Kauno technologijos universitetas

Informatikos fakultetas

Programos kodo užtemdymo metodas

Baigiamasis magistro projektas

Karolis Pavydis

Projekto autorius

prof. Jevgenijus Toldinas

Vadovas

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Programos kodo užtemdymo metodas

Baigiamasis magistro projektas

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

Karolis Pavydis

Projekto autorius

Prof. Jevgenijus Toldinas

Vadovas

doc. Agnius Liutkevičius

Recenzentas

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Karolis Pavydis

Programos kodo užtemdymo metodas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Karolio Pavydžio, baigiamasis projektas tema „Programos kodo užtemdymo metodas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	(parašas)

Pavydis Karolis. Programos kodo užtemdymo metodas. Magistro baigiamasis projektas vadovas prof. Jevgenijus Toldinas; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informacijos ir informacinių technologijų sauga

Reikšminiai žodžiai: Kodo užtemdymas, programų apsauga.

Kaunas, 2019. 77 p.

Santrauka

Šiuolaikinės programos susiduria su apgrąžos inžinerijos grėsmėmis. Piktavaliai gali bandyti analizuoti programą, siekiant išgauti intelektines paslaptis, algoritmus, jautrią programos informaciją, taip pat bandyti modifikuoti programą. Vienas iš apgrąžos inžinerijos prevencijos įrankių yra programos pirminio kodo užtemdymas, kada programos kodas yra modifikuojamas įvairiomis transformacijomis taip, kad taptų sunkiau analizuojamas ir suprantamas, tačiau nepasikeistų jo funkcionalumas. Pirminio programos kodo užtemdymas negarantuoja apsaugos nuo apgrąžos inžinerijos, o tik ją apsunkiną, todėl kartu yra taikomos ir kitos apsaugos priemonės.

Magistrinio darbo objektas – kombinuotas programų pirminio kodo užtemdymo metodų taikymas. Darbe siekiama ištirti, kaip skirtinga kodo užtemdymo metodų taikymo tvarka, veikia kodo užtemdymo efektyvumą.

Pirmoje darbo dalyje yra apžvelgiamos programoms kylančios grėsmės, apsaugos metodai, bei atliekama kodo užtemdymo metodų analizė. Joje nagrinėjami dažniausiai naudojami pirminio programų kodo užtemdymo metodai, bei transformacijos. Taip pat apžvelgiamos kodo užtemdymo efektyvumo metrikos, pagal kurias gali būti vertinama kodo užtemdymo metodo kokybė.

Antroje darbo dalyje yra pateikiamos programų pirminio kodo užtemdymo metodų kombinacijų taikymo tvarkos, užtemdymo programos prototipo modelis. Yra apibrėžiama šios programos struktūra, funkcionalumas, reikalingi papildomi įrankiai. Šioje dalyje taip pat pateiktas etaloninis programos kodo failas, kuris bus naudojamas kodo užtemdymo tyrimui.

Trečioje dalyje yra pateikta eksperimentinio tyrimo eiga. Eksperimento metu buvo ištirti dvidešimt keturi kombinuoti užtemdymo metodai, naudojant kodo užtemdymo prototipą. Tyrimo metu gauti užtemdymo stiprumo, atsparumo, naudojamų resursų, bei bendro kodo užtemdymo efektyvumo įverčiai. Šiame skyriuje taip pat pateikti tyrimo rezultatai. Tyrimo metu gauti kodo užtemdymo metodų efektyvumo įverčiai iliustruoti grafikais, bei pateikti lentelėse. Šioje dalyje matomi rezultatai, kurios kodo užtemdymo metodų taikymo kombinacijos surinko aukščiausius ir žemiausius įverčius.

Pavydis Karolis. Method for Obfuscating Program Source Codes. Master's Final Degree Project supervisor prof. Jevgenijus Toldinas; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Information and Information Technology Security.

Keywords: Code obfuscation, software protection.

Kaunas, 2019. 77p.

Summary

Nowadays software encounters a threat of being reversed engineered. Malicious users could be trying to analyze the program in order to retrieve intellectual property, algorithms or other sensitive information, which could be in the program, and also in order to modify the program. One of the tools that can be used in order to protect software against reverse engineering is code obfuscation, when source code of the program is modified with various transformations in such a way that it would be harder to analyze and understand it, but the functionality of the program would stay the same. However, source code obfuscation does not guarantee protection from reverse engineering, it rather just makes it harder to do so and should be use together with other protection measures.

The object of this master thesis is combined use of source code obfuscation methods. In this project it is looked at how different application order of obfuscation methods affects effectiveness of code obfuscation.

First part of this thesis contains overview of the threats related to software, protection measures and analysis of existing code obfuscation methods, where most used source code obfuscation methods and transformations are overviewed. This part also contains overview of code obfuscation evaluation metrics, which can be used to measure effectiveness of applied code obfuscation.

In the second part of this project suggested combinations of code obfuscation methods are presented, together with model of obfuscation program prototype, its structure, functionality and necessary tools. Also benchmark file which is used to test code obfuscation effectiveness is presented here.

Third part contains implementation of experiment of source code obfuscation methods. In this experiment, twenty-four different obfuscation combinations were tested, using our obfuscator prototype. Measures of strength, potency, cost and overall effectiveness were calculated for all obfuscation combinations. This part also contains the results of this project. The measurements of effectiveness are showed in graphs and tables. In this part you can see which method achieved highest obfuscation effectiveness as well as lowest and also which combination of obfuscation methods corresponds to this method.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Įvadas

Informacijos mainai yra vienas iš svarbiausiu šiuolaikinės visuomenės komponentų. Kompiuterinės programos yra kritiškai reikalingos daugelyje šiuolaikinių organizacijų, bei eiliniams vartotojams. Kai kuri programinė įranga gali būti labai brangi, dėl to dalis vartotojų pradeda siekti naudoti programinę įrangą nelegaliai. Be didelių pastangų vartotojas gali susirasti pataisytą programą, kuria galėtų naudotis už ją nemokėdamas. Dėl piratavimo, programinės įrangos kūrėjai gali prarasti didžiulius pinigus. Tačiau piratavimas nėra vienintelė problema dėl kurios reikalinga saugoti programinę įrangą. Kenkėjiškas programos modifikavimas, norint sutrikdyti normalų galinio vartotojo sistemos darbą arba pavogti jautrius duomenis, taip pat yra problema į kurią turi atsižvelgti programų kūrėjai.

Šiuolaikinės programos yra pakankamai sudėtingos ir yra vertingos programos kūrėjams. Programoje gali būti svarbūs algoritmai, šifravimo raktai, ar kita intelektinė nuosavybė, kurie nepasaugotoje programoje gali būti atskleisti pasinaudojus apgrąžos inžinerija. Siekiant apsisaugoti nuo tokių grėsmių programų kūrėjams reikalinga diegti tam tikras apsaugos priemones, kurios apsaugotu nuo nelegalaus programų naudojimo, ar neteisėtos programos analizės. Tačiau pilnai apsaugoti programą, nuo kenkėjiškų vartotojų yra gana sudėtinga užduotis, kadangi jei programą yra vartotojo kompiuteryje, jis turi pilną kontrolę. Taip pat programos apsauga, neturėtų keisti numatyto programos funkcionalumo ar stipriai padidinti jos imlumą resursams.

Pasinaudojus apgrąžos inžinerija galima identifikuoti kritinius taškus programoje, kurie atsakingi už vartoto autentifikavimą ar kitas patikrinimo operacijas. Taip programos kodo analizė gali leisti įsilaužėliui identifikuoti programos saugumo spragas ir pažeidžiamumus. Turint tokią informaciją, įsilaužėlis gali bandyti išnaudoti esamas spragas ar galbūt panaudoti tam tikrą kritinį algoritmą savo programoje.

Atsižvelgdami į visas potencialias grėsmes programų, kūrėjai naudoja įvairius apsaugos metodus norint apsisaugot nuo neteisėtos programos analizės, kenkėjiško modifikavimo ir piratavimo. Tai ypač aktualu tokioms programoms kurios yra prieinamos ir visuomet yra atviros atakoms. Vienas iš būdų naudojamų apsisaugoti nuo neteisėtos programų analizės yra programos kodo užtemdymas. Naudojant kodo užtemdymą yra stipriai apsunkinamas kodo skaitymas, taip pat pakeičiami kintamųjų ir funkcijų vardai saugoja nuo apgrąžos inžinerijos. Taip pat kodo užtemdymas gali paslėpti jautrius duomenis programoje ar net tam tikras programos spragas ir pažeidžiamus. Tačiau nors kodo užtemdymas gali padėti išvengti apgrąžos inžinerijos, jis negarantuoja absoliučios apsaugos ir turėtu būti naudojamas kartu su kitais apsaugos būdais [1]. Naudojant kodo užtemdymą, tam tikras raktas gali nurodyti, kokios transformacijos buvo naudotos ir kokia tvarka. Šis raktas programos savininkui leidžia atstatyti užtemdytą programą.

