Утврђивање сличности софтверског кода

Захарије Радивојевић, Милош Цветановић

***Апстракт*—Утврђивање сличности софтверског кода представља област истраживања у оквиру софтверског инжењерства. Поред великог броја домена у којима налази примену оно представља кључни елемент за утврђивање постојања софтверских клонова, а самим тим утиче на софтвер током читавог његовог животног циклуса, током дизајна, развоја и одржавања. У овом раду је, на основу искустава аутора, дат преглед домена и карактеристике кодова који се у датим доменима пореде. Описане су и технике које се примењују у овој области независно од домена примене, као и карактеристике кодова које се датим техникама пореде. Такође је изложено пет сукцесивних поступака које су аутори развили за примену у домену откривања неовлашћеног коришћења лиценци. Поступци обухватају описе техника за утврђивање сличности бинарног кода, утицаја архитектуре рачунара на утврђивање сличности бинарног кода, утицаја трансформација преводиоца на утврђивање сличности бинарног кода, могућности за коришћење неуралних мрежа за утврђивање сличности бинарног кода, као и коришћење секвенци операција за утврђивање сличности бинарног кода.**

***Кључне речи*—Софтверски клонови, сличност кода, бинарни код, софтверске метрике, неуралне мреже.**

I. Увод

Анализа изворног кода програма има широки спектар примена, које укључују од рангирања и категоризације програма [1], преко откривања преписивања у домаћим задацима [2], до откривања недостатака у програмима [3]. У овој анализи као један од важних корака јесте утврђивање сличности делова кода. Постоје домени примене код којих је утврђивање сличности кода од пресудног значаја. Ови домени обухватају откривање злонамерног кода [4], реверзно инжењерство софтверских закрпа [5], откривање неовлашћеног коришћења лиценци [6], и слично. Ова анализа се у већини случајева обавља над изворним кодом програма, али се у неким случајевима може обављати и над асемблерским кодом програмима као и директно над бинарним записом извршног кода програма. Без обзира да ли се ради анализа изворног или извршног кода, као и без обзира на домен примене поступак се заснива на откривању софтверских клонова.

Овај рад има за циљ да представи област истраживања која се бави анализом сличности софтверског кода. Рад даје сажетак истраживања у овој области и нема за циљ да да систематичан преглед литературе у овој области. Рад је организован тако да ће у другој глави бити представљени домени примене утврђивања сличности кода. У трећој глави објашњене постојеће технике утврђивања сличности кода, док ће у четвртој глави бити представљени резултати истраживања у области анализе софтверског кода датог у бинарном облику које су аутори спровели током више година. Последња, пета, глава представља закључак овог рада.

Захарије Радивојевић, доктор електротехнике и рачунарства – Електротехнички факултет, Универзитет у Београду, Булевар краља Александра 73, 11000 Београд, Србија (e-mail: zaki@etf.bg.ac.rs).

Милош Цветановић, доктор електротехнике и рачунарства – Електротехнички факултет, Универзитет у Београду, Булевар краља Александра 73, 11000 Београд, Србија (e-mail: cmilos@etf.bg.ac.rs).

II. Домени примене одређивања сличности кода

У овој глави ће бити описани карактеристични домени у којима се може применити утврђивање сличности софтверског кода. За сваки од домена биће наведена сврха са којом се утврђивање сличности користи. Такође, за сваки од домена биће наведен и угледни примерак алата и описан његов начин функционисања. Преглед основних карактеристика појединих домена је дат у табели 1.

A. Детекција клонова

У домену детекције клонова упоређивање сличности кода се користи са циљем упаривања делова кода који могу потицати од сличног изворног кода или од сличног бинарног кода који потиче од истог изворног кода. Два дела кода која представљају софтверски клон поседују исту семантику, али се разликују по нивоу синтаксне сличности и на основу ње су класификовани у четири типа. Два синтаксно идентична кода се класификују као клонови типа 1, а могу се разликовати само по форматирању и пратећим коментарима. Уколико између два кода постоји и разлика у називима литерала онда се класификују као клонови типа 2. Убацивање малог броја додатних инструкција или промена редоследа инструкција која не утиче на резултат извршавања кода се класификује као клон типа 3. Све остале промене које чувају семантику али у већој мери мењају синтаксу оригиналног кода се класификују као клонови типа 4 [7].

Један од алата који се може користити у поступку откивања софтверских клонова је ACD алат [8]. Овај алат покушава да упари секвенце инструкција највеће могуће дужине. За упаривање инструкција потребно је да редослед упарених инструкција буде идентичан у оба кода који се упарују и да се утврди да се инструкције нису раније упариле. Као додатни услов се користи да је циљна адреса скокова иста у односу на остале упарене инструкције. Ако се током упаривања секвенци деси да се мали број инструкција не може упарити оне се игноришу. За сваке две упарене секвенце израчунава се тежина. Сваки пут, када су две инструкције упарене, тежина се

 ISBN 978-86-7466-930-3

ТАБЕЛА 1. ПРЕГЛЕД ДОМЕНА ПРИМЕНЕ

Домен	Тип кода	Тип клона	Циљ потраге	Узрок разлика
Детекција клонова	Изворни	1, 2, 3, 4	Сличне процедуре	Програмер
Откривање неовлашћеног коришћења лиценци	Изворни/Бинарни	1-Изворни, 4-Бинарни	Еквивалентне процедуре	Преводилац
Откривање злонамерног кода	Бинарни	1, 2, 3	Сличан код	Програмер/Преводилац
Откривање рањивости кода	Бинарни	1, 2, 3	Сличан код	Програмер

повећава за одређену вредност. Када се у некој од секвенци наиђе на инструкцију која се занемарује, да би се наставило упаривање, тежина се умањује. Процес упаривања две секвенце почиње са две инструкције које се могу упарити и наставља се све док тежина претходно упарених секвенци не буде једнака нули. Након тога, покушава се са побољшањем упаривања две секвенце откривањем нових инструкција које се могу упарити унутар ових секвенци а које би могле повећати укупан број упарених инструкција.

