V. IKERGAZTE

NAZIOARTEKO IKERKETA EUSKARAZ

2023ko maiatzaren 17, 18 eta 19a
Donostia, Euskal Herria

ANTOLATZAILEA:
Udako Euskal Unibertsitatea (UEU)

INGENIARITZA ETA ARKITEKTURA

Fuzzing adimentsua sistema txertatuetan

Maialen Eceiza Olaizola,
Jose Luis Flores eta
Mikel Iturbe Urretxa

43-50 or.
https://dx.doi.org/10.26876/ikergazte.v.03.05

Fuzzing adimentsua sistema txertatuetan

[1]Maialen Eceiza, [1]José Luis Flores, [2]Mikel Iturbe

[1] Department of Industrial Cybersecurity. IKERLAN Technology Research Center, Basque Research and Technology Alliance (BRTA), [2] Department of Computing and Electronics, Faculty of Engineering, Mondragon Unibertsitatea.

meceiza@ikerlan.es, jflores@ikerlan.es, miturbe@mondragon.edu

Laburpena

Egungo munduan geroz eta gailu elektroniko gehiago daude, horietako asko internetera konektatuak. Eurak seguru mantentzea garrantzizkoa da, gailu horien aurkako eraso zibernetikoek ere gora egin baitute. Teknika asko daude segurtasun probak egiteko, horien artean *fuzzing*-a, erabilera orokorreko gailuetan emaitza bikainak lortzen dituen teknika, baina sistema txertatuetan erabili ohi ez dena. Teknika interesgarria denez, bere ezaugarriak aztertu eta sistema txertatuetan erabiltzeko kontuan hartu behar diren ezaugarriak bilatu dira. Alde batetik, fuzzing-aren ezaugarriak aztertu dira, eta bestetik, sistema txertatuenak, biak modu egokian nola elkartu jakiteko.

Hitz gakoak: ahultasun, *fuzzing*, sistema txertatu, zibersegurtasuna

Abstract

The number of electronic devices has increased considerably in recent years, and many of them are connected to the Internet. It is essential to keep them safe, since cybersecurity attacks have also increased. There are many techniques for doing security tests, one of them is fuzzing, which gets excellent results on general-purpose devices, but it is not widely used in embedded systems. As this is an exciting technique, we have studied its characteristics and sought out the features to consider its use in embedded systems. On the one hand, the features of fuzzing have been analyzed, and on the other hand, embedded systems in order to know how to use them together correctly.

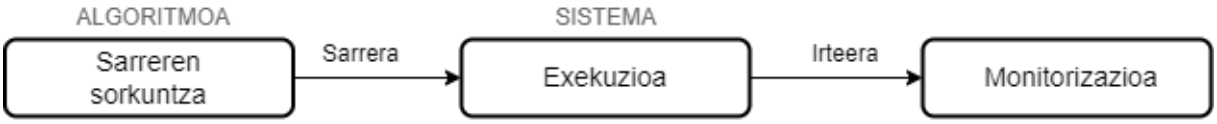
Keywords: vulnerability, fuzzing, embedded system, cybersecurity

1. Sarrera eta motibazioa

Gaur egun, sistema txertatuek eta Internetera konektatuak dauden sistemek gora egin duten bezala (Statista, 2022) zibersegurtasun erasoek ere nabarmen egin dute gora. Azken urteetan gertatu den eraso ezagunenetakoa *Mirai Distributed Denial-of-Service* (DDoS) erasoa da, 2016an gertatu zena (Kolias et al., 2017), eta gaur egun hainbat erasoren arduradun dena, hala ere, beste eraso asko ere gertatu dira. Horren ondorioz, geroz eta garrantzitsuagoa da sistema oro ondo babestea merkaturatu aurretik. Askotan, produktua behin merkataratua edo martxan dagoenean zaila baita eguneratu edo hobekuntzak burutzea. Sistema bat babesteko modu ezberdinak daude, horietako bat *fuzzing*-a da, sistema batean ahultasun eta akatsak aurkitzen laguntzen duen teknika.

