ՀՀ ԳԱԱ ԻՆՖՈՐՄԱՏԻԿԱՅԻ ԵՎ ԱՎՏՈՄԱՏԱՑՄԱՆ ՊՐՈԲԼԵՄՆԵՐԻ ԻՆՍՏԻՏՈՒՏ

Հակոբյան Ջիվան Անդրանիկի

ՖԱԶԶ-ԹԵՍՏԱՎՈՐՄԱՆ ԱՐԴՅՈՒՆԱՎԵՏՈՒԹՅՈՒՆԸ ԲԱՐՁՐԱՑՆՈՂ ՄԵԹՈԴՆԵՐԻ ՄՇԱԿՈՒՄ ԵՎ ԻՐԱԿԱՆԱՑՈՒՄ

Ե.13.04 – «Հաշվողական մեքենաների, համալիրների, համակարգերի և ցանցերի մաթեմատիկական և ծրագրային ապահովում» մասնագիտությամբ տեխնիկական գիտությունների թեկնածուի գիտական աստիճանի հայցման ատենախոսության

ՍԵՂՄԱԳԻՐ

ԵՐԵՎԱՆ – 2021

ИНСТИТУТ ПРОБЛЕМ ИНФОРМАТИКИ И АВТОМАТИЗАЦИИ НАН РА

Акопян Дживан Андраникович

Исследование и разработка методов позволяющих повысить эффективность фаззинга

АВТОРЕФЕРАТ

диссертации на соискание ученой степени кандидата технических наук по специальности 05.13.04 – «Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей»

Ереван - 2021

Ատենախոսության թեման հաստատվել է Երևանի պետական համալսարանում:

Գիտական ղեկավար՝ Ֆիզմաթ. գիտ. դոկտոր Հ. Ի. Ավետիսյան

Պաշտոնական ընդդիմախոսներ՝ Տեխ. Գիտ. դոկտոր Գ. Է. Հարությունյան

Ֆիզմաթ. գիտ. թեկնածու Ռ. Վ. Թոփչյան

Առաջատար կազմակերպություն՝ Հայաստանի Ազգային Պոլիտեխնիկական Համալսարան

Պաշտպանությունը կայանալու է 2021 թվականի դեկտեմբերի 17-ին, ժ. 14:00-ին ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտում գործող 037 «Ինֆորմատիկա» մասնագիտական խորհրդի նիստում հետևյալ հասցեով՝ Երևան, 0014, Պ. Սևակի 1:

Ատենախոսությանը կարելի է ծանոթանալ ՀՀ ԳԱԱ ԻԱՊԻ գրադարանում:

Սեղմագիրն առաքված է 2021թ. նոյեմբերի 8-ին:

Մասնագիտական խորհրդի
գիտական քարտուղար՝ Ֆիզմաթ. գիտ. դոկտոր Հ.Գ. Սարուխանյան

Тема диссертации утверждена в Ереванском государственном университете

Научный руководитель: доктор физ.-мат. наук А. И. Аветисян

Официальные оппоненты: доктор тех. наук Г. Э. Арутюнян

кандидат физ.-мат. наук Р. В. Топчян

Ведущая организация: Национальный Политехнический Университет Армении

Защита состоится 17-ого декабря 2021г. в 14:00, на заседании специализированного совета 037 “Информатика” в Институте проблем информатики и автоматизации НАН РА по адресу: 0014, г. Ереван, ул. П. Севака 1.

С диссертацией можно ознакомится в библиотеке ИПИА НАН РА.

Автореферат разослан 8-ого ноября 2021г.

Ученый секретарь
специализированного совета: д.ф.м.н. А. Г. Саруханян

ԱՏԵՆԱԽՈՍՈՒԹՅԱՆ ԸՆԴՀԱՆՈՒՐ ԲՆՈՒԹԱԳԻՐԸ

Ատենախոսության թեմայի արդիականությունը: Ծրագրային համակարգերը դարձել են մարդու կյանքի անբաժան մասը, և դրանց դերը գնալով ավելի է մեծանում: Հաճախ այդպիսի համակարգերը պարունակում են ծրագրային թերություններ և խոցելիություններ, որոնք կարող են արտահայտվել կիրառության ընթացքում: Նման սխալները երբեմն ոչ միայն ֆինանսական վնասի պատճառ են դառնում, այլ կարող են վտանգել մարդու կյանքն ու առողջությունը: Բազմաթիվ են խոցելիությունների շահագործման օրինակները, մասնավորապես՝ WannaCry[1] կիբերհարձակումը, Ariane 5[2] հրթիռի կործանումը և RSA[3] ալգորիթմի՝ թերությունով իրականացումը: Հայտնաբերված սխալները կարելի է ուղղել և՛ ծրագրի ստեղծման փուլում, և՛ նրա՝ շրջանառության մեջ մտնելուց հետո: Սակայն հարկ է նշել, որ սխալների հայտնաբերումը ծրագրի ստեղծման ընթացքում ավելի հեշտ և մատչելի է դարձնում դրանց շտկումը:

Ծրագրային ապահովումների ստեղծման ընթացքում օգտագործվում են տարատեսակ թեստավորման մեթոդներ և վերլուծության գործիքներ, որոնք թույլ են տալիս հայտնաբերել սխալները: Գոյություն ունեն հատուկ մշակված ստանդարտներ[4, 5], որոնք օգնում են բարձրացնել ծրագրային ապահովման անվտանգությունը: Ըստ այդ ստանդարտների, ֆազզ-թեստավորումը համարվում է այդ մեթոդներից մեկը: Այն տեղի է ունենում ծրագրի կատարվող կոդի հիման վրա: Ֆազզ-թեստավորման գործիքները աշխատանքի ընթացքում մեկնարկում են թիրախային ծրագիրը՝ որպես մուտք փոխանցելով նախապես գեներացված կամ փոփոխված մուտքային տվյալներ, ապա՝ հետևում ծրագրի վարքին: Վթարային ավարտի դեպքում գործիքը տեղեկացնում է դրա և համապատասխան

[1] WannaCry կիբեր հարձակում - https://www.kaspersky.com/resource-center/threats/ransomware-wannacry

[2] Ariane 5 հրթիռի կործանում - http://www-users.math.umn.edu/~arnold/disasters/ariane.html

[3] RSA ալգորիթմի թերություն - https://www.bankinfosecurity.com/tricked-rsa-worker-opened-backdoor-to-apt-attack-a-3504

[4] ГОСТ Р 58143-2018 - http://docs.cntd.ru/document/1200159381

[5] ISO/SAE 21434:2021 "Road vehicles - Cybersecurity engineering" - https://www.sae.org/standards/content/iso/sae21434.d1/

մուտքային տվյալների մասին, ինչը հնարավորություն է տալիս վերարտադրել հայտնաբերված սխալը:

Գոյություն ունեցող ֆազզ-թեստավորման գործիքներն ունեն սահմանափակ կիրառություններ և նախատեսված են կոնկրետ խնդիրների լուծման համար: Օրինակ՝ AFL[6] գործիքը կարող է թեստավորել միայն այնպիսի ծրագրեր, որոնք մուտքային տվյալները ստանում են ֆայլի կամ ստանդարտ մուտքի հոսքի միջոցով, VUzzer[7]-ը՝ IA-32, x86-64 և MIC ճարտարապետությունների համար նախատեսված ծրագրերը, Syzkaller[8]-ն՝ օպերացիոն համակարգի համակարգային ֆունկցիաները, LibFuzzer[9]-ը՝ գրադարանային կամ ծրագրային առանձին ֆունկցիաները, DELTA[10]-ն՝ ցանցային հաղորդակցության SDN[11] համակարգերը:

Գոյություն ունեն այնպիսի ծրագրային ապահովումներ, որոնք մուտքային տվյալները ստանում են մեկից ավել ճանապարհներով՝ ցանցի, ֆայլերի, միջավայրի փոփոխականների, մուտքային դրոշների կամ ստանդարտ մուտքի հոսքի միջոցով: Քանի որ առկա գործիքները ապահովում են սահմանափակ քանակով մուտքերի թեստավորում (սովորաբար ֆայլ և ստանդարտ մուտքի հոսք), վերոնշյալ ծրագրային ապահովումների ամբողջական թեստավորման համար անհրաժեշտ է օգտվել մի քանի գործիքներից: Նման մոտեցումը հանգեցնում է վերլուծության գործընթացի բարդացման, ինչպես նաև ստեղծում է հավելյալ ֆինանսական ծախսեր և, որ ամենակարևորն է, ամբողջությամբ չի լուծում խնդիրը. այդ գործիքները չեն փոխգործակցում միմյանց հետ, այսինքն՝

---

[6] AFL ֆազզ-թեստավորման գործիք - https://www.itgovernanceusa.com/cybersecurity-standards.

