint*) kontejnera. Krajevi *vxcan* tunela u mrežnom imenskom prostoru mreže, međusobno su povezani *cangw* pravilima. U slučaju mreže B, odabrana je opcija stvaranja sučelja za pristup mreži na računalu domaćina. Navedena opcija stvara dodatni *vxcan* tunel, čiji je jedan kraj u imenskom prostoru mreže, a drugi kraj u imenskom prostoru računala domaćina. Kraj tunela u imenskom prostoru mreže, povezuje se s *vxcan* tunelima svih kontejnera.



Slika 5.5: *Peer-to-peer* umrežavanje

Centralizirani način umrežavanja koristi optimizaciju opisanu u potpoglavlju 5.5.2 te za umrežavanje n kontejnera koristi $2n$ *cangw* pravila. Kao i u *peer-to-peer* načinu umrežavanja, dodatak za svaki kontejner stvara *vxcan* tunel, gdje jedan kraj tunela postavlja u mrežni imenski prostor mreže, a drugi kraj u mrežni imenski prostor kontejnera. Glavna razlika između dvaju načina povezivanja je što centralizirani način stvara jedno *vcan* sučelje, nazvano *canbus*, u mrežnom imenskom prostoru svake mreže te na njega povezuje krajeve *vxcan* tunela svakog kontejnera, kao i tunela prema imenskom prostoru računala domaćina (slika 5.6). Na centralno *vcan* sučelje jednostavno je spojiti i *SocketCAN* sučelje prema CAN sklopovlju, čime se omogućava povezivanje virtualnih i fizičkih CAN sabirnica.



Slika 5.6: Centralizirano umrežavanje

5.5.5. Korištenje

Dodatak je moguće učitati na tri načina:

- pokretanjem kroz naredbeni redak
- kao *Systemd* servis
- instalacijom kroz sustav *Docker* dodataka

Pokretanjem kroz naredbeni redak, dodatak radi kao HTTP poslužitelj na TCP pristupu 4343. U slučaju učitavanja dodatka kao *Systemd* servisa s konfiguracijskom datotekom prikazanom u ispisu 5.15, dodatak radi kao HTTP poslužitelj na TCP pristupu 5555. Naposljetku, dodatak se može izgraditi za instalaciju kroz sustav *Docker* dodataka, pri čemu se pokreće u posebnom načinu u kojem se za komunikaciju s *Docker Engine* procesom ne koristi HTTP poslužitelj već *Unix* domenski priključak (engl. *Unix domain socket*). U sva tri navedena slučaja, službeni pomoćni kod za pisanje *Docker* dodataka automatski stvara potrebne specifikacijske datoteke u direktorijima `/etc/docker/plugins` i `/usr/lib/docker/plugins`.

Pri stvaranju *dockercan* mreža korištenjem *Docker* sučelja za naredbeni redak (engl. *command line interface*, CLI) ili putem `compose.yml` datoteka moguće je specificirati tri opcije. Opcija `centralised` određuje koji način umrežavanja će dodatak koristiti, opcija `canfd` će omogućiti slanje CAN-FD poruka kroz stvorenu mrežu, a opcija `host_if` omogućava specificiranje imena sučelja za pristup mreži koje će biti stvoreno na računalu domaćina. Primjeri korištenja dodatka putem *Docker* CLI-a i `compose.yml` datoteke prikazani su u ispisima 5.16 i 5.17.

```
[Unit]
Description=Dockercan network plugin
Before=docker.service
After=network.target
Requires=docker.service

[Service]
User=root
ExecStart=/usr/lib/docker/dockercan_remote -addr 127.0.0.1:5555

[Install]
WantedBy=multi-user.target
```

Ispis 5.15: Konfiguracijska datoteka Systemd servisa

```
$ docker network create -o centralised=true -o canfd=true -o host_if
   =dcan1 --driver dockercan sabirnica1
```

Ispis 5.16: Korištenje dodatka kroz *Docker* CLI

```
services:
  ECM:
    image: alpine
    networks: [ pogonskisklop ]

  TCU:
    image: alpine
    networks: [ pogonskisklop ]

networks:
  pogonskisklop:
    driver: dockercan
    driver_opts:
      centralised: "false"
      canfd:       "true"
      host_if:     "dcan1"
```