B. Откривање неовлашћеног коришћења лиценци

Све већи број кршења ауторских права у софтверској индустрији доводи до огромних економских губитака за носиоце ауторских права [9]. Једна од ситуација у којој може доћи до кршења је лиценцирање софтвера са двоструком лиценцом. Овај тип пословног модела се обично користи да би се подржао софтвер отвореног кода у комерцијалне сврхе. У таквом пословном моделу, власник ауторских права софтвера нуди изворни код бесплатно за некомерцијалну употребу, али остварује профит продајом ауторских права компанијама које желе да користе изворни код у својим производима. Повреда настаје када се изворни код власника користи у производу без лиценце добијене од власника. За разлику од домена откривања клонова, где је изворни код углавном доступан, одређивање сличности кода се у домену кршења лиценци примењује углавном над извршним кодом. Разлике које у извршним кодовима постоје последице су употребе различитих преводиоца над истим изворним кодом.

Један од алата који се може користити у поступку откивања кршења лиценце је BAT алат који је заснован на приступу описаном у раду [10]. Приступ користи три технике за ублажавање негативних ефеката потенцијалних разлика између кодова који се пореде. Прва техника прикупља стринг литерале који се појављују у коду који се пореди. Након тога, покушава да пронађе исте литерале у коду за који се сумња да крши лиценцу. Друга техника претпоставља да ће алгоритам компресије података успети да компримује боље два кода заједно ако међу тим кодовима постоје исте секвенце инструкција. Успех се мери односом између величине архиве, која се састоји од два кода, и укупне величине архива које се појединачно састоје од само једног од два кода које се пореде. Што је однос мањи, то се очекује већа сличност између упоређених кодова. Трећа техника се заснива на израчунавању бинарних разлика између кодова. Што су разлике мање, вероватноћа да посматрани код крши лиценцу је већа. Алат BAT користи се за откривање да ли посматрани код крши лиценцу неког од кодова из репозиторијума, а за ту сврху користи само прву технику.

C. Откривање злонамерног кода

Злонамерни код је сваки код који има могућност да оштети било који рачунарски систем [4]. Количина злонамерног кода се сваке године све брже повећава и представља озбиљну безбедносну претњу. Отуда је откривање злонамерног кода критична тема у рачунарској безбедности. У оквиру комерцијалних антивирусних програма откривање злонамерног кода је засновано на поређењима отисака метода са познатим отисцима метода. Недостатак овог начина откривања злонамерног кода је да не успева да открије нове варијације већ познатог злонамерног кода. У циљу превазилажења тог недостатка може се употребити одређивање сличности злонамерног кода са циљем откривања дела кода који показује слично понашање као што је понашање које има код из већ познате колекције злонамерног кода.

Један од алата који се може користити у поступку откивања злонамерног кода који користи одређивање сличност кодова је алат који упоређује секвенце кодова операција [4]. Током поређења посматрају се низови кодова операција фиксних дужина и узима се у обзир да различити кодови операција имају различиту релевантност током поређења. Код овог алата се најпре обавља анализа великог броја злонамерног кода и регуларног кода и утврђује се релевантност кодова операције. Затим се израчунавају фреквенције појављивања свих секвенци изабране дужине у кодовима који се упоређују. За сваки од кодова формира се вектор чије су компоненте бројеви појављивања појединачних секвенци. Свака од компоненти вектора се затим множи са производом релевантности свих кодова операција који се појављују у посматраној секвенци како би се елиминисао шум који уноси ирелевантан код. На крају сличност између кодова који се пореде се израчунава коришћењем косинусне сличности ових вектора. Приступ такође предлаже комбинацију добијених резултата за неколико различитих дужина посматраних секвенци.

 ISBN 978-86-7466-930-3

D. Откривање рањивости кода

У домену рањивости софтверског кода могу се разматрати анализа рањивости и откривање рањивости. Анализа рањивости има за циљ да формалним методама потврди или оповргне хипотезу да је софтвер рањив. Приступи анализи рањивости се могу класификовати у три категорије: статичка анализа, динамичка анализа и хибридна анализа кода. Откривање рањивости има за циљ да мање формалним поступцима лоцира конкретну рањивост у коду. У овој области постоји више различитих приступа који између осталог обухватају: тестове пенетрације софтвера, насумично тестирање, и статичку анализу токова података [11]. За разлику од претходно разматраних домена у овом домену се одређивање сличности софтверског кода користи за откривања разлика између две различите верзије истог кода.