[7] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "VUzzer: Application-aware evolutionary fuzzing," Proceedings of the Network and Distributed System Security Symposium, 2017.

[8] Syzkaller ֆազզ-թեստավորման գործիք - https://github.com/google/syzkaller

9 LibFuzzer ֆազզ-թեստավորման գործիք - https://llvm.org/docs/LibFuzzer.html

[10] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras, "DELTA: A security assessment framework for software-defined," *Proceedings of the Network and Distributed System,* 2017

[11] SDN ցանցային հաղորդագրություն - https://en.wikipedia.org/wiki/Software-defined_networking

չեն օգտագործում մյուսների կողմից թեստավորման ընթացքում ձեռք բերված տվյալները:

Իրերի համացանց (IoT) համակարգերը ժամանակի ընթացքում լայն տարածում են գտնում կյանքի տարբեր ոլորտներում: Նման ամենահայտնի և տարածված համակարգերից մեկը՝ Samsung SmartThings-ը[12], հիմնական ֆունկցիոնալությունը իրականացնում է Java գրադարանի միջոցով: Գրադարանների թեստավորման համար նախատեսված գործիքները թեստավորում են միայն առանձին ֆունկցիաներ և չեն դիտարկում վերջիններիս կոմբինացիաները: Հետևաբար՝ առկա գործիքներով հնարավոր չէ էֆեկտիվորեն թեստավորել գրադարանային ֆունկցիաների օգտագործման տարբեր սցենարներ:

Գրեթե բոլոր թեստավորման գործիքները հաշվի չեն առնում մուտքային տվյալի կառուցվածքային առանձնահատկությունը և գեներացնում են այնպիսի մուտքային տվյալներ, որոնց մեծ մասն անտեսվում է թիրախային ծրագրի կողմից որպես խոտան կամ վնասված, ինչն էլ հանգեցնում է թեստավորման ժամանակի վատնմանը: Իսկ կառուցվածքային առանձնահատկություններ հաշվի առնող գործիքները կիրառելի են սահմանափակ թվով մուտքային տվյալների տիպերի համար[13]:

Հաշվի առնելով վերոնշյալ կետերը՝ անհրաժեշտ է իրականացնել ֆազզ-թեստավորման համակարգ, որը զերծ կլինի այլ ծրագրերում առկա թերություններից և սահմանափակումներից, ինչի հիման վրա ձևակերպվել են նպատակն ու խնդիրները:

Ատենախոսության նպատակը և խնդիրները: Ատենախոսության նպատակն է նախագծել և իրականացնել ֆազզ-թեստավորման ընդլայնելի համակարգ, ինչպես նաև մշակել, իրականացնել և այդ համակարգում ներդնել ֆազզ-թեստավորման արդյունավետությունը բարձրացնող նոր մեթոդներ: Նպատակին հասնելու համար առաջարկվել և լուծվել են հետևյալ խնդիրները՝

1. Հետազոտել գոյություն ունեցող ֆազզ-թեստավորման մեթոդները և վերլուծել դրանց թերությունները:

[12] SmartThings ծրագրի վեբ էջ - https://www.smartthings.com

[13] Peach ֆազզ-թեստավորման գործիքի - https://gitlab.com/peachtech/peach-fuzzer-community

2. Նախագծել ֆազզ-թեստավորում կատարող համակարգ, որը կարող է հեշտությամբ ընդլայնվել՝ նոր խնդիրներ լուծելու համար, և թեստավորել նպատակային ծրագրի հետևյալ մուտքային աղբյուրները՝ ստանդարտ մուտքի հոսք, ֆայլեր, ցանցային տվյալներ, միջավայրի փոփոխականներ, մուտքային դրոշներ:
3. Ուսումնասիրել և ստեղծել Java լեզվով գրված գրադարանների ֆազզ-թեստավորման համակարգ, որը կկարողանա թեստավորել փոխկապակցված API ֆունկցիաները:
4. Հետազոտել և մշակել նպատակային ծրագրի մուտքային տվյալների ձևափոխման արդյունավետ մեթոդ, որը հաշվի կառնի այդ տվյալների կառուցվածքային առանձնահատկությունները:

Ատենախոսության հիմնական գիտական արդյունքներն ու գիտական նորույթը: Ատենախոսության շրջանակներում ներկայացված գիտական նորույթը կայանում է հետևյալում՝

1. Առաջարկվել և իրականացվել է ֆազզ-թեստավորման համակարգ, որն ընդլայնվելիության հաշվին կարող է աշխատել ինչպես Linux, այնպես էլ Windows և macOS օպերացիոն համակարգերում: Համակարգում ներդրվել է հավելվածների մեխանիզմ, որի շնորհիվ օգտատերը կարող է ավելացնել սեփական հավելվածը՝ ապահովելով նոր ֆունկցիոնալ հնարավորություններ: Արդյունավետության բարձրացման նպատակով մշակվել է լրացուցիչ գործիք, որը թույլ է տալիս համակարգը մեկնարկել բաշխված և զուգահեռ միջավայրերում: Համակարգի օգնությամբ հայտնաբերվել և հաստատվել են մի շարք նոր ծրագրային սխալներ, որոնք ապացուցում են դրա արդյունավետությունը:
2. Համակարգում իրականացվել է Java լեզվով գրված գրադարանների ֆազզ-թեստավորման մեթոդ: Այն թույլ է տալիս թեստավորել ցանկացած գրադարան՝ նախապես չիմանալով վերջինիս կառուցվածքային առանձնահատկությունները: Մեթոդի կարևոր հատկություններից մեկը կայանում է նրանում, որ այն կարող է աշխատել թե՛ վիրտուալ մեքենաների, և թե՛ իրական սարքերի վրա, ինչն ապահովում է բարձր արագագործություն և ճկունություն: Առաջարկված մեթոդի արդյունավետությունն ստուգվել է Samsung

ընկերության թողարկած SmartThings (իրերի համացանց (IoT) համակարգ) հավելվածում թերություններ հայտնաբերելու միջոցով: Թեստավորման ընթացքում հայտնաբերվել են 15 յուրօրինակ սխալներ, որոնք հաստատվել են Samsung ընկերության կողմից:

3. Իրականացվել է ծրագրի մուտքային տվյալների կառուցվածքային առանձնահատկությունների վերականգնման մեթոդ: Առաջարկվող մեթոդը ստատիկ վերլուծության միջոցով հայտնաբերում է մուտքային տվյալների այն հատվածներն ու դրանց համապատասխան արժեքները, որոնք անհրաժեշտ են մուտքային տվյալներն ընդունելի համարվելու համար:
4. Մշակվել է թեստավորման ընթացքում մուտքային տվյալների գեներացիայի արդյունավետությունը բարձրացնող միջակայքային ձևափոխությունների մեթոդ, որի նպատակն է կրճատել այնպիսի մուտքային տվյալների գեներացիան, որոնք թիրախային ծրագրի կողմից կարող են անմիջապես անտեսվել՝ կառուցվածքի անհամապատասխանության պատճառով:

Ատենախոսության արդյունքների տեսական և գործնական նշանակությունը: Ատենախոսության տեսական և գործնական նշանակությունը կայանում է նրանում, որ մշակվել և իրականացվել են.