Ispis 5.17: Primjer `compose.yml` datoteke

5.6. Generatorska skripta

U sklopu sustava za izradu CTF zadataka, napisana je jednostavna skripta za generiranje početnog stabla direktorija i `compose.yml` datoteke za pokretanje zadatka. Pokretanje i početni izbornik skripte prikazan je ispisom 5.18.

```
$ mkdir out
$ python3 generate.py out


   ___  _____  ________ ______   ___
  / _ \/ __ \/ ___/ //_/ _ | / _ \
 / // / /_/ / /__/ ,< / __ |/ , _/
/____/\____/\___/_/|_/_/ |_/_/|_|


    Select option:

    1) Add CAN bus
    2) Add ECU
    3) Done
    4) Exit

Enter choice [1-4]:
```

Ispis 5.18: Glavni izbornik generatorske skripte

5.6.1. Primjer generiranja projekta

Za potrebe primjera, cilj je generirati projekt s dvije CAN sabirnice, za ECU-ove domena pogonskog sklopa i kabine. Na sabirnicu domene kabine potrebno je spojiti ECU ploče s instrumentima, koji se temelji na predlošku programa ECU-a. Uz to, potrebno mu je dodijeliti vizualnu komponentu, odnosno povezati ga s TUI-jem ploče s instrumentima. Sabirnica domene kabine mora biti dostupna udaljeno putem *cannellonni* tunela te lokalno putem *SocketCAN* sučelja na računalu domaćinu. Na sabirnicu domene pogonskog sklopa potrebno je povezati ECU motora, koji se temelji na predlošku programa ECU-a. Sabirnica domene pogonskog sklopa ne smije biti dostupna lokalno niti udaljeno, već samo putem poveznika koji spaja obje sabirnice. Na obje sabirnice potrebno je generirati šum.

Prilikom dodavanja sabirnice nude se opcije: korištenja protokola CAN-FD umjesto klasičnog CAN-a, dodavanja *cannelloni* kontejnera za udaljeno povezivanje, dodava-

nja lokalnog *SocketCAN* sučelja te generiranja šuma na sabirnici. Dodavanje sabirnice domene kabine prikazano je ispisom 5.19.

```
Enter choice [1-4]: 1
Enter bus name:  kabina
[dockercan] Connect host to this bus over vcan interface?[y/n]: y
Enter interface name: ctf_ulaz
[dockercan] Use CAN FD?[y/n]: n
Add random CAN frame generation to this bus?[y/n]: y
Attach cannelloni container?[y/n]: y
```

Ispis 5.19: Dodavanje sabirnice

Prilikom dodavanja ECU-a nude se opcije korištenja predloška ECU programa te povezivanja na TUI ploče s instrumentima. Naposljetku je potrebno odabrati sabirnice na koje će ECU biti povezan. Ispisom 5.20 prikazano je dodavanje ECU-a ploče s instrumentima kojem treba dodijeliti TUI ploče s instrumentima te dodavanje poveznika kojeg treba povezati s obje sabirnice.

```
Enter choice [1-4]: 2
Enter ECU name: poveznik
Use ECU template?[y/n]: n
Connect ECU to IC-TUI?[y/n]: n
Connect ECU to which buses? (e.g. 1,3)
 1) pogon
 2) kabina

Enter choice:  1,2

(...)

Enter choice [1-4]: 2
Enter ECU name: ic-ecu
Use ECU template?[y/n]: y
Connect ECU to IC-TUI?[y/n]: y
Connect ECU to which buses? (e.g. 1,3)
 1) pogon
 2) kabina

Enter choice:  2
```

Ispis 5.20: Dodavanje ECU-a

Nakon svakog dodanog elementa ispisuje se izgled trenutne arhitekture, a odabirom treće opcije skripta generira stablo direktorija i početni kod zadatka (ispis 5.21). U `compose.yml` datoteci kontejner ECU ploče s instrumentima i TUI-ja stavljeni su u zasebnu *Docker bridge* mrežu, kako bi se omogućila komunikacija putem TUI-jevog HTTP API-ja. Uz navedeno, *cannelloni* kontejneru je otvoren TCP pristup 20000, odnosno pretpostavljeni pristup za *cannelloni* TCP poslužitelj. Za svaki idući dodani *cannelloni* kontejner, pristup se inkrementira kako bi se izbjegla kolizija. Za ECU-ove za koje je odabrana opcija korištenja predložaka, generiran je početni kod u odgovarajućim direktorijima te su u `compose.yml` datoteci definiranje putanje izgradnje kontejnera.

```
 ========= NETWORK LAYOUT =========

pogon           kabina
│               │
├ noise_gen     ├ noise_gen
│               │
├ ecm           ├ cannelloni
│               │
├ poveznik      ├ poveznik
│               │
│               ├ ic-ecu

 ========= NETWORK LAYOUT =========

    Select option:

    1) Add CAN bus
    2) Add ECU
    3) Done
    4) Exit

Enter choice [1-4]: 3

$ ls out
cangen  cannelloni  compose.yml  ecm  ic-ecu  ic-tui0
```