Fuzzing-a ahultasunak detektatu edo azaleratzeko teknika automatikoa da. Algoritmo honek sisteman sartuko diren sarrera ezberdinak sortuko ditu, hauek sistemari pasa eta azken honek exekutatu egiten ditu (Zhao et al., 2011). Azkenik, sistemaren irteera monitorizatzen da, errore edo jarrera ezberdin bat detektatzeko helburuarekin sisteman. Prozesu honen oinarrizko funtzionamendua 1. irudian jaso da.

1. irudia. *Fuzzing*-aren oinarrizko prozesua



Teknika hau erabilera orokorreko sistemetan asko erabiltzen da eta 0-egun deituriko ahulezia ugari teknika honen bidez detektatzen dira (Sastry et al., 2017). Halaber, sistema txertatuen berezitasunen ondorioz teknika honekin ez dira emaitza berdinak lortzen, beraz, fuzzing-a sistema txertatuetan erabilia izateko bete beharreko ezaugarriak aztertu dira.

1.1. Sistema txertatuak

Sistema txertatu bat, ekintza espezifiko bat burutzeko diseinatu den sistema da (Papp et al., 2015). *Hardware* eta *software*-z osatuta dago, eta bere baliabideak ekintza espezifiko hori betetzeko diseinatuak daude, beraz beste edozein ekintza gehigarri edo segurtasun sistema ezartzea ezinezkoa izan daiteke baliabide faltagatik. Horregatik, ezin dira segurtasun sistemak jarri, baliabide gehiago beharko baitira. Sistema txertatuak normalean sistema handiago baten parte izaten dira, sistema horrek hainbat funtzio espezifiko burutzen ditu eta sistema txertatuak funtzio espezifiko baten ardura izan ohi du. Gainera, arlo oso ezberdinetan egon daitezke: industrian, osasunean, garraioan, etab. Sistema hauek oso ezberdinak izan daitezke eta horregatik multzokatu egin dira, modu honetan hobeto aztertzeko *fuzzing*-ean eragin ditzaketen ezaugarriak. Sistema txertatuak oso anitzak izan daitezkeenez, zaila da sistema hauek sailkatzea, irizpide ezberdinak erabili daitezkeelako: eskakizun funtzionalak, mikrokontrolagailuaren ezaugarriak, aplikazio mota edo sistema eragilea besteak beste (ikus 2. Irudia). Guk Muench-ek (Muench et al., 2018) esandakoa kontuan hartuz egin dugu gure sailkapena (Eceiza et al., 2021).

- Baliabide altuko sistema: baliabide gehien dituzten sistemak dira. RAM memoria 1 GB baino gehiagokoa da eta hainbat nukleoz osatzen dira. Bestalde, memoriaren kudeaketa unitatea izaten dute, memorian gerta daitezken erroreak detektatzeko. Gainera, sistema eragilea dute, ekintza konplexuak egin ahal izateko. Baliabide hauei esker, hainbat errore detektatzeko gai dira.
- Baliabide ertaineko sistema: 1 MB eta 1 GB arteko RAM memoria dute eta nukleo bat edo bi izan dezakete. Sistema eragilea ere izan dezakete ekintza batzuk burutzen laguntzeko, baina hau mugatua egon ohi da. Memoria kudeatzeko unitatea ere badute, hau baliabide altuko sistemetan baino mugatuagoa da. Horren ondorioz, erroreen aurkako erantzuna mugatuagoa izan ohi da.
- Baliabide gutxiko sistema: RAM memoria 1 MB baino txikiagoa izaten dute, eta nukleo bakarra. Beraien baliabide mugaren ondorioz ezin dute sistema eragilerik eduki. Hala ere, ekintza sinpleak burutzeko gai dira. Ez du memoria kudeatzeko unitaterik, baina memoria babesteko unitatea izaten dute. Horren ondorioz, sistema hauek zailtasunak izaten dituzte erroreak garaiz detektatzeko.