Magistrinio darbo tikslas – ištirti esamas kodo užtemdymo strategijas, bei pasiūlyti savo metodą programos kodo užtemdymui.
Darbo tikslui pasiekti išsikelti uždaviniai:

- Atlikti programoms kylančių grėsmių analizę.
- Išnagrinėti programų apsaugos metodus.
- Išnagrinėti kodo užtemdymo strategijų efektyvumą.
- Sudaryti kodo užtemdymo strategiją.
- Atlikti sudarytos užtemdymo strategijos efektyvumo tyrimą.

1. Programų kodo užtemdymo strategijų analizė

Pastaruoju metu vis daugiau programinės įrangos ir paslaugų yra tiekiama per internetą. Tai gali būti taikomosios programos, e-prekyba, e-bankininkystė. Šios programos pasiekusios galinį vartotoją, jau nėra kontroliuojamos programos kūrėjų [1]. Todėl yra svarbu tinkamai apsaugoti programas, kadangi jose gali būti jautrios informacijos, kaip kad slapti raktai, kredito kortelių numeriai ar kita konfidenciali informacija. Norint apsaugot jautrią informaciją yra naudojama kriptografija ir atitinkamos autentifikavimo sistemos. Tačiau taip pat svarbu ir apsaugoti programą nuo apgrąžos inžinerijos bandymų, kad nebūtų atskleistas programos įgyvendinimas. Kartu turi būti saugomi ir kritiniai programos taškai, kur vyksta autentifikavimas ar yra naudojami konfidencialus duomenys. Vykdant šių kritinių taškų kodą piktavalis vartotojas gali naudoti dinaminę analizę ar bandyti suklastoti duomenis, todėl yra svarbu, kad jie būtų tinkamai apsaugoti siekiant išvengti galimų atakų.

1.1. Programoms kylančios grėsmės

Tiekiant programas į klientų kompiuterius programų kūrėjai praranda programos kontrolę, todėl yra galimybė, kad programa pateks pas kenkėjišką vartotoją [3] [14]. Kliento pusėje programa tampa pažeidžiama kenkėjiško vartotojo. Programa gali būti atakuojama ir pačio piktavalio vartotojo arba kitos piktavališkos programos esančios vartotojo kompiuteryje. Piktavalis vartotojos gali naudoti įvairius apgrąžos inžinerijos įrankius, norėdamas išanalizuoti ir modifikuoti programą. Po sėkmingos programos analizės įsilaužėlis gali mėginti atlikti tokius pakeitimus, kad už programą nereikėtų mokėti, taip pat gali būti bandoma išgauti atitinkamą intelektinę nuosavybę, pavyzdžiui, svarbius algoritmus [1]. Galima išskirti tokias atakas prieš šiuolaikines programas [2]:

- Apgrąžos inžinerija norint išsiaiškint programos veikimą.
- Programos funkcionalumo keitimas, klastojant programos kodą.
- Piratavimas, kuriant nelegalias programos kopijas.
- Programos saugos patikrinimų apėjimas, kaip, kad autentifikacijos ar licencijos patikra.

Norint apsisaugoti nuo apgrąžos inžinerijos programų kūrėjai, naudoja daug įvairių priemonių, tarp jų ir kodo užtemdymą. Norint apsisaugoti nuo apgrąžos inžinerijos rekomenduojama naudoti kodo užtemdymą su programos vykdymo eigos išlyginimu, signalais paremtą užtemdymą ar asemblerio lygio užtemdymą [11][12][13][15].

1.1.1. Apgrąžos inžinerija

Programos apgrąžos inžinerija yra procesas, kada programa yra dekompiliuojama, norint pasiekti asemblerio ar pirminį programos kodą [3][14]. Apgrąžos inžinerija programose naudoja įvarius analizės metodus, norint sužinoti programos veikimą ar išgauti programoje aprašytus duomenis ar intelektinę nuosavybę. Įsilaužėliui programos analizė gali būti pradinis taškas norint toliau modifikuoti programą savais tikslais. Todėl naudojant apsaugos metodus, kurie apsunkina programos analizę, taip pat iš dalies yra apsisaugojama ir nuo neteisėto programos modifikavimo. Naudojant apgrąžos inžineriją programai gali būti taikoma statinė arba dinaminė analizė [1][2][3][19]. Naudojant šiuos apgrąžos inžinerijos metodus yra stengiamasi suprasti programos vykdymo eigą ir

iš to susidaryti pilną programos kodo vaizdą. Apgrąžos inžinerijos procesas pavaizduotas 1.1 paveiksle.

1.1 pav. Apgrąžos inžinerijos procesas [14]

Norint apsaugot programą nuo dinaminės analizės, gali būti naudojamas projektavimo lygio užtemdymas, saugojantis nuo statinės kodo analizės, kodas gali būti užtemdomas asemblerio lygyje [14].

Statinė analizė

Šis analizės metodas yra naudojamas analizuoti programai, kai nėra vykdomas jos kodas [2][3]. Norint atlikti statinę analizę, sukompiliuotas dvejetainės programos kodas analizuojamas ir transformuojamas į asmeblerio komandas (*ang. disassemble*), tolimesnei analizei. Šios asemblerio komandos toliau gali būti bandomos dekompiliuoti į pradinį programos kodą. Turint pradinį programos kodą, jam yra taikoma statinė kodo analizę. Ši analizė gali išgauti daugiau galimų programos vykdymo eigų, nei dinaminę analizė, kuri išanalizuoja tik, įvykdytą programos eigą [19].

Dinaminė analizė

Dinaminė analizė yra atliekama vykdant programos kodą [2][19]. Naudojant šį metodą yra sekamos įvykdytos instrukcijos, pasikeitusios kintamųjų reikšmės. Dinaminė analizė yra imlesnė laikui nei statinė analizė [19].

Hibridinė analizė

Siekiant geresnių apgrąžos inžinerijos rezultatų, gali būti naudojama hibridinė analizė, kuomet yra kombinuojama statinės ir dinaminės analizės metodai [2][3].

Programos kodo klastojimas

Įsilaužėlis sėkmingai atlikęs programos kodo analizę, gali toliau bandyti keisti programos funkcionalumą [10]. Norint tinkamai modifikuoti programą, labai svarbu žinoti, kaip tiksliai programa veikia, todėl šios atakos efektyvumas labai priklauso nuo prieš tai naudotos apgražos inžinerijos rezultatų. Klastojant programos kodą, gali būti apeinami autentifikavimo algoritmai ar įterpiamas kenksmingas kodas. Programa gali būti klastojama statiškai arba dinamiškai.

Atakos prieš programas gali būti klasifikuojamos į tokias kategorijas:

- Kenkėjiška apgrąžos inžinerija.
- Piratavimas.

- Programos klastojimas.

Sistemos, kurios yra atviros ir lengvai pasiekiamos turi aukštesnę riziką būti atakuojamos, nei uždaros sistemos. Tai pat didesnė grėsmė kyla tokioms programos, prieš kurias įsilaužėlis, sėkmingai įvykdęs ataką gautu daug naudos, pavyzdžiui e-bankininkystės sistemos.

Programų dekompiliatoriai

Atliekant apgrąžos inžineriją programinei įrangai gali būti pasitelkiami programų dekompiliatoriai [5]. Dekompiliatorius veikia priešingai nei programų kompiliatoriai – yra paimamas sukompiliuotas programos dvejetainis failas ir vykdant priešingą kompiliacijai procesą iš jo yra bandoma išgauti pradinį ar panašų į pradinį programos kodą, kuris būtų suprantamas žmogui. Dekompiliuojant dvejetainį kodą, grąžintos programos kodo struktūrą gali būti pasikeitus, tačiau to gali užtekti, kad piktavalis galėtu atsekti programos veikimo principus ar kitą jautrią informaciją. Java programų dekompiliavimas vyksta transformuojant Java baitkodą į Java kodą. Ši operaciją yra lengvesnė nei mašininio kodo dekompiliavimas. Kadangi šiame projekte bus tiriamas Java kalbos užtemdymas, toliau apžvelgiamą keletas Java programų dekompiliatorių:

Procyon

Procyon java dekompiliatorius yra atviro kodo įrankis kuris gali grąžinti pirmini kodą iš Java 5 ir naujesnių Java programų [23]. Šis dekompiliatorius gerai tinka dekompiliuojant programą kurioje yra naudojama Java 8 funkcijos, lambdos, anotacijos.

DJ Java Decompiler

Šis įrankis skirtas Java kodo analizei, veikia Windows operacinėje sistemoje ir nereikalauja, kad sistemoje būtų įrašyta Java virtuali mašina [22]. Šio dekompiliatoriaus privalumas, kad yra galima analizuoti daugiau nei vieną Java klasę vienu metu, taip galima, modifikuoti, išsaugoti ir kompiliuoti grąžintą programos kodą. Įrankis gali analizuoti .JAR, .ZIP, .APK, .EAR, .WAR ir .EXE tipo failus. Taip pat galima kodo peržiūra šešioliktainiu formatu.