Ispis 5.21: Završetak postupka generacije projekta

5.6.2. Mogućnosti pristupa za natjecatelje

U kontekstu generiranog projekta iz prethodnog poglavlja, postoji nekoliko načina omogućavanja udaljenog i lokalnog pristupa. Ukoliko natjecatelj ima lokalni pristup zadatku, potrebno mu je ograničiti mogućnosti stvaranjem korisnika s niskom razinom privilegija, kako bi interakcija bila ograničena na korištenje dopuštenih alata na *SocketCAN* sučelju računala domaćina. Uz to, lokalni prikaz TUI-ja, moguće je ostvariti korištenjem naredbe `docker attach` na TUI kontejneru, u slučaju da je TUI pokrenut bez zastavice „-ssh“. Suprotno, u slučaju da je TUI pokrenut u SSH načinu rada, potrebno je korisniku dozvoliti spajanje na lokalni SSH server TUI kontejnera.

Ako natjecatelj pristupa s udaljenog računala, potrebno mu je pružiti upute i podatke za spajanje na *cannelloni* TCP server, kako bi imao pristup nekoj od CAN sabirnica. Za udaljeni prikaz TUI-ja, dovoljno ga je pokrenuti u SSH načinu rada te natjecatelju dati podatke za spajanje. Alternativa ovom načinu udaljenog pristupa je primjenjivanje svih prethodno navedenih mjera za lokalni pristup te omogućavanje SSH pristupa putem korisnika s niskom razinom privilegija.

5.7. Zadaci

U sklopu ovog rada izrađena su tri zadatka koji demonstriraju ranjivosti i ranjive implementacije protokola CAN i UDS. Konkretnije, zadaci demonstriraju iskorištavanje UDS servisa `ReadMemoryByAddress`, iskorištavanje ranjive implementacije UDS `SecurityAccess` servisa te napad lažiranjem putem protokola CAN. Zadaci su namijenjeni za rješavanje korištenjem alata *caringcaribou*, *Scapy* i *can-utils* paketa [2, 6]. U nastavku je opisan tijek rješavanja zadataka.

5.7.1. Iskorištavanje UDS servisa ReadMemoryByAddress

Natjecatelju je na početku zadatka dana informacija da ECU koristi procesor s 32-bitnim memorijskim adresiranjem. Natjecatelj dobiva pristup CAN sabirnici na koju je povezan ECU s UDS poslužiteljem. Korištenjem alata *caringcaribou* odnosno njegovog modula za otkrivanje UDS poslužitelja, natjecatelj saznaje ISO-TP adresu poslužitelja. Potom natjecatelj ponovo koristi alat *caringcaribou* za skeniranje dostupnih UDS servisa te pronalazi servis `ReadMemoryByAddress`. Korištenjem informacije o duljini memorijskih adresa, napadač mora sastaviti skriptu kojom će pročitati cjelokupnu memoriju ECU-a. Skriptu je moguće napisati korištenjem biblioteke *Scapy*,

can-utils alata ili izravnim korištenjem *Berkley socket* API-ja otvaranjem *socketa* nad *SocketCAN* sučeljem (ispis 5.22). Nakon što je pročitao memoriju ECU-a, zastavicu može pronaći traženjem niza znakova koji odgovara danom formatu zastavice u pročitanim podacima.

```
memorySizeLen = 1
memoryAddressLen = 4
data = b""
addr = 0x0
size = 0xFF

while True:
    pckt = sock.sr1(UDS() / UDS_RMBA(memorySizeLen=memorySizeLen,
                                     memoryAddressLen=
                                         memoryAddressLen,
                                     memorySize1=size,
                                     memoryAddress4=addr),
                    verbose=False)
    if isinstance(pckt.payload, UDS_NR):
        break
    if isinstance(pckt.payload, UDS_RMBAPR):
        data += pckt.dataRecord
    print(f"Addr: {addr}", end="\r")
    addr += size

with open("binary", "wb") as f:
    f.write(data)
```