Један од алата који се може користити за потребе откривања рањивости је алат који покушава да упари делове кода односно процедура које су скоро идентичне [12]. Први предуслов за упаривање процедура је да су идентичне према свим датим критеријумима (тзв. селекторима): броју основних блокова, броју ивица у контролном графу тока и броју позваних потпрограма. Други предуслов за упаривање је да не постоји друга процедура која је идентична са процедурама које се пореде на основу коришћених критеријумима. Да би се смањило ограничење другог предуслова у свим корацима алгоритма, посматрају се подскупови процедура из кодова који се пореде а који су дати у бинарном облику. Приликом формирања првих парова, подскупови се формирају издвајањем процедура које задовољавају неку карактеристику. Неке од карактеристика које се користе су број улазних и излазних грана процедура у графу позива, исти називи процедура, референце на исте стрингове и број пута када су се неке од инструкција појавиле. Када даље упаривање по датим карактеристикама више није могуће, формирају се нови подскупови процедура које се позивају из упарених процедура. Подскупове такође формирају процедуре које позивају упарене процедуре. Алгоритам се понавља за сваке две упарене процедуре све до тренутка када даље упаривање више није могуће. Унутар упарених процедура врши се даље упаривање основних блокова и инструкција.

III. Технике утврђивања сличности кода

У овој глави ће бити описане најзаступљеније технике утврђивања сличности софтверског кода описане у прегледним радовима [7], [13], [14], [15], [16], [17]. За сваку од техника биће наведен кратак опис, применљивост, могућности проширења, као и на које типове софтверских клонова се најчешће примењује. Такође, биће наведен и угледни примерак алата и описане његов начин функционисања за сваку од техника. Поред техника које су описане у наставку постоје и хибридне технике код којих се може јавити синергистички ефекат тако да могу детектовати додатне типове клонова у поређењу са типовима клонова који се могу детектовати било којом од коришћених техника детекције одвојено.

A. Технике засноване на поређењу текста

Технике засноване на поређењу текста посматрају секвенцу линија изворног кода. Како би се пронашла секвенца истоветних линија, стрингова или лексема пореде се делови два изворна кода или њихови отисци. Када се у датим деловима кода открије да су поједини делови слични они се проглашавају за клонове одређене класе. Ове технике се углавном користе за откривање клонова код виших програмских језика и развијени алати често подржавају више од једног програмског језика. У случају да их је потребно проширити тако да подрже неки нови програмски језик или није потребно ништа додатно имплементирати или је потребно имплементирати лексички анализатор. Ова техника се углавном може применити за откривање клонова типа 1 и типа 3, а може се применити и за откривање клонова типа 2. Ова техника се у имплементацијама показала као техника средње сложености.

Један од алата који користи технику засновану на поређењу текста је SimCad алат [18]. Овај алат подржава више програмских језика и то: C, C#, Java, Python. Процес откривања клонова се заснива на техникама откривања скривеног знања и рударења података и користи алгоритам за груписање података као и претрагу података засновану на вишенивојским индексима. Овај алат извршава три фазе обраде: прелиминарну обраду, откривање клонова и генерисање резултата. У фази прелиминарне обраде користећи Simhash алгоритам се генеришу отисци кода, након чега се обавља индексирање формирањем вишенивојски индекси за потребе брже претраге. У другој фази се обавља откривање клонова тако што се фрагменти кода групишу у кластере користећи Хемингово растојање између генерисаних отисака који одговарају тим фрагментима кода. Парови фрагмената који не прелазе одговарајући ниво сличности бивају елиминисани из кластера, а такође мора постојати и одређен број фрагмената кода у кластеру. У трећој фази се врши чишћење добијених резултата као и њихов приказ у одговарајућем формату.

B. Технике засноване на поређењу токена

Технике засноване на поређењу токена посматрају секвенцу токена издвојених из изворног кода. Издвајање токена се обавља у поступку лексичке анализе изворног кода. Секвенца токена се формира као скуп токена на одређеном нивоу гранулације. Срж ове технике представља поређење токена који припадају суфиксним стаблима или суфиксним низовима састављених од секвенци токена. Ова техника откривања користи посебан објекат који представља апстракцију конкретних вредности идентификатора и литерала и који води рачуна о очувању њиховог међусобног редоследа. Ове технике се углавном користе за откривање клонова код виших

 ISBN 978-86-7466-930-3

ТАБЕЛА 2. ПРЕГЛЕД ТЕХНИКА

Техника	Тип кода	Тип клона	Проширивост	Сложеност извршавања
Поређење текста	Изворни	1, 2, 3	-/Лексички	Средња
Поређење токена	Изворни	1, 2, 3	Лексички	Мања
Поређење метрика	Изворни/Бинарни	1, 2, 3, 4	Лексички, парсер	Средња
Поређење апстрактног синтаксног стабла	Изворни/Бинарни	1, 2, 3	Парсер	Средња
Поређење графа зависности	Изворни/Бинарни	1, 2, 3, 4	Парсер	Велика

програмских језика, али је за разлику од техника заснованих на поређењу текста зависност од програмског језика већа. У случају да је потребно проширити их подршком за нови програмски језик неопходно је имплементирати лексички анализатор. Ова техника се углавном може применити за препознавање клонова типа 1, 2 и 3, и представља једну од најцитиранијих техника за одређивање сличности. Ова техника се у имплементацијама показала као техника мање сложености.