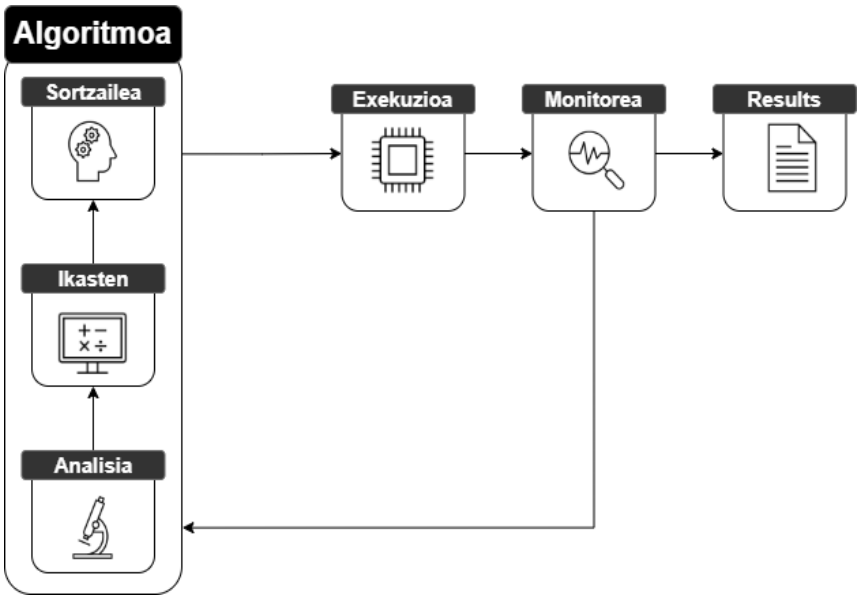
2. irudia. ***Sistema txertatuen aplikazioak.***

1.2. *Fuzzing*

Fuzzing-a errore eta ahuleziak aurkitzeko teknika automatikoa da, algoritmo honek bereziki gaizki sortutako sarrerak sortzen ditu, probatu nahi den sistemak exekutatu ditzan. Ondoren, sistemaren irteerak monitorizatu behar dira. Hasiera batean *fuzzer*-ek pauso horiek bakarrik erabiltzen zituzten, baina azken urteetan *fuzzing*-a eboluzionatu da, adimena emanez eta teknika hobetuz. Gainera, prozesua

hasi aurretik prestaketa fasea egiten da, non sortu beharreko datuen formatua aztertzen den, balio ez duten sarrera gutxiago sor daitezen, hau da, sistema exekutatzeko gai ez diren sarrera kopurua ahalik eta txikiena izan dadin. Alde batetik, berrelikadura erabiltzen da, hau da, sistemaren irteeran monitorizatutako erantzuna erabiltzen da sarrera berriak sortzeko, 3. irudian erakusten den bezala. Bestalde, sarreren sorkuntzan ere teknika berriak ezarri dira, hala nola, analisi estatikoa, algoritmo genetikoak edo estaldura gidatuaren teknika, sarreren sorkuntza prozesua hobetuz (Manes et al, 2021).

3. irudia. *Fuzzer* adimentsua



Fuzzer-ek ezaugarri ezberdinak izan ditzakete, baina hauek sailkatzeko modu errazena lan egiten duten ingurunearen arabera sailkatzea da (Chen et al., 2018).