Ispis 5.22: *Scapy* skripta za čitanje cjelokupne memorije ECU-a

5.7.2. Iskorištavanje ranjive implementacije UDS SecurityAccess servisa

Kao i u prethodnom zadatku, natjecatelj mora iskoristiti alat *caringcaribou* u svrhu otkrivanja ISO-TP adrese UDS poslužitelja te dostupnih UDS servisa, odnosno servisa `SecurityAccess`, `ReadMemoryByAddress`, `ReadDataByIdentifier`.Kroz pokušaje komunikacije sa servisima, natjecatelj mora zaključiti da je servis `ReadMemoryByAddress` zaštićen servisom `SecurityAccess`, ali da `ReadDataByIdentifier` servis nije. Korištenjem funkcije `dump_dids` alata *caringcaribou*, natjecatelj može iščitati vrijednosti iz identifikatora 9 i 21 u heksadekadskom formatu

(ispis 5.23). Pretvaranjem navedenih vrijednosti u ASCII znakove, natjecatelj otkriva broj šasije vozila odnosno prvu zastavicu te niz znakova „SHA-512“. Natjecatelj potom treba zaključiti da `SecurityAccess` neispravno koristi algoritam SHA-512 kao *seed-and-key* algoritam.

```
$ caringcaribou uds dump_dids --max_did 0x25 0x100 0x101

(...)

Identified DIDs:
DID    Value (hex)
0x0009 3356574658374154324444d363034343934
0x0021 5348412d353132
```

Ispis 5.23: Dodavanje ECU-a

Naposljetku, kako bi ostvario pristup servisu `ReadMemoryByAddress`, natjecatelj treba zatražiti *seed* vrijednost slanjem `SecurityAccess` zahtjeva te izračunati ključ odnosno SHA-512 sažetak dobivene *seed* vrijednosti. Nakon autorizacije, natjecatelj treba isčitati sadržaj memorije ECU-a te u njoj pronaći zastavicu, kao u prethodnom zadatku.

5.7.3. Napad lažiranjem putem protokola CAN

Posljednji zadatak sastoji se od poveznika i ECU-a ploče s instrumentima te pripadajućeg TUI-ja. Poveznik podržava UDS servis `RoutineControl`, a pokretanjem bilo koje od rutina, poveznik počinje slati testni niz CAN poruka. Testni niz poruka redom pali statusne LED diode na ploči s instrumetima te inkrementira prikazanu brzinu od 0 km/h do 170 km/h. U svrhu otežavanja zadatka, na sabirnici se konstantno generiraju nasumične CAN poruke. Uvjet za prikaz zastavice na TUI-ju je postići brzinu preko 230km/h i upaliti pokazivače smjera.

Natjecatelj treba iskoristiti mogućnost višestrukog pokretanja rutine kako bi reverznim inženjeringom CAN prometa shvatio koje okvire treba modificirati i ponovno poslati, kako bi ispunio uvjet za prikaz zastavice.

6. Zaključak

Implementirani sustav pruža robusan način za stvaranje CTF zadataka u području sigurnosti automobila. *Docker* dodatkom omogućeno je pouzdano i prilagodljivo umrežavanje kontejnera putem virtualnih CAN sučelja i tunela. Predloškom programa ECU-a olakšana je implementacija CTF zadataka, pružanjem prilagodljive programske arhitekture za implementaciju aplikacijske logike u kontekstu protokola CAN, UDS i XCP. Uz to, demonstrirana je mogućnost povezivanja predložaka s dodatnim programskim proširenjima putem *Docker bridge* mreža. Izrađen je primjer programskog proširenja u obliku vizualne komponente, odnosno TUI-ja ploče s instrumentima. Sustav je objedinjen generatorskom skriptom koja omogućava sastavljanje arhitekture CTF zadatka, kao i generiranje početnog koda te *Docker Compose* konfiguracijske datoteke za jednostavnije postavljanje zadatka. Definirani su načini omogućavanja udaljenog i lokalnog pristupa zadacima. Napravljen je pregled E/E arhitekture modernih automobila kao i pregled postojećih istraživanja sigurnosti u posljednjem i ovom desetljeću. Analizirani su postojeći edukativni materijali u području kibernetičke sigurnosti automobila te su izrađena su tri CTF zadatka koji demonstriraju ranjivosti protokola CAN i UDS, ali i služe kao smjernice za izradu novih CTF zadataka korištenjem implementiranog sustava. Glavni nedostatak implementiranog sustava je nemogućnost simuliranja napada na fizičkom sloju. S obzirom na to da je sustav implementiran prvenstveno programski, nije moguće simulirati napade poput *bus-off* napada koji iskorištava interne brojače grešaka CAN primopredajnika, kao ni DoS napade koji iskorištavaju mehanizam arbitraže sabirnice. S druge strane, implementacija *Docker* dodatka za umrežavanje korištenjem *SocketCAN* paketa, omogućava povezivanje CAN sklopovlja na virtualnu mrežu, čime je moguće zaobići neke od navedenih nedostataka.