Један од алата који користи технику засновану на поређењу текста је CCFinder алат [19]. Овај алат подржава више програмских језика и то: C/C++, C#, Cobol, Java, VB. Обрада коју спроводи овај алат се извршава у четири фазе: лексичка анализа, трансформација међурезултата, одређивање сличности и форматирање добијених резултата. У фази лексичке анализе се свака линија изворног кода дели у низ токена у складу са лексичким правилима датог програмског језика. У другој фази се обавља трансформација добијених низова токена користећи правила трансформације и правила замене одређених типова токена. У трећој фази се обавља упоређивање свих подстрингова како би се идентификовали парови клонова. На крају, у четвртој фази, се препознати клонови пресликавају на линије изворног кода из кога потичу.

C. Технике засноване на метрикама

Технике засноване на метрикама посматрају карактеристике изворног програма издвојене користећи скуп метрика. Издвајање карактеристика се обавља полазећи од кода који је трансформисан у одговарајућу структуру. Метрике се могу израчунавати на основу имена, размештаја елемента, израза, контроле токе функција, и сличних елемената структуре кода. Одређивање сличности се обавља израчунавањем удаљености између одговарајућих чланова у формираном метричком простору. Ове технике се углавном користе за откривање клонова код виших програмских језика и зависност од програмског језика је велика, али се такође могу примењивати и код нижих програмских језика. У случају да је потребно проширити их подршком за нови програмски језик неопходно је имплементирати не само лексички анализатор, већ и одговарајући парсер. Техника се може применити за препознавање свих типова клонова. Ова техника се у имплементацијама показала као техника средње сложености.

Један од алата који користи технику засновану на метрикама је CLAN алат [20]. Овај алат подржава програмске језике C/C++. Обрада коју спроводи овај алат се извршава у четири фазе: препроцесирање директива, парисирање и идентификација фрагмената, екстракција метрика и идентификација клонова. Како је алат намењен језицима C/C++ у првој фази се обавља препроцесирање директива датог језика како би се добио изворни код који се даље може обрађивати. У другој фази се обавља екстракција декларација и дефиниција самих функција користећи наменски парсер. У трећих фази се издвајају метрике коју укључују бројање функција, локалних променљивих, дељених и глобалних променљивих, параметара, исказа скокова и петљи. У четвртој фази се обавља идентификација клонова у поступку квантизација метрика коришћењем прагова за елиминацију шума и спектралне анализе резултата.

D. Технике засноване на апстрактним синтаксним стаблима

Технике засноване на апстрактним синтаксним стаблима трансформишу изворни код у структуру стабла које се касније може упоређивати. За поређење стабала, или њихових подстабала, се може користити више различитих метода које укључују хеширање, трансформацију у префиксна стабла, одређивање најдуже заједничке секвенце употребом динамичког програмирања. Ове технике се углавном користе за откривање клонова код виших програмских језика и зависност од програмског језика је велика, али се такође могу примењивати и код нижих програмских језика. У случају да је потребно проширити их подршком за нови програмски језик неопходно је имплементирати одговарајући парсер. Ова техника се углавном може применити за препознавање клонова типа 1, 2 и 3. Ова техника се у имплементацијама показала као техника средње сложености.

Један од алата који користи технику засновану на метрикама је DECKARD алат [21]. Овај алат подржава програмске језике: C, Java, Php. Обрада коју спроводи овај алат се извршава у пет фаза: генерисање парсера, формирање стабла, генерисање вектора, кластеровање вектора и додатна обрада. У првој фази се на основу формалне граматике језика генерише парсер који се користи и даљој обради. У другој фази се на основу генерисаног парсера и улазног изворног кода формира апстрактно синтаксо стабло које ће се даље обрађивати. У

 ISBN 978-86-7466-930-3

трећој фази се обрађује синтаксно стабло како би се формирали вектори карактеристика фиксне дужине. У овој фази се додатно може обавити и спајање делова који имају заједничко подстабло. У четвртој фази се обавља груписање вектора у кластере користећи еуклидско растојање са циљем одређивања клонова. На крају се као додатна, пета, фаза обавља додатна обрада како би се користећи одређене хеуристике елиминисали погрешно идентификовани клонови.

E. Технике засноване на коришћењу графа зависности

Технике засноване на коришћењу графа зависности које постоје у графу тока контроле и у графу тока података. Графовска репрезентација изворног кода се дели на мање делове у зависности од понашања фрагмента изворног кода. За деобу и поређење се може користити неки од постојећих алгоритама за рад са подстринговима и откривање сличности. Ове технике се углавном користе за откривање клонова код виших програмских језика и зависност од програмског језика је велика. У случају да је потребно проширити их подршком за нови програмски језик неопходно је имплементирати одговарајући парсер. Ова техника се, као и рад са метрикама, може применити за препознавање свих типова клонова. Ова техника се у имплементацијама показала као техника велике сложености.

Један од алата који користи технику засновану на коришћењу графа зависности је PDG-DUP алат [22]. Овај алат подржава програмске језике C/C++. Обрада коју спроводи овај алат се извршава у четири фаза: формирање графа зависности, проналажење парова клонова, уклањање обухваћених клонова и груписање клонова у веће групе. У првој фази се формира граф зависности код кога чворови представљају наредбе програма и предикате, а ивице представљају зависности података и контроле. Друга фаза се обавља у неколико корака. На почетку се обавља деоба свих чворова графа зависности у класе еквиваленције на основу синтаксне структуре исказа и предиката који ти чворови представљају, занемарујући имена променљивих и литерале. У наредном кораку се обавља најбитнији део, срж, алгоритма који проналази изоморфне подграфове коришћењем технике одсецања уназад (*backward slicing*) која датом програмском сегменту додаје претходни повезани део. Поред повратног одсецања користи се и техника одсецања унапред која додаје наредни повезани део. У трећој фази се обавља уклањање свих парова клонова који су већ садржани у неким другим паровима клонова идентификованим у претходном кораку. У четвртој фази се обавља груписање идентификованих парова у веће групе користећи особину транзитивности.