- Kaxa beltza: lehen *fuzzer*-ak mota honetakoak ziren. Ez dute sistemaren funtzionamendua ezagutu behar, sarrerak sartu eta irteera monitorizatzen dute, sisteman aldaketarik egin izan behar gabe.
- Kaxa zuria: mota honetako *fuzzer*-ek, sistemaren barne funtzionamendua ezagutzeaz gai, honen kode-iturria behar dute, hau aztertu eta aztarnak ezartzeko. Horrela, zehazki ze zati frogatu den dakite. Emaitza hobeak lortzen badituzte ere, informazio asko behera denez, sistema guztiak ezin dira mota hauetako algoritmoekin erabili
- Kaxa grisa: mota hauetako *fuzzer*-ak izan dira sortzen azkenak. Aurreko bien nahasketa bat da, ez du kaxa zurikoek haina informazio behar, baina kaxa beltzekoek baino gehiago behar dute. Modu honetan erantzunak hobetzen dira, baina ez da zehazki barne funtzionamendua ezagutu behar.

2. Fuzzing-a sistema txertatuetan

Gogora ekarriko ditugu *fuzzing*-aren prozesuko etapa nagusiak: prestaketa, datu eraketa, exekuzioa eta monitorizazioa. Gainera, egungo fuzzer gehienetan berrelikadura dagoela ere kontuan izan behar da. Sistema txertatuen kasuan nagusiki bi fasetan egoten da arazoa: monitorizazioan eta berrelikaduran. Sistema txertatuen baliabide mugatuek bi fase horien funtzionamendu egokian zuzenki eragiten dutelako (Muench et al., 2018).

Monitorizazioaren fasean sistemaren irteerak aztertzen dira akats edo jarrera ezberdin bat detektatzeko helburuarekin. Hala ere, fase honetan akats edo jarrera ezberdina detektatzeko sistemak gai izan behar du bere kabuz errorea detektatu eta seinale bat aktibatzeko. Sistema txertatuen kasuan, euren baliabide mugak akats hauen detekzio azkarra ekiditen dute. Sistema hauek ez dira gai errore guztiak detektatzeko eta kasu batzuetan berandu detektatzen dituzte, beraz *fuzzer*-ak ez ditu ondo klasifikatuko errorea eragin edo eragin ez dituzten sarrerak. Hau probatzeko hiru gailu ezberdin erabili dira, baliabide

ezberdinetakoak: Raspberry Pi 4 B[1], Beagle Bone Black[2] eta STM32F429ZI[3]. Raspberry-a baliabide gehieneko gailua da, 4 nukleo eta 4GB memoria ditu. Bestalde, STM32-a baliabide gutxienekoa izango da, eta Beagle Bone Black-ak bien arteko baliabideak ditu. Lortutako emaitzak 1.taulan bildu dira.

1. taula. Erroreak detektatzeko ahalmena hiru motako Sistema txertatuetan.

Errorea	Baliabide gutxiko sistema	Baliabide ertaineko sistema	Baliabide altuko sistema
Koma mugikorreko gainezkaketa	×	×	×
Koma mugikorreko azpi-jarioa	×	×	×
Osoko gainezkaketa	×	×	×
Osoko azpi-jarioa	×	×	×
Osoa zerorekin zatitzea	×	×	×
Koma mugikorrekoa zerorekin zatitzea	×	✓	✓
Segmentazio-akatsa	✓	✓	✓
Buffer-gainezkatzea	×	✓	✓
Liberazio bikoitza	×	✓	✓
Puntero nuluaren deserreferentzia	×	✓	✓
Alboz kanpoko idazketa	×	×	×
Alboz kanpoko irakurketa	×	×	×
Memoriaz kanpo	!	!	!
Pila gainezkaketa	✓	✓	✓
Pila azpi-jarioa	✓	✓	✓
Lerrokatu gabeko helbidea	×	✓	✓

Berrelikaduraren kasuan arazoaren jatorria bera da. Sistema ez bada gai abisu seinalea aktibatzeko errore bat dagoenean, horren ondorioz *fuzzer*-ari bidaliko zaion informazioa (sarrera berriak sortzeko) ez da egokia izango. Honen eragin zuzena erroreak aurkitzeko sarrera gehiago beharko direla izango da eta efizientzia jaitsiko da.