U daljnjem radu treba formalizirati načine povezivanja sklopovlja u virtualnu mrežu, nadogradnjom generatorske skripte ili izradom aplikacije koja će dodatno objediniti sve implementirane komponente sustava. Potrebno je i izraditi dodatne konfigurabilne ECU kontejnere, poput kontejnera poveznika koji bi omogućio povezivanje više sabir-

nica i filtriranje CAN poruka između njih. U predlošku programa ECU-a, potrebno je unaprijediti razrede za implementaciju XCP i UDS logike, dodavanjem implementabilnih sučelja za formalnu implementaciju određenih servisa. Naposljetku, iako je implementirani sustav dostatan za simuliranje aplikacijske logike klasičnih ECU-ova kao i ranjivosti protokola UDS, XCP i CAN, potrebno je u daljnjim iteracijama staviti naglasak i na simuliranje napada na sustave zabave.

LITERATURA

[1] Biblioteka can-isotp. URL https://can-isotp.readthedocs.io/en/latest/. Zadnje pristupljeno 24. lipnja 2024.

[2] Alat caringcaribou. URL https://github.com/CaringCaribou/caringcaribou. Zadnje pristupljeno 24. lipnja 2024.

[3] Cloudcar. URL https://cloudcar.canbushack.com/. Zadnje pristupljeno 21. lipnja 2024.

[4] Mercedes-benz s-class w140. URL https://web.archive.org/web/20190610012852/https://www.mercedes-benz.com/en/mercedes-benz/classic/history/mercedes-benz-s-class-w-140/. Zadnje pristupljeno 14. lipnja 2024.

[5] Biblioteka python-can. URL https://python-can.readthedocs.io/en/stable/. Zadnje pristupljeno 24. lipnja 2024.

[6] Biblioteka scapy. URL https://scapy.readthedocs.io/en/latest/. Zadnje pristupljeno 24. lipnja 2024.

[7] network_namespaces(7) — linux manual page, 2024. URL https://man7.org/linux/man-pages/man7/network_namespaces.7.html. Zadnje pristupljeno 25. lipnja 2024.

[8] 00one. Defcon30 car hacking village ctf memorandum, 2022. URL https://www.00one.jp/blog/information/defcon30-chvctf-eng/. Zadnje pristupljeno 21. lipnja 2024.

[9] Emad Aliwa, Omer Rana, Charith Perera, i Peter Burnap. Cyberattacks and countermeasures for in-vehicle networks. *ACM computing surveys (CSUR)*, 54(1): 1–37, 2021.

[10] Android. Android auto, 2024. URL https://www.android.com/auto/. Zadnje pristupljeno 11. lipnja 2024.

[11] Apple. Apple carplay, 2024. URL https://www.apple.com/ios/carplay/. Zadnje pristupljeno 11. lipnja 2024.

[12] AUTOSAR. Specification of secure onboard communication protocol, 2020. URL https://www.autosar.org/fileadmin/standards/R20-11/FO/AUTOSAR_PRS_SecOcProtocol.pdf. Zadnje pristupljeno 20. lipnja 2024.

[13] Blockharbour. Proving grounds, . URL https://proving-grounds.blockharbor.io/. Zadnje pristupljeno 21. lipnja 2024.

[14] Blockharbour. Vsec, . URL https://vsec.blockharbor.io/. Zadnje pristupljeno 21. lipnja 2024.

[15] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, i Ian Jennions. Evaluation of can bus security challenges. *Sensors*, 20(8):2364, 2020.

[16] Zhiqiang Cai, Aohui Wang, Wenkai Zhang, Michael Gruffke, i Hendrick Schweppe. 0-days & mitigations: roadways to exploit and secure connected bmw cars. *Black Hat USA*, 2019(39):6, 2019.