IV. Доприноси утврђивању сличности кода

Откривање неовлашћеног коришћења софтверске библиотеке је проблем детекције клонова који у случају комерцијалних производа има додатну сложеност због чињенице да је код доступан само у бинарном облику. Циљ аутора је био да предложи приступ за процену нивоа сличности између процедура који потичу из различитих бинарних кодова. Основна претпоставка у истраживањима је била да клонови у бинарним кодовима потичу од употребе заједничке софтверске библиотеке, односно истог изворног кода, која се може преводити користећи различите алате. За потребе истраживања бинарни код је дисасемблиран, а за потребе коришћења других алата и декомпајлиран. Детаљан преглед различитих приступа у препознавању сличности бинарног кода дат је у прегледном раду [17] док ће у овој глави бити описана истраживања аутора овога рада спроведена стремећи ка наведеном циљу у области утврђивања сличности бинарног кода. За свако од спроведених истраживања биће наведен кратак опис истраживања, кораци предложеног приступа а потом дати и остварени резултати. Преглед основних карактеристика појединих истраживања је дат у табели 3.

A. Утврђивање сличности бинарног кода

Прво истраживање је имало за циљ да утврди применљивост постојећих техника и алата описаних у претходним поглављима за утврђивање сличности бинарног кода [6]. Имајући у виду да су постојеће технике и алати превасходно намењени раду са вишим програмским језицима у овом истраживању су нека одабрана решења прилагођења раду са бинарним кодом ради евалуације. У истраживању је такође предложен нови приступ који је заснован на метрикама.

Предложени приступ за процену нивоа сличности између две процедуре које се пореде је процес који се обавља у четири фазе: издвајање метрика, упоређивање метрика, трансформације метрика, израчунавање нивоа сличности. У првој фази се спроводи статичка анализа процедура и издвајање метрика тако да се за сваку процедуру формира вектор метрика, при чему елементи тог вектора могу бити скаларне или векторске вредности. Ове метрике укључују одређивање броја и учестаности свих инструкција, појединих типова инструкција (аритметичких, логичких и трансфер података), скокова (условних и безусловних скокова, позива процедура и петљи) и које се представљају скаларним вредностима. Поред скаларних вредности метрика коришћене су и нескаларне, векторске, метрике која укључују бројање сваке појединачне инструкције, сваке адресе доскока, и сваке адресе позива процедуре. У другој фази се обавља поређење сваке о издвојених метрика користећи одговарајући функције за поређење. У трећој фази се обавља трансформација и нормализација добијених резултата поређења на основу претходног знања. У последњој, четвртој, фази комбинују се одабране метрике користећи одговарајућу формулу за формирање једне вредности која представља сличност између две процедуре.

Резултати истраживања су дали одговоре на три истраживачка питања и довела до следећих закључака. Прво питање се односило на разматрање додавања нове

 ISBN 978-86-7466-930-3

ТАБЕЛА 3. ПРЕГЛЕД РЕЗУЛТАТА ИСТРАЖИВАЊА

Приступ	Метрике	Архитектура	Мера сличности	Одзив
1.	Скаларна(13) векторска(6)	ARM	Формуле (7)	35,8%-73,1%
2.	Скаларна(3) векторска(1)	x86	Тежинска сума	37,6%-63.0%
3.	Скаларна(13) векторска(6)	ARM	Формуле (7)	52,7%-81,8%
4.	Векторска(1)	ARM	Неуралнa мрежа	49,2%-82,8%
5.	Векторска(1)	ARM	Левенштајнова удаљеност	57,9%-88,9%

метрике у скуп метрика предложних у постојећим решењима и утврђивање доприноса приликом рангирања процедура. Резултати спроведеног експеримента показују да увођење нових метрика доприноси рангирању процедура за све посматране позиције приликом рангирања. Штавише, са новим метрикама предложени приступ је постигао 1,44 пута бољи одзив (*recall*) за прву посматрану позицију у поређењу са случајем када се користе само постојеће метрике. Друго питање се односило на проверу да ли рангирање процедура обављено предложеним приступом зависи од преводиоца, нивоа оптимизације програма и контекста проблема. Према резултатима експеримента утврђено је да постоји значајан утицај изабраног преводиоца на рангирање процедура који је остварен у предложеном приступу. Међутим, контекст проблема и опције превођења имају мањи утицај на резултате. Треће питање се односило на проверу да ли предложени приступ постиже боље резултате од постојећих алата у смислу прецизности, одзива и F2 мере. Одговарајући експеримент је показао да предложени приступ постиже други најбољи резултат у смислу прецизности и најбољи резултат у смислу одзива. У случају посматрања F2 мере, предложени приступ постиже резултате боље од осталих алата када се посматра до првих шест позиција, док се максимална вредност постиже када се посматрају само прве две позиције при рангирању.