Gainera, *fuzzer*-etan asko zabaldu den teknika instrumentazioa da. Teknika hau aplikatzeko beharrezkoa da kode-iturria edo kode-bitarra izatea, kode horretan aztarnak jartzeko. Horrela, *fuzzing*-a egitean kodearen ze zati frogatu den jakingo da eta ez da behin eta berriz atal bera exekutatuko. Sistema txertatuen kasuan ezin da teknika hau erabili memoria mugatua dutenez ezin dira kodean aztarnak jarri, hauek kodearen tamaina handitzen dute (Muench, 2018).

2.1. Atenea. Sistema txertatuak testeatzeko fuzzerra

Aurreko atalean aipatutako arazoak buruan izanik sistema txertatuetan erabiltzeko Atenea deituriko fuzzer bat diseinatu da. *Fuzzer* honen berezitasun nagusia informazio iturri gehigarri bat erabiltzen duela da, sistemaren erantzun logikoaz gain.

- Albo kanalaren analisia: 1996an azaldu zen lehen aldiz kontzeptu hau, Kocherrek argitaratu zuen aldizkari akademiko batean lehen aldiz (Miller, 1990=. Mota honetako erasoak eraso pasiboak dira eta seinale fisikoak aztertzen dituzte erasoa burutzeko. Beraz, informazio iturri interesgarria da, sisteman eragin zuzenik ez duena, ez baita sisteman aldaketarik egin behar.

[1] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[2] https://beagleboard.org/black
[3] https://www.st.com/en/microcontrollers-microprocessors/stm32f429zi.html

Informazio iturri hauetako batzuk denbora, potentzia kontsumoa, emisio elektromagnetikoa, tenperatura, seinale akustikoak edo argi-emisioa.

Albo kanalaren analisi bidez sistema bere kabuz errore edo jarrera ezberdinak detektatu eta alerta seinalea aktibatzeko gai ez denean informazioa bide honetatik lor daiteke. Teknika hau *fuzzing*-ean monitorizazio fasean sartuko da, erroreak detektatzen laguntzeko.

Akatsak detektatzeko teknika horren bideragarritasuna aztertzeko, aurretiazko proba batzuk egin dira. Horretarako errorerik ez dituen bi kode bat sortu ditugu, bat datu osoekin lan egiteko eta bestea koma mugikorreko datuekin lan egiteko, hauek oinarrizkoak kodeak izango dira. Kode horiek oinarri bezala hartuta beste 16 kode sortu dira, bakoitza errore mota batekin. Orduan, kode hauetako bakoitza *piñata* (baliabide gutxiko gailua) deituriko STM32 batean exekutatu eta seinale elektromagnetiko eta potentzia-seinaleak jaso dira. Seinale hauek preprozesatu eta hiru motako analisi burutu dira (gainbegiratua, ez gainbegiratu orekatua eta ez gainbegiratu ez orekatua) ea albo kanalaren analisia erroreak detektatzeko interesgarria den ikusteko.

- Analisi gainbegiratua: Analisi honetan, errore programei eta oinarrizko programari dagozkien etiketak jarri zaizkie.
- Analisi ez-gainbegiratu orekatua: Analisi hau errealitatetik askoz hurbilago dago, non ez dugun ezagutzen a priori programaren etiketa. Honek ikuspegi errealista hobea islatzen du erroreak bilatzeko orduan. Kasu honetan, programa bakoitzetik lortu den seinale kopurua berdina izango da.
- Analisi ez-gainbegiratu ez-orekatua: Analisi motaren kasuan aurreko egoera berean gaude, hau da, seinaleak ez dira etiketatuak egongo. Gainera, errealitatean lortuko diren seinaletatik gehiengoa errorerik gabekoak izango dira, hau da errorea duten seinaleen kopurua errorerik ez dutenena baino askoz txikiagoa izango da. Horregatik, kasu honetan errore seinaleen proportzioa 1:100 izango da, hau da, 100 seinaleetatik bakarra izango da erroreduna.
  2. taulan analisi bakoitzean lortu diren emaitza onenak bildu dira. Ikus daitekenez erroreak detektatzeko ahalmena %99,79koa da kasu onenean eta %78,13 okerrenean. Taulan, mota bakoitzeko emaitza onenak jaso dira, parametro horiek baitira ondoren erabiliko direnak.