[17] Charmbracelet. Razvojni okvir bubbletea, . URL https://github.com/charmbracelet/bubbletea. Zadnje pristupljeno 24. lipnja 2024.

[18] Charmbracelet. Biblioteka wish, . URL https://github.com/charmbracelet/wish. Zadnje pristupljeno 24. lipnja 2024.

[19] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, i Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. U *20th USENIX security symposium (USENIX Security 11)*, 2011.

[20] Kyong-Tak Cho i Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. U *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, stranice 1044–1055, 2016.

[21] Sam Curry. Web hackers vs. the auto industry: Critical vulnerabilities in ferrari, bmw, rolls royce, porsche, and more, 2023. URL https://samcurry.net/

web-hackers-vs-the-auto-industry. Zadnje pristupljeno 07. lipnja 2024.

[22] dissecto. Dissecto knowledgebase - iso-tp, . URL https://munich.dissec.to/kb/chapters/isotp/isotp.html. Zadnje pristupljeno 17. lipnja 2024.

[23] dissecto. Dissecto knowledgebase - vehicle networks, . URL https://munich.dissec.to/kb/chapters/vehicle_networks.html. Zadnje pristupljeno 11. lipnja 2024.

[24] dissecto. Dissecto knowledgebase - popular car hacks, . URL https://munich.dissec.to/kb/chapters/oem/hacks.html. Zadnje pristupljeno 21. lipnja 2024.

[25] dissecto. Dissecto knowledgebase - uds, . URL https://munich.dissec.to/kb/chapters/uds/uds.html. Zadnje pristupljeno 18. lipnja 2024.

[26] Docker. Pomoćni kod za pisanje docker dodataka u programskom jeziku go. URL https://github.com/docker/go-plugins-helpers/tree/master/network. Zadnje pristupljeno 25. lipnja 2024.

[27] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio L´opez, i Vladlen Koltun. Carla: An open urban driving simulator. U *1st Conference on Robot Learning (CoRL 2017)*, 2017.

[28] Jürgen Dürrwang, Johannes Braun, Marcel Rumez, i Reiner Kriesten. Security evaluation of an airbag-ecu by reusing threat modeling artefacts. U *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, stranice 37–43. IEEE, 2017.

[29] Martin Falch. Obd2 explained, 2022. URL https://www.csselectronics.com/pages/obd2-explained-simple-intro. Zadnje pristupljeno 17. lipnja 2024.

[30] Martin Falch. Uds explained, 2022. URL https://www.csselectronics.com/pages/uds-protocol-tutorial-unified-diagnostic-services. Zadnje pristupljeno 17. lipnja 2024.

[31] Martin Falch. Ccp / xcp on can explained, 2022. URL https://www.csselectronics.com/pages/ccp-xcp-on-can-bus-calibration-protocol. Zadnje pristupljeno 17. lipnja 2024.

[32] Christian Gagneraud i Oliver Hartkopp. Docker containers, vxcan and cangw, 2018. URL https://www.spinics.net/lists/linux-can/msg00297.html. Zadnje pristupljeno 25. lipnja 2024.

[33] Robert Bosch GmbH. Can specification 2.0, 1991. URL http://esd.cs.ucr.edu/webres/can20.pdf. Zadnje pristupljeno 07. lipnja 2024.

[34] Robert Bosch GmbH. *Automotive Handbook*. Wiley, 11 izdanju, 2022. ISBN 9781119911906; 1119911907.

[35] Andy Greenberg. Teslas can still be stolen with a cheap radio hack—despite new keyless tech, 2024. URL https://www.wired.com/story/tesla-ultra-wideband-radio-relay-attacks/. Zadnje pristupljeno 21. lipnja 2024.

[36] Oliver Hartkopp. Socketcan. URL https://www.kernel.org/doc/html/next/networking/can.html. Zadnje pristupljeno 23. lipnja 2024.

[37] Numaan Huq, Craig Gibson, i Rainer Vosseler. Driving security into connected cars: threat model and recommendations. *Trend Micro*, 2020.

[38] Daan Keuper i Thijs Alkemade. 'the connected car ways to get unauthorized access and potential implications. *Computest, Zoetermeer, The Netherlands, Tech. Rep*, 2018.

[39] Alissa Knight. *Hacking connected cars: Tactics, techniques, and procedures*. John Wiley & Sons, 2020.