- ընդլայնելի ֆազզ-թեստավորման համակարգ՝ տարբեր ֆազզ-թեստավորման խնդիրներ լուծելու համար,
- Java լեզվով գրված գրադարանների ֆազզ-թեստավորման մեթոդ,
- ծրագրի մուտքային տվյալների կառուցվածքային առանձնահատկության վերականգնման մեթոդ,
- միջակայքային ձևափոխության միջոցով գեներացվող մուտքային տվյալների որակը բարելավող մեթոդ:

Մշակված մեթոդների արդյունավետությունն ապացուցված է փորձնական արդյունքներով: Իրականացված ֆազզ-թեստավորման համակարգը և առաջարկվող մեթոդները կիրառվում են Ռուսաստանի Գիտությունների Ակադեմիայի Վ. Պ. Իվաննիկովի անվան Համակարգային Ծրագրավորման Ինստիտուտում (ИСП РАН):

Ատենախոսության արդյունքների փորձարկումը և հրապարակումները: Ատենախոսության հիմնական դրույթները քննարկվել են՝

1. «Իվաննիկովյան ընթերցումներ» միջազգային գիտաժողով (Ivannikov Memorial Workshop), Վիլիկիյ Նովգորոդ, Ռուսաստան, 13-14 սեպտեմբերի2019 թ.:

2. Վ. Պ. Իվաննիկովի անվան Ռուսաստանի Գիտությունների Ակադեմիայի Սիստեմային Ծրագրավորման Ինստիտուտի բաց գիտաժողով, Մոսկվա, Ռուսաստան, 5-6 դեկտեմբերի 2019 թ.:

Ատենախոսության թեմայով հրատարակվել են 4 գիտական աշխատանքներ, որոնցից [1]-ը ինդեքսավորվել է Scopus շտեմարանում:

Ատենախոսության ծավալը և կառուցվածքը: Ատենախոսությունը կազմված է ներածությունից, չորս գլուխներից, եզրակացությունից և հավելվածներից, ամբողջ ծավալը կազմում է 102 էջ՝ ներառյալ 21 նկար և 4 աղյուսակ: Ատենախոսությունը պարունակում է 106 անուն հղում:

Ատենախոսության համառոտ բովանդակությունը

Ներածությունում հիմնավորված է ատենախոսության թեմայի արդիականությունը, սահմանված են հետազոտության խնդիրներն ու պահանջները, ներկայացված է հետազոտության նպատակը, հիմնական արդյունքներն ու գիտական նորույթը, ինչպես նաև հետազոտության արդյունքների գործնական և կիրառական նշանակությունը:

Ատենախոսության առաջին՝ «Հետազոտական ակնարկ» գլխում ներկայացված են ատենախոսության թեմայի հետ ընդհանրություն ունեցող ֆազզ-թեստավորման մեթոդներն և գործիքները:

Առաջին բաժնում ներկայացված են թիրախային ծրագրի ֆազզ-թեստավորման մեթոդներն ու մոտեցումները, ինչպես նաև դրանց առավելություններն ու թերությունները:

Երկրորդ բաժնում ներկայացված են ֆազզ-թեստավորման տեսակները, առավել լայն կիրառում ստացած ֆազզ-թեստավորման գործիքները, ինչպես նաև դրանց թերությունները և առավելությունները: Ֆազզ-թեստավորման գործիքները կարելի է բաժանել երեք դասի՝ սպիտակ տուփ (white box), մոխրագույն տուփ (grey box) և սև տուփ (black box)[14]: Ատենախոսության աշխատանքի հիմքում ընկած է մոխրագույն տուփի դասին պատկանող ֆազզ-թեստավորումը:

Երրորդ բաժնում ներկայացված են Java լեզվով գրված իրերի համացանց (IoT) համակարգերի ֆազզ-թեստավորման մեթոդները: Գոյություն ունեցող մեթոդներն ու գործիքները օգտագործելու համար անհրաժեշտ է նախապես իմանալ թիրախային ծրագրի առանձնահատկությունները, ինչի համար անհրաժեշտ է օգտագործել հավելյալ գործիք: Առաջարկված է ֆազզ-թեստավորման գործիք, որն ինքնաշխատ է և կախված չէ հավելյալ գործիքներից:

Չորրորդ բաժնում ներկայացված են ֆազզ-թեստավորման ընթացքում թիրախային ծրագրի համար մուտքային տվյալների գեներացման մեթոդներն ու դրանց թերությունները: Ներկայացված մեթոդները հիմնականում հաշվի չեն առնում ծրագրի մուտքային տվյալների կառուցվածքային առանձնահատկությունները և գեներացնում են այնպիսի մուտքային տվյալներ, որոնք անտեսվում են թիրախային ծրագրի կողմից: Ատենախոսության շրջանակներում առաջարկվում է նոր մեթոդ, որը վերլուծելով մուտքային տվյալներն ու նրանց միջոցով հայտնաբերված կատարման ճանապարհները, պարզում է ծրագրի կառուցվածքային առանձնահատկությունները: Ստացված տեղեկությունների հիման վրա գեներացվում են այնպիսի մուտքային տվյալներ, որոնք թիրախային ծրագրի կողմից չեն անտեսվում:

Հինգերորդ բաժնում ներկայացվել են հետևյալ եզրակացությունները՝

1. Դիտարկված բոլոր ֆազզ-թեստավորման գործիքները նախատեսված են կոնկրետ խնդրի կամ օպերացիոն համակարգի համար: Մեկից ավելի մուտքային տիպ ունենալու կամ տարբեր օպերացիոն համակարգերում աշխատող ծրագրերը վերլուծելու համար անհրաժեշտ է ունենալ մի քանի ֆազզ-գործիք: Հետազոտության

[14] V. J. M. Manès, H. Han, Ch. Han, S. Kil Cha, M. Egele, E. J. Schwartz, M. Woo, "The Art, Science, and Engineering of Fuzzing: A Survey," *IEEE Transactions on Software Engineering,* 2019

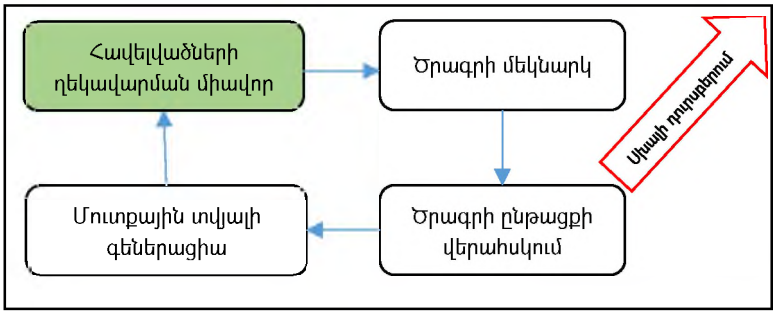
արդյունքում նախագծված և իրականացված ֆազզ-թեստավորման միջավայրը կախված չէ օպերացիոն համակարգից և ընդլայնվելու հնարավորություն է տալիս՝ հաշվի առնելով իր դեմ դրված նոր խնդիրները:

2. Ֆազզ-թեստավորման այն գործիքները, որոնք նախատեսված են Java լեզվով գրված գրադարանների՝ մասնավորապես իրերի համացանց (IoT) համակարգեր թեստավորելու համար, հիմնականում թիրախային ծրագիրը մեկնարկում են վիրտուալ մեքենաներում: Այդ իսկ պատճառով հնարավոր չէ թեստավորել այնպիսի համակարգեր, որոնք աշխատում են միայն տվյալ համակարգին հատուկ միջավայրում իսկ որոշներն էլ թեստավորման համար պահանջում են գրադարանի կառուցվածքային նկարագրություն մասնավորապես՝ հասանելի ֆունկցիաների նկարագրերը: Նման տեղեկություն ստանալու համար մինչ ֆազզ-թեստավորումը անհրաժեշտ է մեկ այլ գործիքի միջոցով ստանալ այդ տեղեկությունը: Ատենախոսության շրջանակներում մշակված վերլուծության միջավայրը թույլ է տալիս առանց նախապատրաստական աշխատանքների ֆազզ-թեստավորման միջոցով վերլուծել վերոնշյալ համակարգերը և հայտնաբերել սխալներ: Բացի այդ, վերլուծությունը կարող է տեղի ունենալ նաև իրական սարքավորումների վրա, ինչը համակարգի համար ապահովում է բնական միջավայր և արագագործություն:
3. Գոյություն ունեցող ֆազզ-թեստավորման գործիքների մեծ մասը նոր մուտքային տվյալներ գեներացնելիս հաշվի չեն առնում դրանց կառուցվածքային առանձնահատկությունները: Այդ պատճառով գեներացված տվյալների մեծամասնությունն անտեսվում են թիրախային ծրագրի կողմից, որպես թերի մուտքային տվյալ: Այն գործիքները, որոնք հաշվի են առնում կառուցվածքային առանձնահատկությունները գեներացիայի ընթացքում օգտվում են համաձայնագրերի մասին տեղեկագրերից: Գոյություն ունեցող տեղեկագրերի և համաձայնագրերի քանակները խիստ բազմազան են: Ատենախոսությունում առաջարկվում է նոր մեթոդ, որը թույլ է տալիս դինամիկ կերպով վերականգնել մուտքային տվյալների կառուցվածքային առանձնահատկությունները: Այնուհետև կիրառվում է միջակայքային ձևափոխությունների մեթոդ, որը թույլ է տալիս

ձևափոխել մուտքային տվյալի միայն այն հատվածները, որոնք չեն խախտի կառուցվածքային առանձնահատկությունները:

Ատենախոսության երկրորդ՝ «ISP-Fuzzer - Ընդլայնելի ֆազզ-թեստավորման միջավայր» գլխում նկարագրված է ISP-Fuzzer վերլուծության միջավայրի ընդհանուր ճարտարապետությունը և առանձնահատկությունները: Միջավայրը նախագծելիս հաշվի է առնվել, որ այն պետք է լինի.

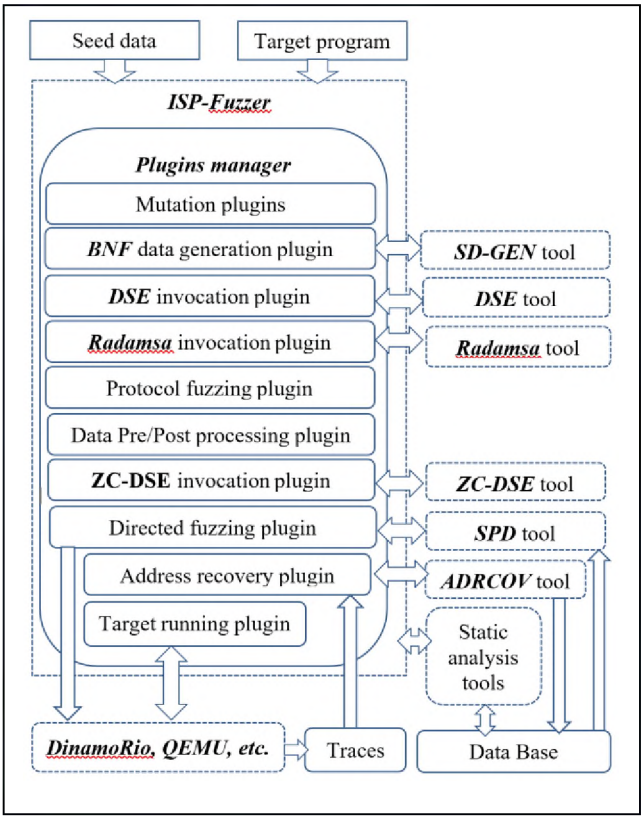
- ընդլայնելի և կարողանա ներառել նոր ֆունկցիոնալություններ,
- մրցունակ՝ առկա այլ վերլուծական գործիքների հետ,
- օպերացիոն համակարգից անկախ,
- կիրառելի զուգահեռ և բաշխված համակարգերում:



Նկար 1. Դինամիկ վերլուծության միջավայր

Առաջին բաժնում նկարագրված է վերլուծության միջավայրի կառուցվածքային նկարագրությունը: Որպեսզի միջավայրը կարողանա ներառել նոր ֆունկցիոնալ հնարավորություններ, ներդրվել է հավելվածների մեխանիզմ: Ինչպես ցույց է տրված նկար 1-ում, ներդրված մեխանիզմը ավելանում է դասական ֆազզ-թեստավորման հիմնական երեք փուլերին: Ավելացված չորրորդ փուլը՝ հավելվածների ղեկավարման միավորը, աշխատացնելով առկա հավելվածները, թույլ է տալիս բարձրացնել վերլուծության արդյունավետությունը և, առհասարակ, միջավայրը դարձնում կիրառելի՝ պահանջվող խնդիրը լուծելիս: Հատուկ կարգավորումների ֆայլի շնորհիվ օգտատերը կարող է հեշտությամբ ավելացնել նոր հավելված կամ փոխել առկա հավելվածների վարքը: Որպեսզի միջավայրը լինի կիրառելի Windows, MacOS և Linux օպերացիոն համակարգերում, ստեղծվել

և ներդրվել է հատուկ գրադարան, որը պարունակում է միջավայրին անհրաժեշտ բոլոր ֆունկցիոնալությունները:



Նկար 2. Մշակված միջավայրի կառուցվածքը

Երկրորդ բաժնում նկարագրված են իրականացված և միջավայրում ինտեգրված հավելվածները (նկար 2), ինչպես նաև բազմապրոցես ու բաշխված համակարգերում աշխատելու մեխանիզմները:

Մուտքային տվյալների գեներացման հավելված: Ավելացվել է հատուկ հավելված, որը թույլ է տալիս օգտատիրոջը ղեկավարել տվյալների գեներացման գործընթացը, ըստ ցանկության անջատել կամ միացնել առկա ալգորիթմները, ինչպես նաև ավելացնել սեփականը:

Դինամիկ Սիմվոլային Կատարում – ԴՍԿ (անգլ. Dynamic Symbolic Execution: DSE) կանչի հավելված: Ինչպես ցույց են տվել որոշ աշխատություններ, ֆազզ-թեստավորման աշխատանքի ընթացքում կարող են հանդիպել բարդ պայմաններով ճյուղավորման օպերատորներ, որոնց բոլոր ճյուղերի՝ ֆազզ գործիքով անցնելը գրեթե անհնար է: Այդպիսի դեպքերում լավագույն լուծումներից մեկը ԴՍԿ գործիքի ինտեգրումն է[15]: ԴՍԿ-ի միջոցով հայտնաբերվում են այնպիսի մուտքային տվյալներ, որոնք անցնում են վերոնշյալ ճյուղավորման օպերատորներով: Գործիքն աշխատանքից հետո հայտնաբերված մուտքային տվյալները փոխանցվում են ֆազզերին և օգտագործվում հետագա մուտքային տվյալներ գեներացնելիս:

Արձանագրությունների ֆազզ-թեստավորման հավելված: Ծրագրային ապահովման ամբողջական վերլուծության համար անհրաժեշտ է վերլուծել ծրագրի բոլոր մուտքերը: Ներկայումս երկու կամ ավելի համակարգերի հաղորդակցության կազմակերպման համար լայն կիրառություն է ստացել արձանագրությունների միջոցով տվյալների փոխանակումը: Ուստի անհրաժեշտ է վերլուծության մեջ ներառել նաև մուտքային տվյալների արձանագրությունների միջոցով փոխանցման տարբերակը: Ելնելով վերը նշված պահանջներից՝ դինամիկ վերլուծության միջավայրում ներդրվել է արձանագրությունների վերլուծության հավելված, որի հիմքում ընկած է Peach գործիքը: Ստանալով անհրաժեշտ արձանագրության տեսակը՝ միջավայրը ընտրում է հատուկ peach pit ֆայլը, որում մանրամասն նկարագրված է արձանագրության կառուցվածքը: Peach pit ֆայլը թույլ է տալիս ձևափոխել միայն ինֆորմացիոն դաշտերը՝ ծառայողականները թողնելով անփոփոխ:

ZC-DSE գործիքի կանչի հավելված: Մուտքային տվյալներ գեներացնելիս մեծ է հավանականությունը, որ ձևափոխության ալգորիթմները կփչացնեն մուտքային տվյալի ծառայողական դաշտերում գրված ինֆորմացիան: Այդ պատճառով ստեղծվել է ZC-DSE գործիքը, որը կանխում է նման դաշտերի ձևափոխումը: Գործիքի և դրա միջոցով ձեռք բերված արդյունքները ավելի մանրամասն ներկայացված են չորրորդ գլխում:

[15] Ս. Ասրյան, Կատարող կոդում սխալների հայտնաբերման դինամիկ վերլուծության մեթոդներ, Երևան, 2019

Ծրագրի մեկնարկի հավելված: Առաջարկվող ֆազզ-թեստավորման միջավայրը մոխրագույն արկղ (grey box) դինամիկ վերլուծության տեսակի է: Այսպիսի գործիքներին հատուկ է վերլուծության ընթացքում թիրախային ծրագրի երկուական կոդի ծածկույթի հաշվումը: Դրա համար կատարվող կոդի յուրաքանչյուր բազային բլոկում ավելացվում են հատուկ հրամաններ, որոնք պահպանում են կատարվող բազային բլոկի մասին տեղեկատվությունը: Այն ստանալու համար օգտագործվել են DynamoRio[16] և QEMU[17] գործիքները, որոնք ծրագրի աշխատանքի ընթացքում ավելացնում են մեքենայական անհրաժեշտ հրամանները: Այս գործիքների առավելությունը կայանում է նրանում, որ նրանք աշխատում են Windows, MacOS և Linux օպերացիոն համակարգերում, ինչպես նաև Intel և ARM պրոցեսորների համար նախատեսված մեքենայական կոդի հետ: Այս գործիքների ինտեգրման համար ստեղծվել է հատուկ հավելված, որը պատասխանատու է թիրախային ծրագրի մեկնարկի և երկուական կոդի ծածկույթի հավաքման համար: Հավելվածը նույնպես ընդլայնելի է, այսինքն՝ օգտատերը կարող է ավելացնել կոդի ծածկույթի հաշվման իր տարբերակը:

Որպեսզի հնարավոր լինի միջավայրն օգտագործել զուգահեռ և բաշխված համակարգերում, ստեղծվել է հատուկ գործիք: Անհրաժեշտ է ստեղծել հատուկ աշխատանքային միջավայր, որը հասանելի է բոլոր մեքենաներին: Գործիքը յուրաքանչյուր մեքենայում զուգահեռ մեկնարկում է ֆազզ-թեստավորման օրինակներ, որոնք էլ իրենց հերթին ընդհանուր աշխատանքային տիրույթում ստեղծում են սեփական, լոկալ աշխատանքային միջավայրը, որտեղ պահում են վերլուծության արդյունքները: Սինխրոն աշխատելու համար յուրաքանչյուր ֆազզ-թեստավորման գործիք պարբերաբար դիտարկում է մյուս բոլոր գործիքների վերլուծության արդյունքները և ինտեգրում հայտնաբերած սեփական արդյունքներից տարբերվողները:

Իրականացված գործիքն օժտված է հետևյալ հատկություններով.

1. Մեկնարկված ֆազզ-թեստավորման պրոցեսների դիտարկում: Այս հատկության միջոցով գործիքը մեկնարկում է թեստավորման նոր պրոցես, եթե նախորդներից որևէ մեկի աշխատանքն ընդհատվել է:

[16] DynamoRIO դինամիկ գործիքավորման միջավայր - https://dynamorio.org
[17] QEMU վիրտուալ մեքենա - https://www.qemu.org

2. Ինտերակտիվ կերպով թեստավորման պրոցեսների ավելացում և պակասեցում:
3. Վարպետ (չի օգտագործում պատահական տվյալների գեներացման ալգորիթմ) և ենթակա (օգտագործում է պատահական տվյալների գեներացման ալգորիթմ) թեստավորման պրոցեսների քանակի կարգավորում:

Երրորդ բաժինը բաղկացած է եզրակացությունից և ֆազզ-թեստավորման գործիքի կիրառմամբ ստացված արդյունքներից:

Ստեղծվել է դինամիկ վերլուծության միջավայր, որը շնորհիվ.

- հավելվածների մեխանիզմի ընդլայնելի է և կարող է ներառել առաջացած նոր խնդիրների լուծումներ,
- հատուկ ստեղծված գրադարանի կիրառելի է Windows, Linux և MacOS օպերացիոն համակարգերում,
- զուգահեռացման գործիքի կարող է աշխատել բաշխված համակարգերում:

Ստեղծված դինամիկ վերլուծության միջավայրի արդյունավետությունն ապացուցված է իրական ծրագրային ապահովումների վերլուծության միջոցով: Միջավայրը ստուգվել է տարբեր ծրագրերի վերլուծությամբ տարբեր օպերացիոն համակարգերում:

Ծրագրի անվանում	Օպերացիոն համակարգ	Լրացուցիչ հավելված	Սխալի տիպ	Սխալների քանակ
colcrt	Ubuntu-18.04	-	վթարային ավարտ	19
column	Ubuntu-18.04	-	վթարային ավարտ	7
gtbl	Ubuntu-18.04	-	վթարային ավարտ	20
troff	Ubuntu-18.04	-	վթարային ավարտ	4
tbl	Ubuntu-18.04	-	վթարային ավարտ	24
exiv2	Ubuntu-18.04	-	անվերջ աշխատանք	1
captoinfo	Ubuntu-16.04	-	վթարային ավարտ	1
gtbl	Ubunut-16.04	-	վթարային ավարտ	20
infotocap	Ubuntu-16.04	-	վթարային ավարտ	1
exiv2	Ubuntu-16.04	ԴՍԿ	վթարային ավարտ	1
bison	Ubuntu-16.04	-	անվերջ աշխատանք	2
jasper	Ubuntu-16.04	ԴՍԿ	վթարային ավարտ	19
colcrt	Ubuntu-16.04	ԴՍԿ	վթարային ավարտ	1

iptables-xml	Ubuntu-16.04	ԴՍԿ	վթարային ավարտ	2
dc	Ubuntu-16.04	ԴՍԿ	վթարային ավարտ	4
teckit_compile	Ubuntu-16.04	ԴՍԿ	վթարային ավարտ	1
tic	Debian 6.0.10	ԴՍԿ	վթարային ավարտ	2
tbl	Debian 6.0.10	ԴՍԿ	վթարային ավարտ	153