JDProject

JDProject dekompiliatorius skirtas analizuoti Java 5 ir naujesnių versijų programas [20]. Šis įrankis gali būti naudojamas Windows, Linux ir Mac OS operacinėse sistemose. JDProject yra patogus kadangi yra galimi įskiepiai Eclipse ir Intellij IDEA platformoms

AndroiChef Decompiler

AndroiChef Decompiler įrankis skirtas naudoti Windows aplinkoje ir gali būti naudojamas analizuoti užtemdytus Java 6, Java 7 ir Java 8 .class ir .jar tipo failus. Įrankis geba dekompiliuoti 98,04% Java programų, sugeneruotų tradiciniais kompiliatoriais [21]. Įrankis taip pat gali būti naudojamas Dalvik baitkodo analizei iš .apk, ir .dex tipo failų.

1.2. Programų apsaugos metodai

Norint apsisaugoti nuo kenkėjiškų veiksmų prieš programas yra naudojami teisiniai, programiniai ir aparatūriniai sprendimai [3]. Teisinė apsauga yra skirta apsiginti nuo vartotojų, kurie nėra tvirtai pasiryžę atlikti nelegalius veiksmus, todėl bijodami gręsiančios atsakomybės galbūt susilaikys nuo atakos. Autorinės teisės gali drausti daryti tam tikros intelektinės nuosavybės kopijas, tačiau techniškai tai nėra apsauga nuo piratavimo, todėl turi būti naudojama kartu su kitomis priemonėmis.

Programų apsaugai taip pat gali būti naudojamos aparatūrinės priemonės, specialūs raktai, kuriuose būtų kritinė programos dalis, ar licencijos kodas. Aparatūrinės apsaugos priemonės susiduria su programų tiekimo ir aptarnavimo problema, palyginus didele kaina. Programiniai apsaugos sprendimai gali būti pakankamai pigūs ir lankstūs įvairioms sistemoms. Programose jautrūs duomenys gali būti saugomi naudojant kriptografinius sprendimus, norint apsisaugot nuo apgrąžos inžinerijos atakų, gali būti naudojamas kodo užtemdymas. Programinių apsaugos metodų stiprumas priklauso, nuo naudojamų šifravimo algoritmų patikimumo, užtemdymo strategijų efektyvumo ir tinkamo jų įgyvendinimo, bei turi būti nuolat atnaujinami, kadangi laikui bėgant tobulėja ir apgrąžos inžineriją atliekančios programos, bei auga įsilaužėlių kompetencija

Kliento – serverio sistema

Vienas iš metodų norint apsaugoti programas nuo neleistino naudojimo yra leisti programą savininko serverio pusėje, o ne vartotojo kompiuteryje, tokiu atveju programa yra labiau kaip paslauga, kuri yra prieinama vartotojui [3][10]. Tokiu atveju programa gali būti patalpinta apsaugotame serveryje, tačiau pats kodas nėra apsaugotas. Naudojant šį metodą programos vykdomasis kodas visada lieka serveryje ir yra iš išorės nepasiekiamas, o vartotojas tik gauna atitinkamą paslaugą pagal įvestus duomenis. Taip pat šis metodas gali būti naudojamas dalinai - serveryje paliekama veikti tik kritinė programos dalis, o likusi nekritinė dalis veikia vartotojo kompiuteryje. Šio metodo trūkumas, kad esant dideliam sistemos apkrovimui, programa gali tapti nepasiekiama.

Kodo šifravimas

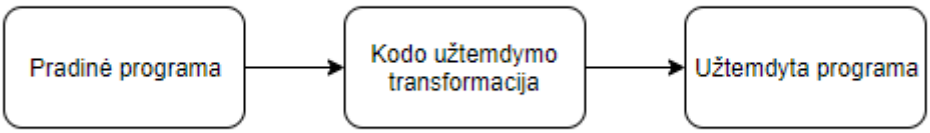
Norint sutrukdyti įsilaužėliui skaityti programos kodą jis gali būti šifruojamas [4]. Šis metodas apsaugo programas nuo statinės apgrąžos inžinerijos, kadangi įsilaužėlis negali skaityti programos kodo ar jo modifikuoti. Vykdant tokiu būdu apsaugota programą, dalis jos kodo yra dešifruojama ją bevykdant, tuo metu kodas yra matomas atmintyje iš kuri gali perimamas. Perimtas kodas jau gali būti analizuojamas.

Kodo pasirašymas

Kodo pasirašymas naudojamas norint apsaugoti programą nuo neleistino jos modifikavimo. Kiekviena kart leidžiant programą yra patikrinama ar nėra pažeistas jos vientisumas [4].

Kodo užtemdymas

Kodo užtemdymas gali būti apibrėžiamas, kaip programos transformacijos siekiant padaryti programos kodą sunkiai analizuojamu [1][2][7][10]. Programoje panaudotos kodo užtemdymo strategijos apsunkina tiek fizinę kodo analizę, tiek apsaugo nuo automatizuotu apgrąžos inžinerijos įrankių. Kodo užtemdymas pakeičia kintamųjų ir funkcijų vardus, pakeičia loginę struktūrą, bei įveda programos funkcionalumo nekeičiančių operacijų, taip paslėpdamas programoje esančią informacija ar galimas programos spragas. Naudojant kodo užtemdymą yra naudojama keletas transformacijų, kurios padarytų, kodą labiau atsparų apgrąžos inžinerijai ir kodo klastojimui, tačiau išlaikant originalų funkcionalumą. Kodo užtemdymas taip pat gali būti taikomas procesoriaus instrukcijų lygmenyje.



1.2 pav. Taikant įvairias kodo transformacijas pradinei programai yra gaunama užtemdyta programa, kurios veikimo rezultatas yra toks pat, tačiau programos kodas yra sunkiau analizuojamas

Kodo užtemdymą galima apibrėžti taip [15]:
Jei turime pradine programa P ir galutinę programą P', kuriai pritaikyta transformacija τ. Transformacija τ galima laikyti užtemdymo transformacija jei yra išpildomos tokios sąlygos:

- Jei vykdant P yra grąžinama klaida, P' gali būti įvykdyta sėkmingai arba taip pat grąžinti klaidą.
- Jei P įvykdoma be klaidų, tuomet P' turi grąžinti tokius pačius rezultatus kaip ir P

P' gali turėti papildomo funkcionalumo, kuris yra nematomas vartotojui, taip pat gali skirtis programų vykdymo greitis, dydis, sunaudojami resursai.
Nors ir kodo užtemdymas apsunkina programos analizę tačiau jis negarantuoja pilnos apsaugos, o tik prailgina laiką, kurį turėtų skirti įsilaužėlis norėdamas išanalizuoti programą. Naudojant kodo užtemdymą, kartu su kitais programų apsaugos metodais siekiama, kad eiliniam įsilaužėliui tai taps per brangu ir per imlu laikui bandyti analizuoti ar modifikuoti programą.

1.3. Kodo užtemdymo efektyvumo kriterijai

Šiame skyriuje apžvelgiamos, programų kodo užtemdymo efektyvumo metrikos, reikalingos įvertinti pasirinktos kodo strategijos efektyvumą

1.3.1. Kodo užtemdymo efektyvumo matavimas

Norint įvertinti programos kodo užtemdymo strategijos efektyvumą literatūroje [1][15] rekomenduojama išskaidyti užtemdymo efektyvumo įvertį į keletą dedamųjų. Bendras programos kodo užtemdymo efektyvumas S_{efek} susideda iš trijų pagrindinių įverčių – užtemdymo stiprumo, atsparumo prieš automatinius įrankius, bei naudojamų resursų kainos. Užtemdymo stiprumas S_{stip} nurodo, kiek kodo užtemdymas yra apsunkinęs programos suprantamumą žmogui analizuojančiam programos kodą. Atsparumas prieš automatinius įrankius S_{atsp} nurodo, kaip užtemdyta programa atsilaiko prieš automatinius įrankius skirtus atlikti apgrąžos inžinerija, kurie bando grąžinti pirminį programos kodą iš sukompiliuotos programos. Resursų kainos įvertis S_{kain} nurodo, kiek papildomu resursų sunaudoja užtemdyta programa. Skaičiuojant bendra programos kodo užtemdymo efektyvumą šiems trims įverčiams yra taikomi skirtingi svorio koeficientai. Didesnė kodo užtemdymo efektyvumo vertė S_{efek} reiškia efektyvesnį kodo užtemdymą, tuo tarpu nulinė vertė reiškia, kad kodas nėra užtemdytas. Literatūroje [15] rekomenduojami svorio koeficientai x = 0,4 ir y = 0,6.
Kodo užtemdymo efektyvumas apskaičiuojamas pagal (1.1) formulę:

$$S_{efek} = x \cdot S_{stipr} + y \cdot S_{atsp} - S_{kain} \quad (1.1)$$

čia S_{efek} – stiprumo įvertis; S_{atsp} – atsparumo įvertis; S_{kain} – kainos įvertis;

1.3.2. Kodo užtemdymo stiprumas

Šis įvertis nurodo kiek programos kodas yra sudėtingas suprasti žmogui. Kad būtų paprasčiau apskaičiuoti kodo užtemdymo stiprumą rekomenduojama šį įvertį išskaidyti į keletą sudėtinių dalių: programos eigos sudėtingumą, ciklų gylį, kintamųjų sudėtingumą, bei programos ilgį [1][15].