[40] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, i Stefan Savage. Experimental security analysis of a modern automobile. U *2010 IEEE Symposium on Security and Privacy*, stranice 447–462, 2010. doi: 10.1109/SP.2010.34.

[41] Tencent Keen Security Lab. Car hacking research: Remote attack tesla motors, 2016. URL https://keenlab.tencent.com/en/2016/09/19/

Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/. Zadnje pristupljeno 21. lipnja 2024.

[42] Tencent Keen Security Lab. New vehicle security research by keenlab: Experimental security assessment of bmw cars, 2018. URL https://keenlab.tencent.com/en/2018/05/22/New-CarHacking-Research-by-KeenLab-Experimental-Security-Assessment-of-BMW-Cars/. Zadnje pristupljeno 08. lipnja 2024.

[43] Tencent Keen Security Lab. Experimental security research of tesla autopilot, 2019. URL https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf. Zadnje pristupljeno 21. lipnja 2024.

[44] Tencent Keen Security Lab. Mercedes-benz mbux security research report, 2020. URL https://keenlab.tencent.com/en/whitepapers/Mercedes_Benz_Security_Research_Report_Final.pdf. Zadnje pristupljeno 21. lipnja 2024.

[45] Canis Automotive Labs. can security. URL https://canislabs.com/cansecurity/. Zadnje pristupljeno 07. lipnja 2024.

[46] Timm Lauser i Christoph Krauß. Formal security analysis of vehicle diagnostic protocols. U *Proceedings of the 18th International Conference on Availability, Reliability and Security*, stranice 1–11, 2023.

[47] Hangbin Liu. Introduction to linux interfaces for virtual networking, 2018. URL https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking. Zadnje pristupljeno 25. lipnja 2024.

[48] Charlie Miller i Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.

[49] Charlie Miller i Chris Valasek. A survey of remote automotive attack surfaces. *black hat USA*, 2014:94, 2014.

[50] Charlie Miller i Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91):1–91, 2015.

[51] moby. Biblioteka libnetwork. URL https://github.com/moby/moby/tree/master/libnetwork. Zadnje pristupljeno 24. lipnja 2024.

[52] Ahmad MK Nasser. *Automotive Cybersecurity Engineering Handbook: The automotive engineer's roadmap to cyber-resilient vehicles*. Packt Publishing Ltd, 2023.

[53] CBC News. Car theft caught on video shows high-tech thieves stealing lexus in seconds, 2020. URL https://www.cbc.ca/news/canada/toronto/high-tech-vehicle-theft-42-division-1.5651337. Zadnje pristupljeno 10. lipnja 2024.

[54] Sen Nie, Ling Liu, i Yuefeng Du. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25(1):16, 2017.

[55] Sen Nie, Ling Liu, Yuefeng Du, i Wenkai Zhang. Over-the-air: How we remotely compromised the gateway, bcm, and autopilot ecus of tesla cars. *Briefing, Black Hat USA*, 91:1–19, 2018.

[56] Paolo Prinetto, Gianluca Roascio, i Antonio Varriale. Hardware-based capture-the-flag challenges. U *2020 IEEE East-West Design & Test Symposium (EWDTS)*, stranice 1–8. IEEE, 2020.

[57] Saurav Rana. Docker networking demystified, 2023. URL https://blog.kubesimplify.com/docker-networking-demystified. Zadnje pristupljeno 25. lipnja 2024.

[58] Dominik Reinhardt, Maximilian Güntner, Markus Kucera, Thomas Waas, i Winfried Kühnhauser. Mapping can-to-ethernet communication channels within virtualized embedded environments. U *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, stranice 1–10. IEEE, 2015.

[59] Martin Ring, Tobias Rensen, i Reiner Kriesten. Evaluation of vehicle diagnostics security–implementation of a reproducible security access. *SECURWARE 2014*, 213, 2014.

[60] Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, i Ivan Seskar. Security and privacy vulnerabilities of {In-Car} wireless networks: A tire pressure monitoring system case study. U *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[61] Upstream Security. Global automotive cybersecurity report. 2019.

[62] Thomas Sermpinis. Uds fuzzing and the path to game over. TROOPERS, 2022. URL https://troopers.de/troopers22/agenda/tr22-993-uds-fuzzing-and-the-path-to-game-over/.