B. Утицај архитектуре рачунара на утврђивање сличности бинарног кода

Друго истраживање је имало за циљ да утврди применљивост приступа предложеног у првом истраживању на другу архитектуру [23]. С обзиром да је у првом истраживању коришћена ARM архитектура, за потребе другог истраживања је коришћена x86 архитектура. Имајући у виду разлике у архитектури било је потребно поновно креирање скупа података на коме ће се обавити тестирање. Овом приликом је коришћен сличан скуп програмских преводилаца, али не потпуно исти, јер ниси сви преводиоци подржавали обе архитектуре.

У оквиру другог истраживања је предложен нови приступ који представља подскуп приступа предложеног у првом истраживању. У овом новом приступу број различитих корака који су спровођени током испитивања у појединим фазама је редукован, и такође је редукован број фаза. Предложени приступ уместо четири фазе има само три фазе. У првој фази је смањен број издвојених метрика, тако да су коришћене метрике које одређују дужину процедуре, број скокова, број позиваних процедура и броје инструкција. Иако је смањен број метрика сачувано је постојање и скаларних и векторских метрика. У другој фази се обавља поређење сваке о издвојених метрика користећи одговарајући функције за поређење, на истоветан начин као и првом истраживању. Трећа фаза се може посматрати као обједињење треће и четврте фазе из првог истраживања и у њој се израчунава сличности користећи хармонијску средину, тежинску суму и наивни Бајес.

Резултати за x86 архитектуру показују да је најбољи одзив постигнут користећи приступ заснованим на тежинској суми. Одзив се креће од 37% до 63% када се посматра прва односно првих десет позиција. Поређење резултата остварених на x86 и ARM архитектурама показује да су постигнућа добијена за првих десет позиција на обе архитектуре упоредива.

Ово показује да се технике поређења бинарних кодова у извесној мери могу користити за поређење кодова који су преведени за исту архитектуру рачунара. Истраживања која су спровели други истраживачи након овог истраживања, почев од 2016. године а која се баве поређењем сличностима бинарног кода између архитектура рачунара [17], су показала да се технике за поређење могу примењивати не само на једној архитектури рачунара, већ се могу примењивати и на кодове код којих су једни преведени за једну архитектуру рачунара а други за други архитектуру рачунара. Овиме се проширује оно што је другим истраживањем утврђено а то је да се технике поређења бинарних кодова у извесној мери могу посматрати независно од архитектуре рачунара.

C. Утицај трансформација преводиоца на утврђивање сличности бинарног кода

Као што је претходно описано, резултати првог истраживања су показали да постоји значајан утицај изабраног преводиоца на рангирање процедура. Из тог разлога треће истраживање је имало за циљ да детаљније испита утицај трансформација програмског преводиоца на утврђивање сличности бинарног кода [24]. У истраживању је такође предложен и нови приступ који дефинише нове технике за смањење утицаја процеса превођења на перформансе поступка утврђивања сличности бинарног кода.

У односу на приступ предложен у првом истраживању у овом приступу је у првој фази, у којој се спроводи

 ISBN 978-86-7466-930-3

статичка анализа, додато пет техника које се спроводе као независни кораци. Прва техника предлаже игнорисање инструкција за рад са стеком. Ова техника је уведена је уведена зато што је приликом анализе уочено да различити преводиоци на различите начине интерагују са стеком и да инструкције које том приликом користе не носе информације о семантици кода. Друга техника предлаже игнорисање инструкција за трансфер података. Ова техника је уведена из сличног разлога као и прва техника јер различити преводиоци користе различите приступе приликом алокације регистара и трансфера између њих или меморијске локације на којој се неки податак налази. Трећа техника предлаже измену неких од разматраних векторских метрика. Овде се уместо посматрања статистике о појединим инструкцијама посматра статистика о секвенцама инструкција одређене дужине. Четврта техника предлаже уграђивање позваних процедура у оквиру позивајуће процедуре. Ова техника је уведена како би се елиминисала разлика између бинарних кодова која може настати усред одлуке преводиоца да обави аутоматско уграђивање у време превођења. Пета техника предлаже увођење прага сличности између процедура. Ова техника је уведена како би смањила број лажно позитивно идентификованих клонова, с обзиром да преводиоци могу различиту логику да имплементирају користећи сличне инструкције.

Резултати истраживања су дали одговоре на три истраживачка питања и довела до следећих закључака. Прво питање се односило на одређивање постигнућа постојећих решења зависе од тога да ли се користи произвољни преводилац и произвољни ниво оптимизације. Према резултатима експеримента утврђено је да постоји значајан утицај изабраног преводиоца на рангирање процедура, и за разлику од првог истраживања овом приликом су тачно измерени утицаји који имају промена преводиоца и промена нивоа оптимизације. Друго питање се односило на проверу да ли додавање сваке од предложених техника једне по једне основном приступу повећава постигнуће ако се упоређене процедуре преводе са произвољним преводиоцима и нивоима оптимизације. Према резултатима експеримента утврђено је да све технике, изузев друге технике које филтрира инструкције трансфера података, доприносе побољшању резултата утврђивања сличности. Треће питање се односило на проверу да ли додавање свих предложених техника заједно основном приступу повећава постигнуће и да ли резултира синергистичким ефектима ако се упоређене процедуре преводе са произвољним преводиоцима и нивоима оптимизације. Резултати експеримента су показали да додавање свих техника има синергистички ефекат који доноси 6,7% до 9,3% повећања одзива за првих пет посматраних позиција приликом рангирања.