Programos eigos sudėtingumo įvertis
Šis įvertis matuoja kaip nuosekliai ar nenuosekliai yra vykdomas programos kodas [15]. Nenuoseklus programos vykdymas reiškia, kad programos kode yra naudojamos sąlygos, pagal kurias yra atliekami šuoliai programos logikoje, kaip pavaizduota 1.3 paveiksle. Programos eigos sudėtingumo įvertis susideda iš sąlygų (*ang. labels)* įverčio, šuolių įverčio, bei neskaitomumo įverčio.

```
Label1:
   statement
   goto Label3
Label2:
   statement
   goto Label1
Label3:
   Goto Label2
```

1.3 pav. Nenuosekli programos struktūra

Sąlygų įvertis nurodo kiek programoje yra tokių pačių sąlygos pavadinimų (*ang. label)*. Pasikartojantys šiuo atvejų yra tokie patys sąlygos pavadinimai, tačiau esantys skirtinguose metoduose. Kuo daugiau pasikartojimų tuo didesnis šis įvertis. Sąlygų įvertis apskaičiuojamas pagal (1. 2) formulę:

$$S_l = \frac{l}{l'} \tag{1.2}$$

čia S_l – sąlygų įvertis; l – pasikartojančios sąlygų pavadinimai užtemdytoje programoje;
l' – visi sąlygų pavadinimai užtemdytoje programoje;

Programos šuolių įvertis nurodo kiek nenuoseklių šuolių yra užtemdytoje programoje. Šie šuoliai apibūdinami kaip *go-to* instrukcijos [15]. Šis įvertis apskaičiuojamas pagal 1.3 formulę:

$$S_g = \frac{g}{g'} \tag{1.3}$$

čia S_g – nenuoseklių šuolių įvertis; g – nenuoseklių šuolių skaičius užtemdytoje programoje;
g' – šuolių skaičius programoje;

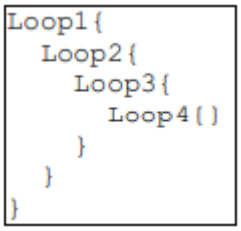
Neskaitomumo įvertis vertina kiek programos kodo eilučių dekompiliavimo įrankis negalėjo grąžinti į pirminį programos kodą [15]. Tai reiškia, kad programos instrukcijos yra suprantamos procesoriui, tačiau negali būti pateikiamos, kaip programos kodas kuris būtų suprantamas žmogui. Šis įvertis nurodo koks procentas kodo eilučių programoje yra negrąžinamos dekompiliatoriaus. Neskaitomumo įvertis apskaičiuojamas pagal (1.4) formulę:

$$S_{neskait} = \frac{c}{c'} \quad (1.4)$$

čia $S_{neskait}$ – neskaitomumo įvertis; c – negrąžinamų eilučių skaičius;
c' – eilučių skaičius užtemdytoje programoje;

Ciklų gylio sudėtingumo įvertis

Ciklų sudėtingumo įvertis matuoja skirtinguose hierarchiniuose lygiuose programoje esančių ciklų skaičių [15]. Ciklų hierarchiniai lygiai gali būti suprantama, kaip ciklas esantis kito ciklo viduje kaip matoma 1.4 paveiksle.



1.4 pav. Ciklų hierarchiniai lygiai

Ciklų gylio įvertis gali būti apskaičiuojamas pagal (1.5) formulę:

$$S_c = \sum_{i=1}^{N} lygis_i \cdot kiekis_i \quad (1.5)$$

čia S_c – ciklų gylio įvertis; N – didžiausias ciklo gylis programoje; i – ciklo gylio lygis;

Paskaičiavus kiekviename hierarchiniame lygyje esančių ciklų kiekį ir gavus ciklų įvertį galima paskaičiuoti ciklų gylio sudėtingumo įvertį programoje. Šis įvertis paskaičiuojamas pagal (1.6) formulę:

$$S_{cikl} = \frac{S_c - S'_c}{S'_c} \quad (1.6)$$

čia S_{cikl} – ciklų gylio sudėtingumo įvertis; S'_c – ciklų gylio įvertis pradinėje programoje;
S_c – ciklų gylio įvertis užtemdytoje programoje;

Kintamųjų sudėtingumo įvertis

Kintamųjų sudėtingumo įvertis taip pas susideda iš kelių dedamųjų. Jį sudaro pasikartojančių kintamųjų įvertis, papildomų kintamųjų įvertis, prasmingų kintamųjų įvertis, bei šifruotų eilučių įvertis [15].

Pasikartojančių kintamųjų įvertis

Pasikartojančių kintamųjų įvertis nurodo kiek kintamųjų programos kode turi tą patį pavadinimą, tačiau skirtingas reikšmes [15]. Šis įvertis gali būti paskaičiuojamas pagal (1.7) formulę:

$$S_{kart} = \frac{k_{kart}}{k'} \quad (1.7)$$

čia S_{kart} – pasikartojančių kintamųjų įvertis; k' – visų kintamųjų skaičius užtemdytoje programoje; k_{kart} – pasikartojančių kintamųjų skaičius;

Papildomų kintamųjų įvertis

Šis įvertis nurodo kiek kintamųjų užtemdytam programos kode yra turintys tas pačias reikšmes, tačiau skirtingus pavadinimus [15].

Papildomų kintamųjų įvertis apskaičiuojamas pagal (1.8) formulę:

$$S_{ekstra} = \frac{k_{ekstra}}{k'} \quad (1.8)$$

čia S_{ekstra} – papildomų kintamųjų įvertis; k' – visų kintamųjų skaičius užtemdytoje programoje; k_{ekstra} – papildomų kintamųjų skaičius;

Prasmingų kintamųjų įvertis

Prasmingų kintamųjų įvertis S_{pras} nurodo ar kodo užtemdymo transformacija pakeitė kintamųjų pavadinimus į tokius, kurie neturi prasmingos reikšmės [15]. Šis įvertis gali turėti dvi reikšmes – jei kodo užtemdymo transformacija nepadarė pakeitimų kintamųjų pavadinimams, šis įvertis lygus nuliui, jei kodo užtemdymo transformacija pakeičia pavadinimus į tokius, kurie neturi prasmės tuomet įvertis lygus vienam.

Šifruotų kintamųjų įvertis

Šis įvertis vertina šifruotas eilutes užtemdytame kode. Skaičiuojant šį įvertį galimos reikšmės nuo nulio iki trijų [15]. Mažiausias įvertis nulis yra tada, kai kintamieji yra nešifruojami. Įverčio reikšmė 1 yra gaunama jei kintamieji šifruoti, tačiau dešifravimo metodas yra toje pačioje klasėje. Jei dešifravimo metodas yra aptinkamas programoje, tačiau ne toje pačioje klasėje įvertis yra lygus 2. Jei kintamieji yra šifruoti, o dešifravimo metodas visoje programoje yra neaptinkamas tuomet šifruotų kintamųjų įvertis yra lygus 3.

Programos ilgio įvertis

Programos ilgio įvertis nurodo kiek pailgėjo programos kodas pritaikius kodo užtemdymo transformaciją, lyginant su pradine programa. Programos ilgio įvertis apskaičiuojamas pagal (1.9) formulę:

$$S_{ilg} = \frac{i}{i'} \quad (1.9)$$

čia S_{ilg} – programos ilgio įvertis; i' – eilučių skaičius užtemdytoje programoje; i – eilučių skaičius pradinėje programoje;

Bendras kodo užtemdymo stiprumo įvertis

Bendras stiprumo įvertis S_{stipr} susideda iš visų dedamųjų aprašytų anksčiau. Šis įvertis matuojamas skalėje nuo nulio iki šimto, kur nulis reiškia maža stiprumą, o šimtas labai didelį kodo užtemdymo

stiprumą. Skaičiuojant šį įvertį prie kiekvienos dedamosios yra naudojami svorio koeficientai, kadangi skirtingi įverčiai, turi nevienodą įtaka programos analizės sudėtingumui [15]. Bendras kodo užtemdymo stiprumo įvertis gali būti apskaičiuotas pagal 1.20 formulę:

$$S_{stipr} = x \cdot S_{cikl} + y \cdot S_{l} + y \cdot S_{g} + z \cdot S_{neskait} + z \cdot S_{kart} + \\ + z \cdot S_{ekstra} + z \cdot S_{pras} + x \cdot S_{enc} + x \cdot S_{ilg} \quad (1.20)$$

Literatūroje [15] rekomenduojama naudoti svorio koeficientus kurie yra x = 12,5; y = 18,75; z = 6,25.