[63] Thomas Sermpinis. Horror stories from the automotive industry. Chaos Communication Camp, 2023. URL https://pretalx.c3voc.de/camp2023/talk/UEHEVD/.

[64] Craig Smith. *The car hacker's handbook: a guide for the penetration tester*. no starch press, 2016.

[65] Valdemar Švábenskỳ, Pavel Čeleda, Jan Vykopal, i Silvia Brišáková. Cybersecurity knowledge and skills taught in capture the flag challenges. *Computers & Security*, 102:102154, 2021.

[66] ECU Testing. Powertrain control module (pcm) – clearing the confusion. URL https://www.ecutesting.com/categories/ecu-explained/powertrain-control-module/. Zadnje pristupljeno 10. lipnja 2024.

[67] Ken Tindell. Can bus security. Technical report, Canis Automotive Labs, 2022.

[68] Kevin Tindell. Can injection: keyless car theft, 2023. URL https://kentindell.github.io/2023/04/03/can-injection/. Zadnje pristupljeno 19. lipnja 2024.

[69] Tsuyoshi Toyama, Takuya Yoshida, Hisashi Oguma, i Tsutomu Matsumoto. Pasta: Portable automotive security testbed with adaptability. *Proceedings of the Black Hat Europe*, 2018.

[70] Tsuyoshi Toyama, Takuya Yoshida, Hisashi Oguma, i Tsutomu Matsumoto. Carla-pasta, 2018. URL https://github.com/pasta-auto/CARLA-PASTA. Zadnje pristupljeno 21. lipnja 2024.

[71] Tsuyoshi Toyama, Takuya Yoshida, Hisashi Oguma, i Tsutomu Matsumoto. Pasta1.0, 2018. URL https://github.com/pasta-auto/PASTA1.0. Zadnje pristupljeno 21. lipnja 2024.

[72] CANOpen US. History of can. URL https://canopen.us/home/history-of-can. Zadnje pristupljeno 07. lipnja 2024.

[73] Vector. Testing of security-protected ecus and networks with the security manager, 2021. URL https://cdn.vector.com/cms/content/events/2021/Webinars21/Vector_Webinar_Security_Manager.pdf. Zadnje pristupljeno 18. lipnja 2024.

[74] Nils Weiss. Fault injection attacks on secure automotive bootloaders. TROOPERS, 2023. URL https://troopers.de/troopers23/talks/8knnr7/.

[75] Lennert Wouters, Eduard Marin, Tomer Ashur, Benedikt Gierlichs, i Bart Preneel. Fast, furious and insecure: Passive keyless entry and start systems in modern supercars. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):66–85, 2019.

Proširivi sustav za stvaranje CTF zadataka specifičnih za sustave upravljanja i zabave u automobilima

Sažetak

Ovaj rad opisuje implementaciju proširivog sustava za stvaranje CTF zadataka specifičnih za sustave upravljanja i zabave u automobilima. Napravljen je generalni pregled E/E arhitekture modernih automobila, tipova arhitektura kao i najčešćih komunikacijskih, dijagnostičkih i kalibracijskih protokola. Uz navedeno, razmotrena su postojeća istraživanja sigurnosti te stvarni napadi na sustave automobila, ali i napadi na same protokole kojima sustavi automobila komuniciraju. Analizirani su postojeći materijali za obuku sadašnjih ili budućih stručnjaka sigurnosti te je sukladno njihovim prednostima i nedostacima razvijen novi proširivi sustav za stvaranje CTF zadataka. S pomoću novog sustava razvijena su tri zadatka koji demonstriraju moguće napade na protokole CAN i UDS.

Ključne riječi: UDS, CAN, CTF, kibernetička sigurnost, automobilska indrustrija, vozila, simulator

Extensible system for creating CTF tasks specific to automotive control and infotainment systems

Abstract

This paper describes the implementation of an extensible system for creating CTF tasks specific to automotive control and infotainment systems. It provides a general overview of the E/E architecture of modern cars, types of architectures, as well as the most common communication, diagnostic, and calibration protocols. In addition, an overview of existing security research and actual attacks on car systems, as well as attacks on the protocols used by car systems, is provided. Existing training materials for current or future security experts are analyzed, and based on their advantages and disadvantages, a new extensible system for creating CTF tasks has been developed. Using the new system, three tasks were developed that demonstrate possible attacks on the CAN and UDS protocols.

Keywords: UDS, CAN, CTF, cybersecurity, automotive, vehicles, simulator