2. taula. Albo kanalaren analisiaren bideragarritasun lanaren emaitzen laburpena.

Analisi mota	Zehaztasuna
Gainbegiratua	% 99,79
Ez gainbegiratu orekatua	% 78,13
Ez gainbegiratu ez orekatua	% 94,01

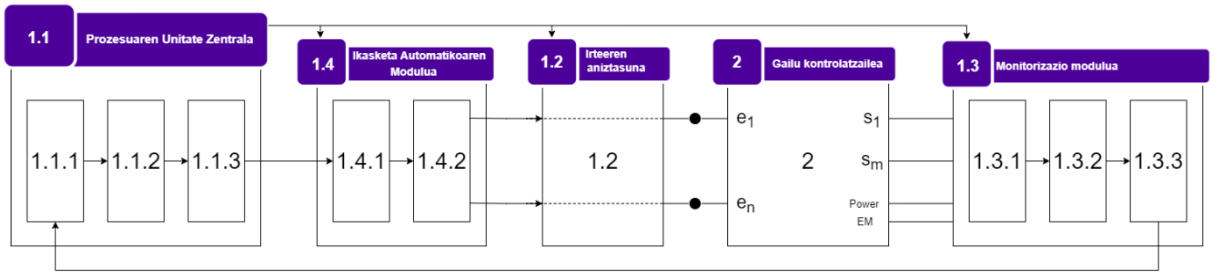
Atenea *fuzzer*-aren helburu nagusia, baldintza okerrenetan modu eraginkorrean lan egitea izango da, hau da, kaxa beltzeko ingurunean eta baliabide gutxiko sistema txertatuetan, horrela, baliabide gehiagoko sistemetan ere erabili ahalko da. Ateneak sistema txertatuetan ahultasunak detektatzea ahalbidetzen du. Gaur egun, fuzzer adimendunek sistemaren erantzun logikoa erabiltzen dute informazio iturri gisa sarrera berriak sortzeko eta hau arazo izan daiteke sistemek nahikoa baliabide ez dutenean, sistema txertatuetan gertatzen den bezala esaterako. Sistema txertatuek, akats batzuk gertatzen direnean ezin baitute abisu-seinalerik piztu edo berehala jakinarazi. Horregatik, akatsak antzemateko beste metodo batzuk aztertzea beharrezkoa da. Lehen esan bezala, sistemaren seinale fisikoak aztertzea informazio iturri interesgarria izan daiteke, seinale horiek akatsaren arabera aldatzen baitira. Beraz, Ateneak sistemaren erantzun logikoaz gain, seinale fisikoengandik lortuko du informazioa sarrera berriak sortzeko.

Atenearen funtzionamendua egokia izan dadin lau atal dira beharrezkoak: zerbitzaria, protokolo itzultzailea, kontrolatzailea eta frogatu nahi den sistema. Lauek batera lan egin beharko dute *fuzzer*-ak modu egokian funtziona dezan.

- Zerbitzaria: Atenearen funtsezko zatia da, prozesuaren erdigunea da. Sistema probatzeko sarrerak sortzeaz arduratuko da. Berrelikadura bidez informazioa jasoko du eta informazio hori erabiliko du sarrera berriak sortzeko.
- Protokolo itzultzailea: honek algoritmoaren eta sistemaren arteko bitartekari lana egingo du. Sistemaren arabera, zati hau aldatu egin daiteke eta, sarrerak sistemak exekutatu ahal izateko egokitzeaz arduratuko da. Gainera, prozesu bera egin beharko du sistemaren irteerekin, egokitu egin behar baitira algoritmoak sarrera berriak sortzeko erabili ahal izateko.
- Kontrolatzailea: sistemaren zati hori arduratzen da sistema ikuskatzeaz, hau da, bere seinale fisikoak bildu eta prozesatzeaz.
- Sistema: zati honetan frogatu nahi den sistema eta monitorizazio sistema daude (frogatu nahi den gailua eta seinale fisikoak jasotzeko sentsoreak).