1.3.3. Kodo užtemdymo atsparumo įvertis

Kodo užtemdymo atsparumo įvertis nurodo užtemdyto kodo atsparumą prieš įrankius skirtus atlikti programinės įrangos apgrąžos inžineriją, bandant grąžinti pradinį programos kodą iš dvejetainio kodo [15]. Atsparumo įvertis priklauso nuo to, kokie rezultatai yra gaunami atlikus kodo dekompiliacija. Jei dekompiliatorius grąžina pirminį kodą be klaidų šis įvertis yra lygus nuliui, jei įrankis grąžina pirmini kodą su klaidom – įverčio reikšmė vienas, jei programos nepavyksta dekompiliuoti tuomet įvertis yra lygus dviem. Jei programa yra sudaryta iš keleto kodo failų, tuomet yra testuojamas kiekvienas failas ir naudojamas rezultatų vidurkis.

1.3.4. Naudojamų resursų kainos įvertis

Kadangi užtemdyta programa gali būti didesnė ir sudėtingesnė, gali pasikeisti jos vykdymo laikas, yra skaičiuojamas resursų kainos įvertis, kuris yra neigiama dedamoji programos užtemdymo efektyvumui [15]. Šis įvertis nurodo kiek papildomu resursų sunaudoja užtemdyta programa lyginant su pradine programa. Resursų kainos įvertis gali būti išskaidytas į tris dalis: naudojamą atmintį, programos vykdymo laiką bei užimama vietą diske

Naudojamos atminties įvertis

Šis įvertis nurodo kiek papildomai užtemdyta programa sunaudoja kompiuterio atminties lyginant su pradine programa [15]. Šiam įverčiui apskaičiuoti, naudojama atmintis išskiriama į dvi dedamas *heap* tipo atmintį ir ne *heap* tipo atmintį. *Heap* tipo atmintyje yra programos vykdymo metu reikalingi duomenys, tokie kaip masyvai ar inicializuoti objektai, tuo tarpu ne *heap* atmintyje yra laikoma klasės struktūros, metodų, kintamųjų duomenys. Atminties įverčiai apskaičiuojami pagal 1.21, 1.22 ir 1.23 formules:

$$S_{heap} = \frac{m'_{heap} - m_{heap}}{m_{heap}} \quad (1.21)$$

čia S_{heap} –naudojamos *heap* atminties įvertis; m'_{heap} – naudojama *heap* atmintis užtemdytoje programoje; m_{heap} – naudojama *heap* atmintis pradinėje programoje;

$$S_{ne-heap} = \frac{m'_{ne-heap} - m_{ne-heap}}{m_{ne-heap}} \quad (1.22)$$

čia $S_{ne\text{-}heap}$ –naudojamos ne *heap* atminties įvertis; $m'_{ne-heap}$ – naudojama ne *heap* atmintis užtemdytoje programoje; $m_{ne-heap}$ – naudojama ne *heap* atmintis pradinėje programoje;

Apskaičiavus šiuos atminties įverčius, bendras naudojamos atminties yra paskaičiuojamas pagal formulę:

$$S_m = x \cdot S_{ne-heap} + y \cdot S_{heap} \quad (1.23)$$

čia S_m– naudojamos atminties įvertis; $S_{ne\text{-}heap}$ –naudojamos ne *heap* atminties įvertis;
S_{heap} –naudojamos *heap* atminties įvertis;

Literatūroje rekomenduojami atminties koeficientai x = 0,375 ir y = 0,625 [15].

Programos vykdymo laiko įvertis
Norint įvertinti programos kodo užtemdymo efektyvumą taip pat svarbu, kiek užtemdymo transformacijos apkrauna ir sulėtina programą [15]. Programos vykdymo laiko įvertis paskaičiuojamas pagal (1.24) formulę:

$$S_{laik} = \frac{t' - t}{t} \quad (1.24)$$

čia S_{laik}– programos vykdymo laiko įvertis; t' – užtemdytos programos vykdymo laikas;
t – pradinės programos vykdymo laikas;

Užimamos vietos įvertis
Šis įvertis nurodo kiek pasikeitė programos dydis pritaikius kodo užtemdymo transformacijas. Užimamos vietos įvertis apskaičiuojamas pagal (1.25) formulę:

$$S_{viet} = \frac{v' - v}{v} \quad (1.25)$$

čia S_{laik}– užimamos vietos įvertis; v' – užtemdytos programos užimama vieta;
v – pradinės programos užimama vieta;

Bendras naudojamų resursų kainos įvertis
Apskaičiavus vykdymo laiko, naudojamos atminties, bet užimamos vietos diske įverčius, galima paskaičiuoti bendra naudojamų resursų įvertį pagal (1.26) formulę. Kiekvienas įvertis yra dauginamas iš svertinio koeficiento. Literatūroje rekomenduojami koeficientu svoriai x = 0,4; y = 0,15; z = 0,45;

$$S_{kain} = x \cdot S_m + y \cdot S_{viet} + z \cdot S_{laik} \quad (1.26)$$

čia S_m– naudojamos atminties įvertis; S_{viet} –naudojamos vietos įvertis;
S_{laik} –vykdymo laiko įvertis;

1.4. Kodo užtemdymo transformacijos

Efektyvus kodo užtemdymas apima ne tik kintamųjų ir funkcijų vardų pakeitimus, bet kartu naudoja ir kitas kodo transformacijas, kad padidinti apsaugą prieš apgrąžos inžineriją [1][6][7]. Kodo užtemdymo transformacijos gali atlikti, bet kokius veiksmus su kodu, iki tol, kol nėra pakeičiamas programos funkcionalumas. Nors kodo užtemdymas ir negarantuoja apsaugos nuo apgrąžos

inžinerijos, tačiau taikant keletą užtemdymo transformacijų gali pakeist programą tiek, kad įsilaužėlis su ribotais resursais nesugebės sėkmingai atlikti apgrąžos inžinerijos. Gali būti išskiriamos tokios pagrindinės kodo transformacijos klasės[1][2][12][13][14]:

- Leksinės transformacijos.
- Išdėstymo transformacijos.
- Srauto valdymo transformacijos.
- Duomenų transformacijos.
- Prevencinės transformacijos.

Leksinės transformacijos

Šios transformacijos pakeičia kintamųjų, konstantų ir funkcijų vardus į tokius, kurie neturi jokios semantinės reikšmės [6][7][8]. Taip pat gali būti įterpiami klaidinantys komentarai. Nauji kintamųjų ar konstantų pavadinimai parenkami naudojant atsitiktini eilutės generavimą arba naudojant maišant kitus pavadinimus esančius programoje. Keičiant kintamųjų ir metodų pavadinimus programoje, turi būti atsižvelgiama, kad visi pavadinimai būtų pakeičiami teisingai, kadangi kintamieji gali būti naudojami kitose funkcijose, klasės gali turėti paveldimumą, kai kurie pavadinimai negali būti užtemdomi [8]. Leksinės transformacijos pavaizduotos 1.5 paveiksle.

```
public class Test{

    public void Hello(){

        String hello = "HelloWorld";
        int password = 12345;
        String login = "admin"

        System.out.println(hello);

    }

}
```

a)

```
public class lnrfccphzz {
    public void txxibsqdgq() {
        java.lang.String azicqciuro = "HelloWorld";
        int djtkncftbz = 12345;
        java.lang.String thegckkcff = "admin";
        java.lang.System.out.println(azicqciuro);
    }
}
```

b)

1.5 pav. Leksinė kodo transformacija. Pradinis kodas (a), užtemdytas kodas (b).

Išdėstymo transformacijos

Naudojant programos išdėstymo transformacijas yra pakeičiamas kodo išdėstymas, pašalinamas kodo formatavimas, komentarai bei sumaišomi identifikatoriai, kad žmogui atliekančiam apgrąžos

inžineriją būtų sunkiau skaityti programos kodą [1][6][7][8][11]. Išdėstymo transformacijos pavyzdys pavaizduotas 1.6 paveiksle.

```
loop = 1;
loopSum = 0;

while (loop <= 100){
        loopSum = loopSum + loop;
        loop = loop + 1;
}
```

a)

```
i=1;sum=0;while(i<=100){sum=sum+i;i=i+1;}
```

b)

1.6 pav. Išdėstymo kodo transformacija. Pradinis kodas (a), užtemdytas kodas (b).