Աղյուսակ 1. Միջավայրի օգնությամբ հայտնաբերված սխալներ

Ատենախոսության երրորդ՝ «Java լեզվով գրված գրադարանների վերլուծություն» գլխում նկարագրված է առաջարկվող վերլուծության համակարգը, նրա աշխատանքի սկզբունքը, ինչպես նաև, համակարգի միջոցով ձեռք բերված արդյունքները: Մշակվել և ISP-Fuzzer միջավայրում ներդրվել է նոր համակարգ՝ նախատեսված Java լեզվով գրված գրադարանների, մասնավորապես՝ ինտերնետ համացանց (IoT) համակարգերի վերլուծություն համար:

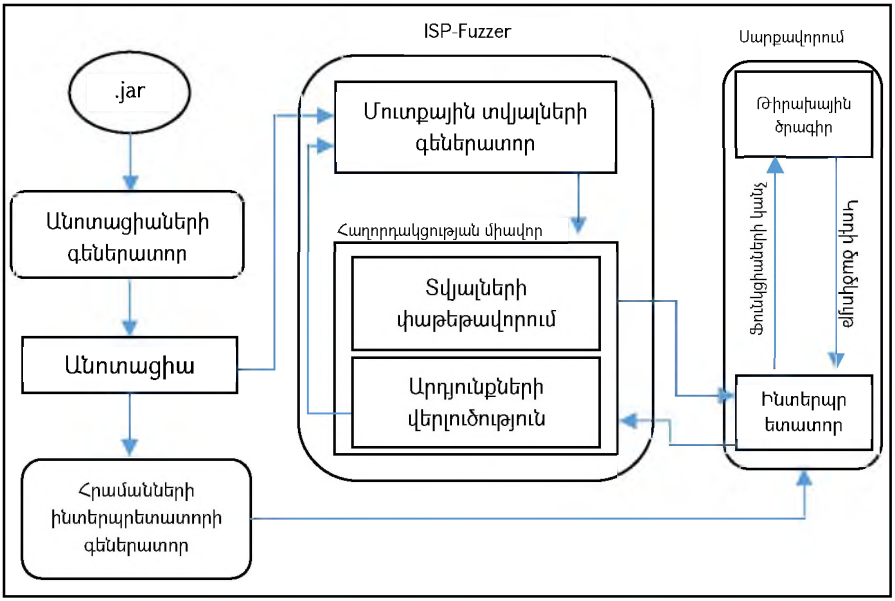
Առաջին բաժնում տեղ է գտել համակարգի ընդհանուր նկարագիրը: Համակարգն ամբողջությամբ ինքնաշխատ է և նախատեսված է Java լեզվով գրված իրերի համացանց (IoT) համակարգերի վերլուծության համար: Այն

- պետք է կարողանա թեստավորել Java լեզվով գրված իրերի համացանց (IoT) համակարգի գրադարան՝ չունենալով դրա մասին որևէ նախնական տվյալ:
- պետք է լինի ամբողջությամբ ինքնաշխատ:
- պետք է լինի միջավայրից անկախ:
- պետք է լինի ընդլայնելի:

Համակարգը բաղկացած է մի քանի գործիքներից և հավելվածներից, որոնք հատուկ ստեղծվել են տվյալ խնդրի լուծման համար: Ինչպես պատկերված է նկար 3-ում, որպես մուտք այն ստանում է թիրախային գրադարանը: Առաջին գործիքի միջոցով թիրախային գրադարանի յուրաքանչյուր ֆունկցիայի և դասի համար ստեղծում է համապատասխան անոտացիա, որն օգտագործվում է մուտքային տվյալների և ֆունկցիաների կանչի հաջորդականություններ գեներացնելիս: Անոտացիա ստեղծելիս յուրաքանչյուր ֆունկցիա կամ դաս համարակալվում է, ինչի շնորհիվ այն հնարավոր է միարժեքորեն նույնականացնել:

Հաջորդ գործիքի միջոցով ստեղծված մեկնաբանության համար գեներացվում է գրադարանային ֆունկցիաների կատարման ինտերպրետատոր: Մեքենան անհրաժեշտ է ներդնել պահանջվող վիրտուալ միջավայրում կամ իրական

սարքավորման մեջ, ինչը թույլ կտա ձերբազատվել միջավայրի կախվածությունից և ապահովել թիրախային գրադարանի իրական վարք:



Նկար 3. Իրերի համացանց համակարգի ֆազզ-թեստավորման համակարգի կառուցվածքը

Գրադարաններում սխալներ հայտնաբերելու համար համակարգն օգտագործում է ֆազզ-թեստավորման գործեք: Նրանում ներդրվել են երկու հավելվածներ, որոնցից առաջինը, մուտքում ստանալով գրադարանի նկարագրությունը, գեներացնում է ֆունկցիաների կանչի հաջորդականություններ: Երկրորդ հավելվածը կապ է ապահովում վերլուծության գործիքի և մեքենայի միջև: Նման կապ ապահովելու համար ստեղծվել է հատուկ արձանագրություն, որի միջոցով վերլուծության գործիքը ուղարկում է գեներացված մուտքային տվյալները և ստանում կոդի ծածկույթի և ծրագրի վարքի մասին տեղեկություն:

Երկրորդ բաժնում նկարագրված են երկու ինքնաշխատ գործիքներ: Առաջինը գրադարաններում առկա դասերի և ֆունկցիաների համար ստեղծում է հատուկ անոտացիաներ, որոնք կանոնակարգված են և ամբողջությամբ արտահայտում

են ֆունկցիաների ու դասերի նկարագրությունը: Անոտացիան կառուցելիս գործիքը նախ համարակալում է բոլոր ֆունկցիաները, ապա դիտարկում փոխանցվող արգումենտների և վերադարձվող արժեքի տիպերը: Յուրաքանչյուր արգումենտ նկարագրելու համար գործիքը օգտագործում է հետևյալ 3 դասակարգիչներից մեկը՝

- MustBeNull՝ չի ընդունում 0-ից տարբեր արժեք,
- NotNull՝ չի կարող ընդունել 0 արժեք,
- Any՝ կարող է ընդունել ցանկացած արժեք:

Այս դասակարգիչները անհրաժեշտ են մուտքային տվյալների գեներատորին՝ ընդունելի տվյալներ գեներացնելու համար:

Երկրորդ գործիքը ստեղծված անոտացիաների հիման վրա կառուցում է հրամանների թարգմանիչ (կատարող մեքենա): Այն անհրաժեշտ է ներդնել վիրտուալ մեքենայի կամ սարքավորման մեջ: Թարգմանիչը դինամիկ վերլուծության գործիքից ստանում է ֆունկցիաների կանչի հաջորդականություն և կանչում գրադարանի համապատասխան ֆունկցիաները: Յուրաքանչյուր կանչի ընթացքում թարգմանիչը հետևում է ծրագրի վարքին և անսպասելի վարքի դեպքում տեղեկացնում հայտնաբերված սխալի մասին: Որպեսզի վերլուծության գործիքը կարողանա գնահատել գեներացված մուտքային տվյալների պիտանելիությունը, ծրագրի յուրաքանչյուր մեկնարկի ժամանակ կուտակվում և ուսումնասիրում է կոդի ծածկույթի մասին տեղեկատվությունը:

<u>Երրորդ բաժնում</u> նկարագրված են անոտացիաների միջոցով կապակցված ֆունկցիաների կանչերի հաջորդականություն գեներացնող և մեքենային փոխանցող հավելվածները:

Հաջորդականության գեներացման համար հավելվածը ֆունկցիաների ցուցակից ընտրում է կամայական մեկը և դիտարկում դրա արգումենտները հերթականությամբ: Եթե արգումենտն ունի պրիմիտիվ տիպ, ապա գեներացվում է այդ տիպի արժեք, հակառակ դեպքում՝ ֆունկցիաների ցուցակից ընտրում և հաջորդականությունում ավելացնում է այնպիսին, որի վերադարձի տիպը համընկնում է արգումենտի տիպի հետ: Եթե ֆունկցիաների ցուցակում չկա նմանը, ապա ավելացվում է այդ դասի կառուցիչի կանչ: Նույն գործողությունները ռեկուրսիվ կրկնում են ավելացված ֆունկցիաների համար: Պրիմիտիվ տիպեր գեներացնելիս հավելվածն օգտվում է անոտացիաներում առկա