D. Коришћење неуралних мрежа за утврђивање сличности бинарног кода

Четврто истраживање је имало за циљ да испита могућност употребе неуралних мрежа за утврђивање сличности бинарног кода [25]. За разлику од приступа описаном у претходним истраживањима који се заснивају на посматрању метрика у овом истраживању су посматране само инструкције. За потребе евалуације је искоришћен исти скуп података који је коришћен у првом и трећем истраживању.

Предложени приступ обавља поређење процедура у пет фаза: издвајање инструкција, кодирање инструкција, коришћење једног првог слоја неуралне мреже, коришћење другог слоја неураллне мреже, поређење. У првој фази се спроводи издвајање токена који одговарају појединим инструкцијама. У другој фази се обавља кодирање инструкција у целобројну вредност. У поступку кодирања су примењене три технике: лексикографско кодирање, семантичко мапирање и мапирање засновано на класи. У трећој фази се полазећи од секвенце кодираних инструкција формирана секвенца карактеристика исте дужине користећи неуралне мреже. Коришћена је Дуготрајно-краткотрајна меморија LSTM (*Long Short-Term Memory*). У четвртој фази се обавља издвајање одређеног броја карактеристика користећи неуралну мрежу која занемарује одређен број улазних карактеристика DNN (*Dropout Neural Network*). У последњој, петој, фази комбинују се рачунање сличности између две процедуре користећи издвојене карактеристике и поступак агрегације и нормализације.

Резултати истраживања су дали одговоре на три истраживачка питања и довели до следећих закључака. Прво питање се односило на разматрање какве резултате даје предложени поступак у односу на поређење процедура само по њиховој дужини. Резултати спроведеног експеримента показују да предложени приступ даје бољи одзив на свакој посматраној позицији. Друго питање се односило на проверу да ли рангирање процедура обављено предложеним приступом зависи од преводиоца. Према резултатима експеримента утврђено је да постоји неки утицај изабраног преводиоца на рангирање процедура од 0,55% до 1,64% када се посматра одзив. Овај утицај је знатно мањи у односу утицај који је присутан код првог и трећег приступа. Треће питање се односило на проверу да ли предложени приступ постиже боље резултате од постојећих алата у смислу одзива. Одговарајући експеримент је показао да предложени приступ постиже незнатно боље резултате, просечно 1,14%. Овај приступ је дао бољи одзив на вишим позицијама, док у првих 9 позиција даје лошије резултате у односу на трећи предложени приступ.

E. Коришћење секвенци операција за утврђивање сличности бинарног кода

У првом истраживању је примећено да метрика која узима у разматрање секвенцу инструкција највише доприноси успешном одређивању сличности. Из тог разлога пето истраживање је имало за циљ да детаљније испита могућности те метрике и да је учини робуснијом на промене које чини преводилац. Сходно томе

 ISBN 978-86-7466-930-3

предложен је нови приступ заснован на испитивању секвенце кодова операција [26].

Предложени приступ обавља поређење процедура у четири фаза: издвајање инструкција, кодирање инструкција, одређивање Левенштајнове удаљености, рачунање релативне Левенштајнове удаљености. У првој фази се спроводи издвајање токена који одговарају појединим инструкцијама. У другој фази се обавља кодирање инструкција у целобројну вредност на истоветан начин ономе описаном у четвртом приступу. У трећој фази се обавља рачунање Левенштајнове удаљености користећи *Needleman–Wunsch* алгоритам. У последњој, четвртој, фази се коришћењем релативне Левенштајнове удаљености добија сличности између две процедуре.

Ово истраживање је требало да да одговор на једно истраживачко питање о томе какве резултате даје предложени приступ у односу на остале приступе које су аутори развили за ARM архитектуру. Резултати спроведеног експеримента показују да предложени приступ даје бољи одзив на свакој посматраној позицији од свих претходно разматраних приступа. У односу на приступ из четвртог истраживања даје од 3,59% до 9,45% бољи одзив, као и од 0,17% до 4,73% бољу прецизност када се посматрају првих 20 позиција. У односу на приступ из трећег истраживања даје од 3,45% до 9,53% бољи одзив, као и од 0,45% до 1,72% бољу прецизност када се посматрају првих 20 позиција.

V. Закључак

Утврђивање сличности кода представља област истраживања са дугом историјом, доста запажених резултата али и перспективом за даљи развој у будућности. Овај рад је имао за циљ да ову област приближи истраживачима и да да осврт на истраживања која су аутори спровели. У раду је најпре објашњен значај и домени у којима се резултати истраживања примењују. Након тога је дат сажет преглед најважнијих техника у овој области. На крају су изложена истраживања и резултати до којих су аутори дошли у претходним годинама.

Захвалница

Рад на овом пројекту је делимично био финансиран од стране Министарства просвете, науке и технолошког развоја Републике Србије (2022/200103), као и Фонда за науку Републике Србије (АВАНТЕС).

Референце

[1] L. Li, T. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, Y. Le Traon, “Static analysis of android apps: A systematic literature review,” *Information and Software Technology*, vol. 88, pp. 67-95, Aug. 2017, 10.1016/j.infsof.2017.04.001.