Programos vykdymo eigos transformacijos

Šio tipo transformacijos keičia programos vykdymo eigą, nekeisdamos jos funkcionalumo. Į programą gali būti įterpiami nereikšmingi sąlyginiai blokai, bereikšmės instrukcijos, kurios apsunkintų analizę, taip pat neaiškių kintamųjų įvedimas, nereikalingų ciklų naudojimas, vykdymo eigos išlyginimas. Programos vykdymo eigos transformacijos pavyzdys pateiktas 1.7 paveiksle.

```
loop = 1;
loopSum = 0;

while (loop <= 100){
        loopSum = loopSum + loop;
        loop = loop + 1;
}
```

a)

```
tempFoo = 1;
while(tempFoo != 0){
        switch (tempFoo){
                case 1:
                loop = 1;
                loopSum = 0;
                tempFoo = 2;
                break;
                case 2:
                if (loop <= 100)
                        tempFoo = 3;
                else
                        tempFoo = 0;
                break;
                case 3:
                loopSum = loopSum + loop;
                loop = loop + 1;
                tempFoo = 2;
                break;
        }
}
```

b)

1.7 pav. Programos vykdymo eigos transformacija. Pradinis kodas (a), užtemdytas kodas (b).

Duomenų transformacijos

Šios transformacijos padaro duomenis naudojamus programoje sunkiau suprantamus [1][17]. Duomenys gali būti išskaidomi, užkoduojami, pakeičiama masyvų struktūra. Gali būti įvedamos

nereikalingos klasės, papildomi nereikalingi duomenys. Masyvai ir eilutės išskaidomi ir po to vėl sujungiami. Duomenų užtemdymo transformacija, taikant eilutės išskaidyma pavaizduota 1.8 paveiksle.

```
public class Test{
    public void Hello(){
        String hello = "HelloWorld";
        System.out.println(hello);
    }
}
```

a)

```
public class Test{
    public void Hello(){
        String ab = "He";
        String qq = "ll";
        String wq = "ow";
        String eq = "or";
        String dd = "ld";
        String hl = ab + qq + wq + eq + dd;
        System.out.println(hl);
    }
}
```

b)

1.8 pav. Eilutės išskaidymo metodas. Pradinis kodas (a), užtemdytas kodas (b).

Taip pat duomenų užtemdymo transformacijose gali būti naudojamas masyvų restruktūrizavimas, kada yra pakeičiama masyvo elementų struktūra ir tuomet naudojamas papildomas pagalbinis masyvas restruktūrizuoto masyvo skaitymui [1]. Masyvų restruktūrizavimo transformacija pavaizduota 1.9 paveiksle.

```
int[] array = {a,b,c,d,e};
for (int i = 0; i < array.lenght; i++){
    array[i] = array[i] + i;
}
```

a)

```
int[] array = {e,b,d,c,a};
int[] helper = {t,u,v,x,z}
for (int i = 0; i < array.lenght; i++){
    array[helper[i]] = array[helper[i]] + i;
}
```

b)

1.9 pav. Masyvų restruktūrizavimas. Pradinis kodas (a), užtemdytas kodas (b).

Prevencinės transformacijos

Prevencinės transformacijos yra visos, kurios nutaikytos atsilaikyti prieš specifinius užtemdymo kodo analizatorius [14]. Šiose transformacijose tarp instrukcijų gali būti įterpiami nereikalingi baitai, kad

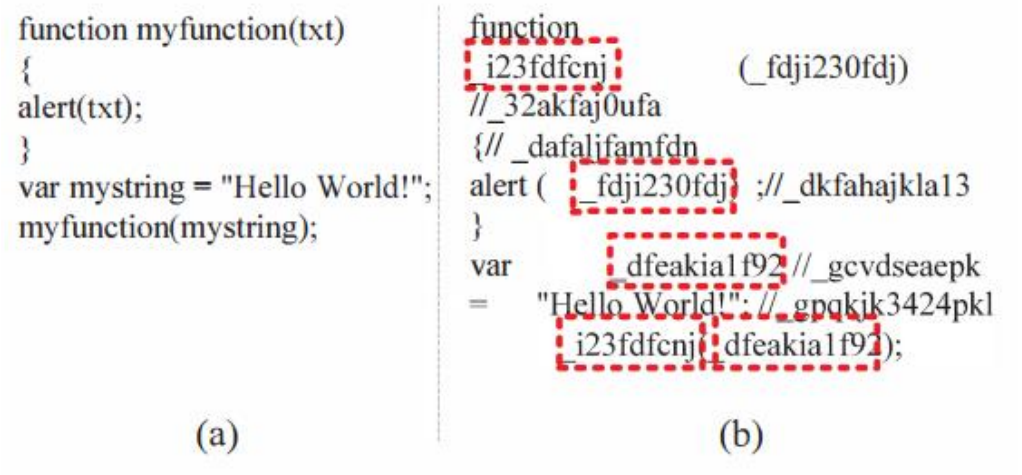
suklaidinti analizatorius. Saugantis nuo derintuvų kodas gali būti generuojamas taip, kad aptiktų ar yra derinimo režime ir tuomet turėtų kitą funkcionalumą.

1.5. Kodo užtemdymo metodai

Kodo užtemdymo strategijos naudoja įvairias kodo užtemdymo transformacijas, ar kelių transformacijų kombinacija kodui paslėpti ir saugotis nuo galimos apgrąžos inžinerijos [11]. Toliau apžvelgiami dažniausiai naudojami kodo užtemdymo metodai.

1.5.1. Atsitiktinis užtemdymas

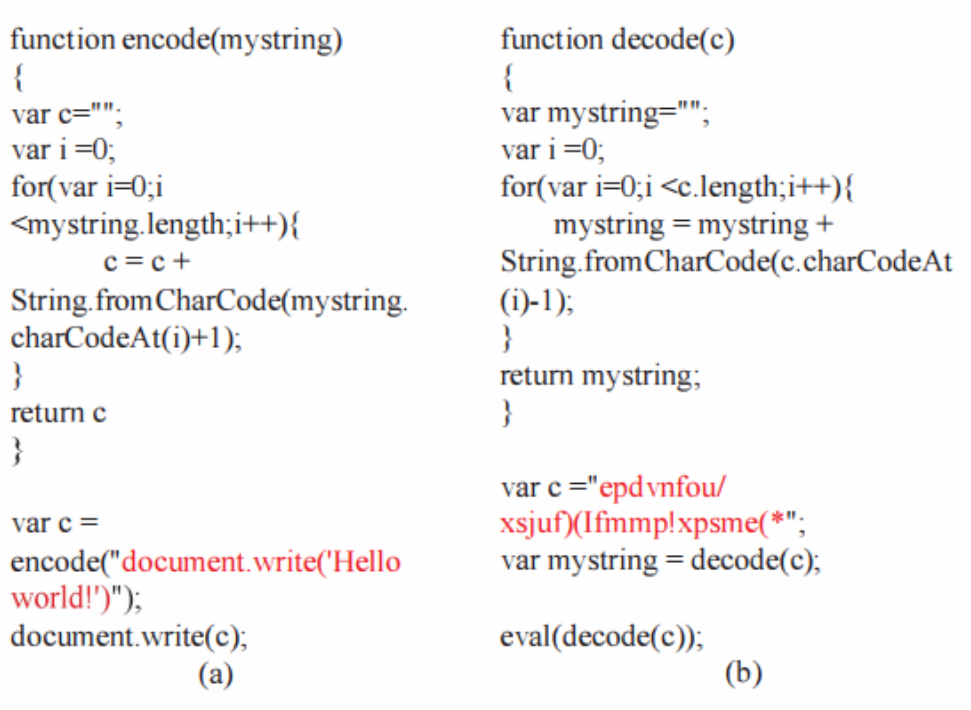
Naudojant šį metodą yra atsitiktinai keičiami ar įterpiami kodo elementai, nekeičiant kodo semantikos [11]. Taip pat atsitiktinai parenkami kintamųjų vardai, keičiamas programos formatavimas, atsitiktinai pridedama nereikalingų komentarų. Siekiant padidinti užtemdymo efektyvumą, gali būti naudojami keli kintamųjų vardų pakeitimo metodai. Atsitiktinio užtemdymo pavyzdys pavaizduotas 1.10 paveiksle



1.10 pav. Kodo užtemdymas taikant atsitiktinio užtemdymo metodą. Pradinis kodas (a), užtemdytas kodas (b) [11].

1.5.2. Užtemdymas naudojant kodavimą

Naudojant šį užtemdymo metodą, kodas gali būti paverčiamas į unikodo ar šešioliktainį atitikmenį [11][17]. Kitas būdas yra naudoti tam tikrą kodavimo funkciją, kuria yra pakeičiamas programos kodas, kuris yra dekoduojamas prieš vykdant programą. Kodo užtemdymo pavyzdys naudojant kodavimo ir dekodavimo funkcijas pavaizduotas 1.11 paveiksle.



```
function encode(mystring)
{
var c="";
var i =0;
for(var i=0;i
<mystring.length;i++){
        c = c +
String.fromCharCode(mystring.
charCodeAt(i)+1);
}
return c
}

var c =
encode("document.write('Hello
world!')");
document.write(c);
```

(a)

```
function decode(c)
{
var mystring="";
var i =0;
for(var i=0;i <c.length;i++){
     mystring = mystring +
String.fromCharCode(c.charCodeAt
(i)-1);
}
return mystring;
}

var c ="epdvnfou/
xsjuf)(Ifmmp!xpsme(*";
var mystring = decode(c);

eval(decode(c));
```

(b)

1.11 pav. Kodo užtemdymas naudojant kodavimo ir dekodavimo funkcijas. Pradinis kodas (a), užtemdytas kodas (b) [11].