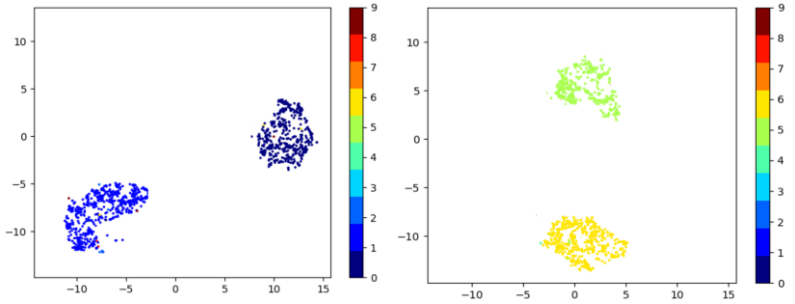
Fuzzer-aren funtzionamendu orokorra 4.irudian jasotzen da. Ziklo hau behin eta berriz egingo da eta ziklo bakoitzean sarrera multzo bat exekutatuko da. Alde batetik, zerbitzarian, prozesu-unitate zentral bat eta ikaskuntza automatikoko modulua ditugu. Horiek lotura dute sarrerak sortzearekin, sarreren aniztasunarekin (sarrerak biltzen dituena) eta monitorizazio-moduluak sistemaren seinale guztiak jasotzearekin. Gainera, gailu kontrolatzailea, probatu nahi den gailu edo sistema da. Irteeren aniztasuna deituriko modulua protokoloarekin erlazionatua dago eta monitorizazio modulua kontrolatzailearekin.

4. irudia. Atenearen funztionamendua



Fuzzer-ak bi fasetan funtzionatzen du. Lehenik, kalibratze fasea eta ondoren, exekuzioa. Kalibratze fasean baliozkoak diren sarrerak bidaltzen dira sistemara, hau da errorerik eragingo ez dutenak. Hauek exekutatu eta bere emaitza logikoa eta fisikoak aztertuko dira. Modu honetan funtzionamendu egokiaren erantzunak zeintzuk diren jakingo dira. Gainera, sistema sarrerarik exekutatzen ez dagoen bitartean sortzen duren erantzun fisikoak ere jasoko dira. Era honetan, emaitzak bi multzotan sailkatuko dira, 5.irudian jaso den bezala.

5. irudia. Kalibrazio seinale elektromagnetiko eta potentzia-seinale preprozesatuak



Bi multzo horiek izan ondoren, exekuzio fasera pasako da, non sistema frogatzeko sarrerak sortzen joango den *fuzzer*-a. Sarrera hauek frogatu nahi den sisteman exekutatuko dira eta monitorizazio moduluak seinaleak jaso, prozesatu eta guztiak bilduko ditu. Gero, unitate zentralean datu basean gordetzen dira datuak eta hauek garrantziaren arabera ordenatu. Datuak ordenatuak daudenean, irizpide elitista jarraituz aukeraketa egiten da, probabilitate-banaketen parametroak estimatzeko. Datu onenetatik sarrera berriak sortuko dira eta prozesua errepikatuko da. Honela, zikloz ziklo erroreak eragiten dituzten sarrerak modu errazagoan aurkituko dira.