դասակարգիչներից: Եթե դասակարգիչը MustBeNull կամ Any է, ապա արգումենտում փոխանցում է 0 արժեք, հակառակ դեպքում՝ կամայական, զրոյից տարբեր արժեք:

Ֆազզ-թեստավորման գործիքի և սարքավորման միջև կապ ստեղծելու համար գործիքում ներդրվել է առանձին հավելված, որը սարքավորման հետ հաղորդակցվում է հատուկ արձանագրության միջոցով: Հավելվածը, մուտքում ստանալով գեներացված կանչի հաջորդականությունը, հատուկ ալգորիթմով փաթեթավորում և համացանցի միջոցով այն ուղարկում է սարքավորմանը: Վերջինս ապափաթեթավորում է ստացված հաջորդականությունը և կանչում համապատասխան ֆունկցիաները: Կատարման ավարտին սարքավորումը վերադարձնում է կոդի ծածկույթի և ծրագրի վարքի մասին տեղեկություն: Հավելվածի՝ համացանցով աշխատելը թույլ է տալիս վերլուծությունը կատարել բազմապրոցես և բաշխված համակարգերում:

Չորրորդ բաժնում նկարագրված են համակարգի միջոցով ձեռք բերված արդյունքները և գոյություն ունեցող գործիքների հետ համեմատությունները:

Համակարգի ստեղծման ժամանակ առաջնահերթ էր PluginBase գրադարանի վերլուծությունը, որը հանդիսանում է Samsung SmartThings հավելվածի հիմնական գրադարանը: Վերլուծության արդյունքում հայտնաբերվել են 15 յուրօրինակ սխալներ, որոնք բոլորը հաստատվել են Samsung Electronics R&D հետազոտական խմբի կողմից: Գործընթացը կատարվել է 2 իրական սարքավորումների վրա զուգահեռ և տևել է 7 օր:

Ատենախոսության չորրորդ՝ «Վերլուծության արդյունավետության բարձրացումը միջակայքային ձևափոխությունների միջոցով» գլխում նկարագրված է մեթոդ, որը վերլուծության ընթացքում մուտքային տվյալներ գեներացնելիս հաշվի է առնում ծրագրի առանձնահատկությունները: Նկարագրված մեթոդը թույլ է տալիս տվյալներ գեներացնելու ընթացքում ձևափոխել առանձին հատվածները: Նման մոտեցումը կանխում է մուտքային տվյալների ծառայողական դաշտերի ձևափոխությունը և բարձրացնում վերլուծության արդյունավետությունը:

Առաջին բաժնում նկարագրված է առաջարկվող մեթոդի ընդհանուր նկարագիրը: Մեթոդը կիրառելի է այն և միայն այն դեպքում, երբ թիրախային ծրագրի վերլուծության ընթացքում հայտնաբերվել են առնվազն երկու՝ իրարից տարբեր կատարման ճանապարհներ: Մեթոդը բաղկացած է երկու քայլից,

որոնցից առաջինում օգտագործվում է ZC-DSE ստատիկ վերլուծության գործիքը՝ որպես մուտք փոխանցելով հայտնաբերված ճանապարհները և համապատասխան մուտքային տվյալները, որոնց միջոցով հայտնաբերվել են այդ ճանապարհները: Գործիքն աշխատանքի արդյունքում վերադարձնում է մուտքային տվյալի առանձին հատվածների ամրագրված արժեքների մասին տեղեկություն: Փոխանցվող ինֆորմացիայի անբավարար լինելու պատճառով միշտ չէ, որ հնարավոր է հայտնաբերել նման արժեքներ: Այս խնդիրը լուծվում է նպատակային ծրագրի հետագա մեկնարկների միջոցով, երբ վերլուծության արդյունքում հայտնաբերվում են նոր ճանապարհներ:

Երկրորդ քայլում մուտքային տվյալների գեներացման հավելվածը, օգտագործելով ամրագրված արժեքները, ձևափոխում է միայն այն հատվածները, որոնց համար նախորդ քայլում չի հայտնաբերվել որևէ տեղեկատվություն:

Երկրորդ բաժնում նկարագրված է ZC-DSE (Zero-Cost Dynamic Symbolic Execution) ստատիկ վերլուծության գործիքը և դրա վերլուծության արդյունքների կիրառմամբ մուտքային տվյալների գեներացման եղանակը:

ZC-DSE գործիքը մուտքում ստանում է ֆազզ-թեստավորման ընթացքում հայտնաբերված ճանապարհները և համապատասխան մուտքային տվյալները: Գործիքի նպատակն է պարզել կատարված բազային բլոկների կախվածությունը մուտքային տվյալներից: Այդ նպատակով մշակվել են երկու ալգորիթմներ, որոնցից առաջինը խմբավորում է մուտքային տվյալները ըստ նրանց համապատասխանող կատարման հետագծում առկա բազային բլոկների, իսկ երկրորդը ստեղծում է կապ այդ բազային բլոկների և մուտքային տվյալի առանձին հատվածների միջև:

Առաջին ալգորիթմը բաղկացած է հետևյալ չորս քայլերից.

Առաջին քայլում դիտարկում ենք ծրագրի կատարման հերթական ճանապարհը:

Երկրորդ քայլում դիտարկում ենք հետագծում «հետաքրքիր» բազային բլոկը: Որպեսզի բազային բլոկը համարվի «հետաքրքիր» այն պետք է բավարարի հետևյալ չափորոշիչներից մեկին.

- Most common: B բազային բլոկը կհամարվի «հետաքրքիր», եթե հայտնաբերված ճանապարհների առնվազն P տոկոս կազմող ճանապարհները պարունակում են այդ բազային բլոկը: Այս չափորոշչի հիմքում ընկած է այն դատողությունը, որ առավել հաճախ կատարվող

բազային բլոկը համապատասխանում է ծառայողական տվյալների ստուգմանը:

- All: Բոլոր բազային բլոկները համարվում են «հետաքրքիր»:
- Custom: Հատուկ կոնֆիգուրացիոն ֆայլի միջոցով օգտատերը կարող է սահմանել սեփական չափորոշիչը:

Երրորդ քայլում, եթե տվյալ բազային բլոկը դեռ չի դիտարկվել, պահում ենք այն և այն մուտքային տվյալը, որը համապատասխանում է ընթացիկ հետագծին, իսկ եթե արդեն դիտարկվել է, ապա ընթացիկ հետագծին համապատասխանող մուտքային տվյալն ավելացնում ենք բազային բլոկին՝ համապատասխան մուտքային տվյալների բազմության մեջ:

Չորրորդ քայլում, եթե ընթացիկ հետագծում կա չդիտարկված բազային բլոկ, վերադառնում ենք երկրորդ քայլին, հակառակ դեպքում՝ վերադառնում ենք առաջին քայլին:

Ալգորիթմի ելքում ստանում ենք յուրաքանչյուր բազային բլոկի համար տվյալ բլոկին համապատասխանող մուտքային տվյալների բազմությունը: Երկրորդ ալգորիթմում դիտարկում ենք այդ բազմությունները: Եթե այն պարունակում է մեկից ավել մուտքային տվյալ, կատարվում է ըստ բայթերի հատում, որի արդյունքում առաջացած ոչ դատարկ բազմությունը պահվում է որպես «հետաքրքիր» բազային բլոկների կատարման համար անհրաժեշտ ամրագրված արժեքներ:

Գործիքի ավարտին, որպես աշխատանքի արդյունք, վերադարձվում է json[18] ֆորմատի ֆայլ, որում նկարագրված են բոլոր «հետաքրքիր» բազային բլոկները և դրանց կատարման համար անհրաժեշտ ամրագրված արժեքներ: Մուտքային տվյալների գեներացման ժամանակ ամրագրվում են հայտնաբերված արժեքները և մնում անփոփոխ ամբողջ ձևափոխության ընթացքում:

Երրորդ բաժինը ներառում է եզրակացություն և առաջարկվող մեթոդի միջոցով ձեռք բերված արդյունքները:

Առաջարկվող մեթոդը թույլ է տալիս կառուցել բազային բլոկների և մուտքային տվյալների հատվածների միջև կապեր, որոնք գեներացնելիս այդ կապերը հաշվի առնելը թույլ է տալիս ձևափոխել միայն այն միջակայքերը, որոնք կապված չեն որևէ բազային բլոկի հետ:

Առաջարկվող մեթոդի արդյունավետությունը ստուգելու համար օգտագործվել է ատենախոսության շրջանակներում նախագծված և իրականացված ֆազզ-

[18] JSON ֆորմատի WEB էջ - https://www.json.org/

թեստավորման միջավայրը: Ընտրվել են մի շարք իրական ծրագրեր, որոնք նախ թեստավորվել են միջավայրի միջոցով (առանց հավելյալ գործիքի), ապա նաև գործիքի և առաջարկվող մեթոդի համադրությամբ:

Ծրագրի անուն	ISP - Fuzzer		ISP - Fuzzer + ZC-DSE		Տարբերություն	
	Ճանապարհներ	Սխալներ	Ճանապարհներ	Սխալներ	Ճանապարհներ	Սխալներ
readelf	1175	0	2081	1	+906	+1
djpeg	48	0	45	1	-3	+1
objdump	391	0	407	0	+16	0
ar	11	0	9	0	-2	0
latex	80	0	196	1	+116	+1
optipng	465	34	504	41	+39	+7
gif2png	309	10	366	37	+57	+27
tiff2ps	166	0	187	0	+21	0
tiff2bw	54	0	63	0	+9	0
tiff2pdf	88	0	109	0	+21	0
tiff2rgba	37	0	44	0	+7	0
jasper	4	0	4	0	0	0
zipclock	91	0	108	1	+17	+1
dvi2tty	391	0	459	0	+68	0
strings	7	0	71	0	+64	0
pdftk	9	0	11	0	+2	0

Աղյուսակ 2. Առաջարկվող մեթոդի արդյունավետությունը

Ատենախոսության հիմնական արդյունքները հրապարակված են հետևյալ գիտական հոդվածներում.

1. Sargsyan S., Hakobyan J., Mehrabyan M., Mishechkin M., Akozin V., Kurmangaleev Sh., ISP-Fuzzer: Extendable fuzzing framework. Proceedings of Ivannikov Memorial Workshop, Velikiy Novgorod, Sep 2019, 68 - 71 pp.

2. Sargsyan S.S., Vardanyan V.G., Hakobyan J.A., Aghabalyan A.M., Mehrabyan M.S., Kurmangaleev Sh.F., Gerasimov A.Yu., Ermakov M.K., Vartanov S.P. Automatic API fuzzing framework. Proceedings of the Institute for System Programming, vol. 32, issue 2, 2020, pp. 161-174 DOI: 10.15514/ISPRAS-2020-32(2)-13.

3. Sargsyan S. S., Hakobyan J. A., Movsisyan H. M., Mehrabyan M. S., Sirunyan V. T., Kurmangaleev Sh. F., Improving fuzzing performance by applying interval mutations. Proceedings of ISP RAS 2019, volume 31, issue 5, 78 - 88 pp.

4. Hakobyan J., "Design and develop methods for fuzz testing improvement," ВЕСТНИК РАУ, pp. 117-124, 2021

Резюме

Акопян Дживан Андраникович

Исследование и разработка методов позволяющих повысить эффективность фаззинга

Программные обеспечения стали неотъемлемой частью жизни человека и с каждым годом увеличивается их роль. Часто, такие системы содержат программные дефекты и уязвимости, которые могут нанести не только финансовый ущерб, но и могут поставить под угрозу жизнь и здоровье человека. Существует множество примеров уязвимостей, к примеру, WannaCry кибератака и крушение ракеты Ariane 5.

Для обнаружения ошибок используются различные методы тестирования и инструменты анализа. Одним из таких методов считается фазз тестирование. Во время фазз тестирования генерируются входные данные и запускается целевая программа с этими данными.

После запуска отслеживается поведение программы, в случае аварийного завершения генерируется сообщение об обнаруженной ошибке.

Существующие инструменты фазз тестирования имеют ограниченное применение и предназначены для решения конкретной проблемы. Для тестирования сложных программ такие инструменты не применяются.

Как правило, используется комбинация нескольких инструментов, которая может привести к дополнительным затратам ресурсов и в итоге полностью может не решить проблему.

Системы интернет вещей (IoT) со временем получили широкое распространение. Одной из самых популярных систем считается Samsung SmartThings, в котором основная функциональность выполняется в виде Java библиотеки. А инструменты, которые предназначены для фазз тестирования библиотек, тестируют только отдельные функции и не рассматривают комбинации функций.

Почти все инструменты фаззинга не принимают во внимание структурные особенности входных данных целевой программы. Во время тестирования генерируются такие данные, которые в значительной степени игнорирует в

начале работы целевая программа, из-за некорректности данных. Это приводит к снижению эффективности тестирования.

Целью диссертации является разработка и реализация расширяемой системы фазз тестирования. Проведенные исследования и представленные разработки позволяют внедрить в эту систему новые методы, которые способствуют повысить эффективность фазз тестирования.

Результаты работы:

- Разработана расширяемая платформа для фазз тестирования. С помощью системы плагинов возможно вносить новые возможности. Платформу можно использовать как в Linux, так и в Windows и macOS ОС.
- Разработана метод тестирования Java библиотек позволяющий обеспечить комбинированные вызовы библиотечных функций.
- Разработан метод восстановления структуры входных данных.
- Разработан метод генерации входных данных основанный на интервальных мутациях входных данных.

Resume

Jivan Hakobyan

Design and develop methods to improve fuzz testing

The software has become an integral part of human life and continues increasing its role. Such systems contain software defects and vulnerabilities that can cause not only financial damage but also endanger human life and health. Examples of vulnerabilities are numerous, such as the WannaCry cyberattack and the crash of the Ariane 5 rocket.

Various testing methods and analysis tools are used to detect defects. One of these methods is fuzz testing. During fuzz testing, input data is generated, and the target program is launched with this data. The program's behavior is monitored during its run and in case of an abnormal termination defect information is reported.

The existing fuzz testing tools have limitations and are designed to solve a specific problem. Such tools are not appropriate for complex programs testing. Usually, it is required to use a combination of several tools, which may require additional resources and most importantly will not completely solve the problem.

Internet of Things (IoT) systems are becoming widely used. One of the most popular and widespread alike/similar systems is Samsung SmartThings. Here the main functionality is performed in the form of a Java library. Existing tools, designed for fuzz testing of libraries, test only individual functions and do not consider combinations of functions.

Almost all fuzzing tools do not take into account the structural features of the input data required by the target program. During testing, incorrectly generated data is mostly ignored by the target program, which decreases fuzzing efficiency.

The purpose of the dissertation is to develop and implement an extensible fuzz testing system, as well as research, develop and introduce new methods in that system to improve the efficiency of fuzz testing.

Main results of the work

- Designed and developed an extensible platform for fuzz testing. New features can be added using the plugin system. The platform can be used in Linux, Windows, and macOS operating systems.
- Designed and developed a method for testing of several function-call combinations in Java libraries.
- Designed and developed a method for target program's input data structure restoration.
- Designed and developed a method for input data generation based on interval mutation.