1.5.3. Programos vykdymo eigos užtemdymas

Šio tipo užtemdymo transformacijos keičia programos vykdymo eigą ir yra gana efektyvios prieš automatinius apgrąžos inžinerijos įrankius [16]. Programos vykdymo eigos užtemdymo metodai gali būti išskiriami į keturis tipus: nematoma predikacija, skaičiavimo transformacijos, apibendrinimo transformacijos ir eilės tvarkos transformacijos [8][16]. Šie metodai taip pat gali būti skirstomi į tris kategorijas: duomenų užtemdymo, statinio kodo perrašymo ir dinaminio kodo perrašymo.

Nematomų predikacijų taikymas

Nematoma predikacija yra tokių sąlygų programoje taikymas, kurių reikšmė yra iš anksto žinoma programos užtemdymo įrankiui. Naudojant šį metodą galimos kelios kodo transformacijos:

- Sąlygų išplėtimas – nematomas predikatas yra įterpiamas į ciklo sąlygą arba *if-else* instrukciją. Tokiu atveju įterpta sąlyga nepakeičia ciklo ar instrukcijos sąlygos reikšmės.
- Nereikalingų operandų pridėjimas – yra pridedama kintamųjų su nematomomis reikšmėmis prie programoje esamų aritmetinių instrukcijų [8].
- Bereikšmio kodo įterpimas – į programos kodą yra įterpiamas kodas, kurio vykdymo sąlyga yra tam tikras nematomas predikatas, tačiau šis yra aprašytas taip, kad sąlyga visad yra neigiama ir kodas programoje niekad nėra vykdomas.

Kodo užtemdymas naudojant netomus predikatus pavaizduotas 1.12 paveiksle. Paveiksle yra taikomas tiek sąlygų išplėtimas – nereikalinga bool sąlyga, pridėta nereikalinga aritmetinė operacija, bei įterpta kodo kuris nevykdomas.

```
if (arr[m] == x)
    return m+1;
```

a)

```
boolean bool;
int redop = 1;
if (arr[m] == x && !bool)
    if(bool){
        /*deadcode*/
        return 0;
    }
    return (m+1) * redop;
}
```

b)

1.12 pav. Kodo užtemdymas naudojant nematomus predikatus [8]. Pradinis kodas (a), užtemdytas kodas (b).

Bereikšmio kodo įterpimas

Šis metodas paremtas įvairios papildomos logikos įterpimu į programos kodą, kurie nekeičia programos vykdymo eigos, tai gali būt tiek įvairios sąlygos, kurios visada yra neigiamos, niekad nekviečiamos funkcijos ar tiesiog kintamieji, kurie nereikalingi programos vykdymui ir nekeičia jos rezultatų [8]. Bereikšmio kodo įterpimo pavyzdys pateiktas 1.13 paveiksle.

```
if (isPresent == false)
    System.out.println("Element
                    not present");
```

a)

```
if (isPresent == false) {
    String str="Hello World!";
    System.out.println("Element
                    not present");
}
```

b)

1.13 pav. Bereikšmio kodo įterpimas. Pradinis kodas (a), užtemdytas kodas (b). Į kodą įterptas papildomas kintamasis, kuris nedaro įtakos programos rezultatui. [8]

Eilės tvarkos užtemdymas

Naudojant šį metodą yra keičiama kodo išraiškų, ciklų, sąlygų ar metodų vykdymo tvarka išlaikant reikalingas priklausomybes [8].Šiam metodui galima išskirti tokias pagrindines strategijas:

- Išraiškų tvarkos užtemdymas – kai yra pakeičiama išraiškos sąlyga, išlaikant tokią reikšmę rezultate. Kodo fragmentas su eilės tvarkos užtemdymu pateiktas 1.14 paveiksle.

```
if (arr[m] < x)
    l = m + 1;
```

a)

```
if (x >= arr[m])
    l = m + 1;
```

b)

1. 14 pav. Kodo fragmente (b) yra pakeista *if* sąlygos tvarka išlaikant tokį patį funkcionalumą [8][16]

- Taip pat tvarkos užtemdymas gali būti naudojamas ir metodams ar kitiems kodo blokams užtemdyti.

- Ciklų skaičiavimo užtemdymas – kai yra pakeičiamos ciklo vykdymo sąlygos, nepakeičiant buvusio funkcionalumo. Ciklų sąlygų užtemdymo pavyzdys pateiktas 1.15 paveiksle.

```
for(int i=0;i<arr.length;i++)
    System.out.println(arr[i]);
```

a)

```
for(int i=arr.length-1;i>=0;i--)
    System.out.println(arr[i]);
```

b)

1.15 pav. Ciklų sąlygų pakeitimas. Pradinis kodas (a), užtemdytas kodas (b)

Programos vykdymo eigos išlyginimas

Naudojant šį kodo užtemdymo metodą yra taip pakeičiama programos vykdymo eiga, kad kiekvienas pagrindinis programos blokas turėtų tą patį pirmtaką ir tolimesnį programos tašką [16]. Programos vykdymo išlyginimui gali būti naudojama *switch* instrukcija ir tam tikras programos eigą kontroliuojantis kintamasis, bei *case* instrukcijos, kuriose butų skirtingi programos blokai. Programos vykdymo eigos užtemdymo pavyzdys pateiktas 1.16 paveiksle.

```
int arr[] = {2, 3, 4, 10, 40};
for (int i =0; i < arr.length; i++)
    System.out.println(arr[i]);
```

a)

```
int var=2;
while(var!=0){
 switch(var){
  case 1:
     for(int i =0;i<arr.length;i++)
       System.out.println(arr[i]);
     var=0;
     break;
  case 2:
     int arr[] = {2, 3, 4, 10, 40};
     var=1;
     break;
  default:break;
  }
}
```

b)

1.16 pav. Programų vykdymo eigos užtemdymas. Pradinis kodas (a), užtemdytas kodas (b) [8][16]

Programos funkcijų transformacijų metodai

Šie programos kodo užtemdymo metodai skirti užtemdyti programos funkcijų iškvietimą [8][16]. Naudojant šį metodą gali būti pasirenkami keli užtemdymo keliai. Vienas būdas yra vietose kur yra kviečiama funkcija, vietoj funkcijos kvietimo tiesiog įterpti kodą, kuris yra funkcijos bloke. Taip pat gali būti naudojama priešinga strategija, kai keletas programoje naudojamų instrukcijų yra įtraukiamos į kviečiama funkciją. Užtemdant metodų kvietimus programoje, gali būti naudojamas metodų klonavimas, kai programoje yra kviečiama funkcijos kopijos

1.6. Java programavimo kalba

Java yra aukšto lygio programavimo kalba, kuri gali veikti įvairiose platformose, nepriklausomai nuo naudojamos operacinės sistemos architektūros [24]. Tai yra pasiekiama naudojant Java virtualia mašiną, kurioje yra vykdomas programos kodas. Java kalba turi panašią sintaksę kaip C/C++ kalba, tačiau C kalba yra kompiliuojama į mašininį kodą skirta procesoriui, kuris yra sunkiau analizuojamas

ir tokio programoms yra sunkiau atlikti apgrąžos inžinerija, tuo tarpu Java kalba yra kompiliuojama į Java baitkodą, tinkančiu vykdyti Java virtualiai mašinai. Dėl to Java kalba parašytoms programoms turėtų būti taikomas kodo užtemdymas, siekiant padidinti programos atsparumą apgrąžos inžinerijai, dekompiliuojant programą ir bandant analizuoti pirminį programos kodą. Šiame projekte yra tirti kodo užtemdymą pasirinkta Java programavimo kalba, todėl toliau apžvelgiama pagrindinės Java kalbos savybės.

Java programavimo kalbos savybės

Java yra objektinė programavimo kalba, naudojanti klasių paveldimumo, bei abstrakcijos principus [24]. Objektai ir klasės yra vienos pagrindinių šios kalbos dalių. Java klasėse yra programos vykdomo funkcijos ir naudojami duomenys. Java kalba taip pat palaiko dinaminį bibliotekų naudojimą bei paralelinį procesų vykdymą programoje. Java kompiliatorius saugo programą nuo nereikalingo atminties sunaudojamo, pašalindamas sukurtus objektus, kurie programoje daugiau nėra naudojami. Taip pat Java virtuali mašina tikrina programos kodą ar nėra naudojamos draudžiamos operacijos, tokios kaip kreipimasis į atmintį, kuri nepriklauso vykdomai programai. Dėl Java virtualios mašinos naudojimo, Java programos gali būti kilnojamos iš vienos sistemos į kitą, kadangi yra nepriklausančios nuo platformos.