3. Ondorioak

Egun, ziber eraso saiakerek gora egiten jarraitzen dute, beraz garrantzizkoa da sistema oro babesturik mantentzea. Horregatik, merkaturatu aurretik ondo frogatzea komenigarria da eta teknika ezberdinak aurkitzen dira horretarako, horien artean *fuzzing*-a. Teknika honen erabilera ez dago hedatua sistema txertatuetan, hauen baliabide eskasiak emaitza eraginkorrak lortzea mugatzen baitu. Horregatik, teknika hau sistema txertatuetara moldatu eta emaitza onak lortzeko ildo berriak bilatu dira. Albo kanalaren analisiaren bidez, sistema txertatuen monitorizazioan dauden mugak gainditzea lor daitekeela ikusi da. Beraz, *fuzzing*-ean albo kanalen analisia informazio iturri bezala erabiliz, sistema txertatuetan *fuzzing*-aren emaitzak eraginkorragoak izan daitezkeela ikusi da.

4. Etorkizuneko ildoak

Lan honetan zehar nagusiki *fuzzing*-arentzat informazio iturri gehigarri gisa albo kanalaren analisia landu dugu, baina beste informazio iturri batzuk ere interesgarriak izan daitezke.

Hasteko, albo kanalaren analisian seinale elektromagnetikoak eta potentzia-seinalean aztertu ditugu, baina badaude beste seinale batzuk informazio iturri gisa erabil daitezkeenak, hala nola, tenperatura, soinua edo argia.

Bestalde, *Joint Test Action Group*-a (JTAG) informazio iturri gisa erabil daiteke. JTAGa sistema txertatuak arazteko interfaze estandar bat da, gainera gailu sorta zabal batean dago eskuragarri. Horregatik, JTAGa erabiltzeak sistema txertatuaren memoriako barne-egoeraren monitorizazioa ahalbidetuko luke, instrumentaziorik eta emulaziorik gabe.

5. Erreferentziak

Chen C., Cui B., Ma J., Wu, R. Guo J., eta Liu W. (2018), A systematic review of fuzzing techniques, *Computers & Security*,, 75, 118-137.

Eceiza (2022), Novel approaches for IoT and Embedded Device Fuzzing and its Evaluation, Tesi memoria, *Mondragon Unibertsitatea*.

Eceiza M., Flores, J.L., eta Iturbe M. (2021), Fuzzing the Internet of Things: A Review on the Techniques and Challenges for Efficient Vulnerability Discovery in Embedded Systems, *IEEE Internet-of-things*, 8, 10390 – 10411.

Kolias C., Kambourakis G., Stavrou A., eta Voas J. (2017), DDoS in the IoT: Mirai and Other Botnets, *Computer*, 50, 80-84.

Li J., Zhao B. eta Zhang C. (2018), Fuzzing: a survey, *Cybersecurity*, 6, 1-13.

Miller B. P., Frediksen L., and So B. (1990), An empirical study of the Reliability of UNIX Utilities, *Communications of the ACM*, 33, 32-44.

Muench M., Stijohann J., Kargl F., Francillon A., eta Balzaroti D. (2018), What you corrupt is not what you crash: Challenges in fuzzing embedded devices, *Network and Distributed Systems Security (NDSS) Symposium 2018*, 1-15.

Papp D., Ma Z.eta Buttyan L. (2015), Embedded systems security: Threats, vulnerabilities, and attack taxonomy, *2015 13th Annual Conference on Privacy, Security and Trust, PST 2015*, 13, 145—152.

Shastry B., Leutner M., Fiebig T., Thimmaraju K., Yamaguchi F., Rieck K., Schmid S., Seifert J.-P., eta Feldmann A., (2017), Static Program Analysis as a Fuzzing Aid, *RAID 2017: Research in Attacks, Intrusions, and Defenses*, 26-47.

Statista, (2022), Internet of Things (IoT) and non-IoT active device connections worldwide from 2010 to 2025. *Sarean dagoen artikulua*.

Zhao J., Wen Y. eta Zhao G. (2011), H-Fuzzing: A New Heuristic Method for Fuzzing Data Generation, *Network and Parallel Computing*, 6985, 32-43.

6. Eskerrak eta oharrak

Lan hau *Novel approaches for IoT and Embedded Device Fuzzing and its Evaluation (*Eceiza, 2022) izenburua duen tesitik eratorritako lana da.