[2] M. Misic, Z. Sustran, and J. Protic, “A comparison of software tools for plagiarism detection in programming assignments,” *International Journal of Engineering Education*, vol. 32, no. 2, pp. 738-748, 2016.

[3] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. di Penta, “How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines,” in IEEE International Working Conference on Mining Software Repositories, 2017, doi: 10.1109/MSR.2017.2.

[4] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, “Idea: Opcode-Sequence-Based Malware Detection,” In: F. Massacci, D. Wallach, N. Zannone, (eds) Engineering Secure Software and Systems. ESSoS 2010. Lecture Notes in Computer Science, vol 5965. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-11747-3_3.

[5] T. Dullien and R. Rolles, “Graph-based comparison of executable objects (english version),” Sstic, 2005, doi: 10.1.1.96.5076.

[6] S. Stojanović, Z. Radivojević, and M. Cvetanović, “Approach for estimating similarity between procedures in differently compiled binaries,” *Information and Software Technology*, vol. 58, pp. 259-271, 2015, doi: 10.1016/j.infsof.2014.06.012.

[7] C. K. Roy, J. R. Cordy, R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: a qualitative approach,” *Sci. Comput. Program*, vol. 74, pp. 470–495, 2009.

[8] Davis, I.J. and Godfrey, M.W, “From Whence it Came: Detecting Source Code Clones by Analyzing Assembler,” Proc. WCRE 10, Beverly, MA, October 13–16, pp. 242–246. IEEE, Los Alamitos, 2010.

[9] P.E.Chaudhry, and A. Zimmerman “Protecting Your Intellectual Property Rights: Understanding the Role of Management, Governments, Consumers and Pirates,” Springer, New York, 2012.

[10] A. Hemel, K.T. Kalleberg, R. Vermaas, and E. Dolstra, “Finding Software License Violations through Binary Code Clone Detection,” Proc. MSR 11, Honolulu, HI, May 21–22, pp. 63–72. ACM, New York, 2011.

[11] S. M. Ghaffarian and H. R. Shahriari. “Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey,” ACM Comput. Surv. 50, 4, Article 56 (July 2018), 36 pages. DOI:https://doi.org/10.1145/3092566

[12] T. Dullien, and R. Rolf, “Graph-based Comparison of Executable Objects (English version),” Proc. SSTIC 05, Rennes, France, June 1–3, pp. 1–13. STIC, Paris, 2005.

[13] D. Rattan, R. Bhatia, and M. Singh, “Software clone detection: a systematic review,” *Inform. Softw. Technol.*, vol. 55, pp.1165–1199, 2013.

[14] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, “Comparison and evaluation of clone detection tools,” *IEEE Trans. Software Eng*. vol. 33, pp. 577–591, 2007.

[15] C. K. Roy, J. R. Cordy, “A Survey on Software Clone Detection Research,” Technical Report 2007-541, Queen’s University, Canada, 2007.

[16] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam and B. Maqbool, “A Systematic Review on Code Clone Detection,” *IEEE Access,* vol. 7, pp. 86121-86144, 2019, doi: 10.1109/ACCESS.2019.2918202.

[17] I. Haq, J. Caballero, “A Survey of Binary Code Similarity,” *ACM Computing Surveys,* vol. 54, no. 3, pp. 1-38, April, 2022.

[18] M. S. Uddin, C.K. Roy, K.A. Schneider, A Hindle, “On the effectiveness of Simhash for detecting near-miss clones in large scale software systems,” in: Proceedings of 18th Working Conference on Reverse Engineering (WCRE 2011), Limerick, Ireland, 2011, pp. 13–22.

[19] T. Kamiya, S. Kusumoto, K. Inoue, “CCFinder: a multilinguistic token-based code clone detection system for large scale source code,” *IEEE Trans. Software Eng*., vol. 28, pp. 654–670, 2002.

[20] E. Merlo, “Detection of plagiarism in university projects using metrics-based spectral similarity,” in: Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, Dagstuhl, Germany, 2006, pp. 1–10.

[21] L. Jiang, G. Misherghi, Z. Su, S. Glondu, “DECKARD: scalable and accurate treebased detection of code clones,” in: Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, USA, 2007, pp. 96–105.

[22] R. Komondoor, S. Horwitz, “Using slicing to identify duplication in source code,” in: Proceedings of the 8th International Symposium on Static Analysis (SAS’01), vol. LNCS 2126, Paris, France, 2001, pp. 40–56.

[23] K. Berta, S. Stojanović, M. Cvetanović, Z. Radivojević, “Estimation of Similarity between Functions Extracted from x86 Executable Files,” *Serbian Journal of Electrical Engineering*, vol. 12, no. 2, pp. 253 - 262, Jun, 2015.

 ISBN 978-86-7466-930-3

[24] Z. Radivojević, M. Cvetanović and S. Stojanović, "Comparison of Binary Procedures: A Set of Techniques for Evading Compiler Transformations," *The Computer Journal*, vol. 59, pp. 106–118, 2015.
[25] N. Pejić, M. Cvetanović, Z. Radivojević, "Estimating similarity between differently compiled procedures using neural networks," ТЕЛФОР XXVII, Belgrade, Nov, 2019.
[26] N. Pejić, M. Cvetanović, Z. Radivojević, "Comparing Assembler Procedures by Analyzing Sequences of Opcodes," Telfor Journal, vol. 12, no. 1, pp. 46 - 49, Jul, 2020.

 ISBN 978-86-7466-930-3