1.7. Analizės išvados

1. Šiuolaikinės programos susiduria su tokiomis grėsmėmis kaip apgrąžos inžinerija ir kodo klastojimas, piratavimas.
2. Vienas iš programų apsaugos metodų yra kodo užtemdymas, kuris apsunkina apgrąžos inžineriją, dėl ko iš dalies apsisaugoma ir nuo kitų grėsmių kaip kodo modifikavimas.
3. Kodo užtemdymui naudojamos įvairios kodo transformacijos, kurios padaro kodą sunkiau suprantamą, tačiau neaišku kaip skirtinga kodo užtemdymo metodų taikymo tvarka veikia kodo užtemdymo efektyvumą, todėl priimtas sprendimas tai ištirti.

2. Programų pirminio kodo užtemdymo metodo sudarymas

Apžvelgus literatūroje dažniausiai taikomus programų pirminio kodo užtemdymo metodus, darbe siūlomas kombinuotas užtemdymo metodų taikymas. Darbo tikslas atlikus tyrimą pagal kodo užtemdymo efektyvumo metrikas įvertinti, kada yra gaunami geresni rezultatai, taikant užtemdymo metodų kombinacijas skirtinga tvarka.

2.1. Kombinuoti pirminio programos kodo užtemdymo metodai

Šiame darbe pasiūlytas kombinuotas literatūroje nagrinėtų kodo užtemdymo metodų taikymas. Kombinuoto metodo sudarymui pasirinkta naudoti keturi dažniausiai naudojami programų kodo užtemdymo metodai, kurie bus atliekami Java programos kodui skirtinga tvarka. Tyrimui siūlomi kodo užtemdymo metodai pateikti 2.1 lentelėje.

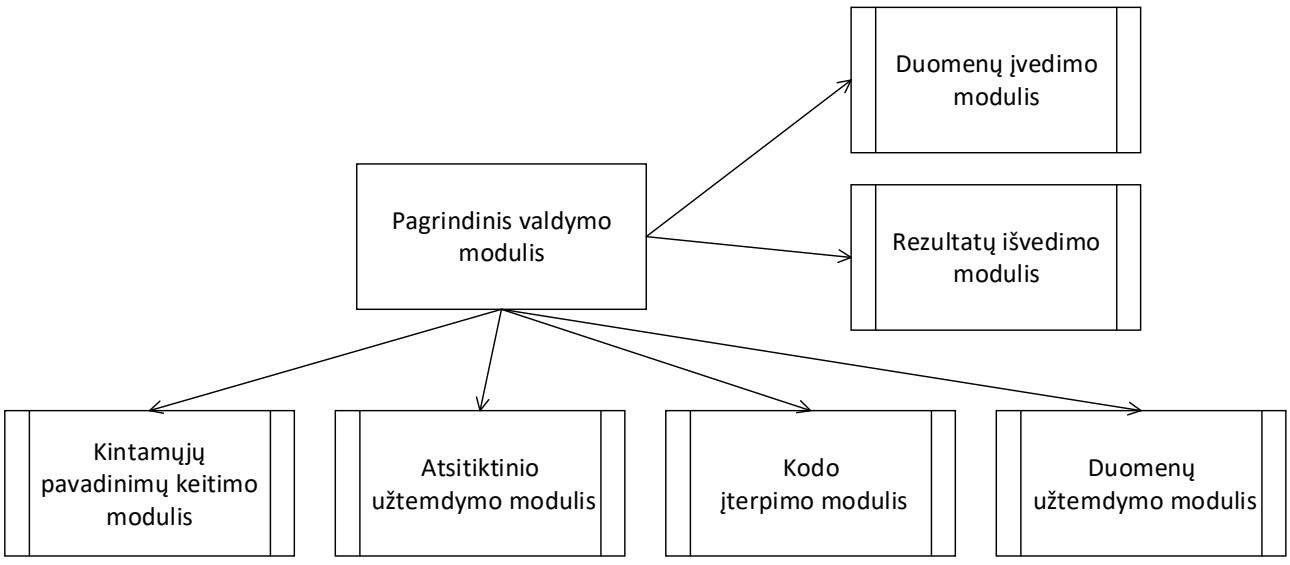
2.1 lentelė. Siūlomi kodo užtemdymo metodai

Užtemdymo metodas	**Užtemdymo metodų taikymo tvarka**
1 metodas	Duomenų užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas
2 metodas	Duomenų užtemdymas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas
3 metodas	Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas
4 metodas	Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas
5 metodas	Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas
6 metodas	Duomenų užtemdymas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas
7 metodas	Bereikšmio kodo įterpimas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas
8 metodas	Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas
9 metodas	Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Duomenų užtemdymas
10 metodas	Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas
11 metodas	Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas
12 metodas	Atsitiktinis užtemdymas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas
13 metodas	Atsitiktinis užtemdymas → Duomenų užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas
14 metodas	Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas
15 metodas	Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas
16 metodas	Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Bereikšmio kodo įterpimas

Užtemdymo metodas	Užtemdymo metodų taikymo tvarka
17 metodas	Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas
18 metodas	Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas
19 metodas	Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas
20 metodas	Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas
21 metodas	Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Duomenų užtemdymas → Bereikšmio kodo įterpimas
22 metodas	Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas
23 metodas	Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas
24 metodas	Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas

2.2. Programavimo kalbų pirminio kodo užtemdymo programos struktūra

Programų kodo užtemdymo įrankis susidaro iš kelių pagrindinių modulių: pirminio programos kodo nuskaitymo iš tekstinio failo; Kodo užtemdymo valdymo modulio, kuris atliktu užtemdymo transformacijas skirtinga tvarka, pagal pasirinktus parametrus; Rezultatų išvedimo modulio, kuris apskaičiuotų užtemdymo įverčius pagal žinomas kodo užtemdymo metrikas. Programavimo kalbų pirminio kodo užtemdymo programos struktūra pateikta 2.2 paveiksle.



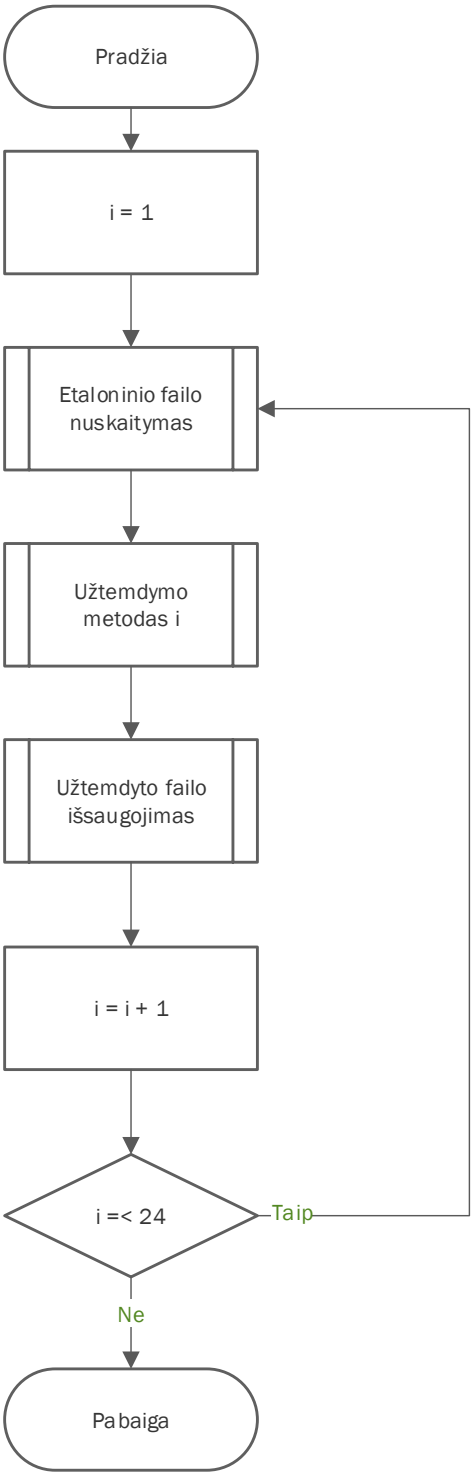
2.2 pav. Programavimo kalbų pirminio kodo užtemdymo programos struktūra

Programos funkcionalumas

• Kodo nuskaitymo modulis nuskaito tekstini failą, su programos pirminiu kodu ir pagal jį yra sugeneruojamas abstrakčios sintaksės medžio meta modelis kuris perduodamas kodo užtemdymo moduliui.

• Naudojant skirtingus užtemdymo modulius yra transformuojamas programos meta modelis, atliekant kodo užtemdymo transformacijas.
• Transformuotas kodas yra įrašomas į naują failą.

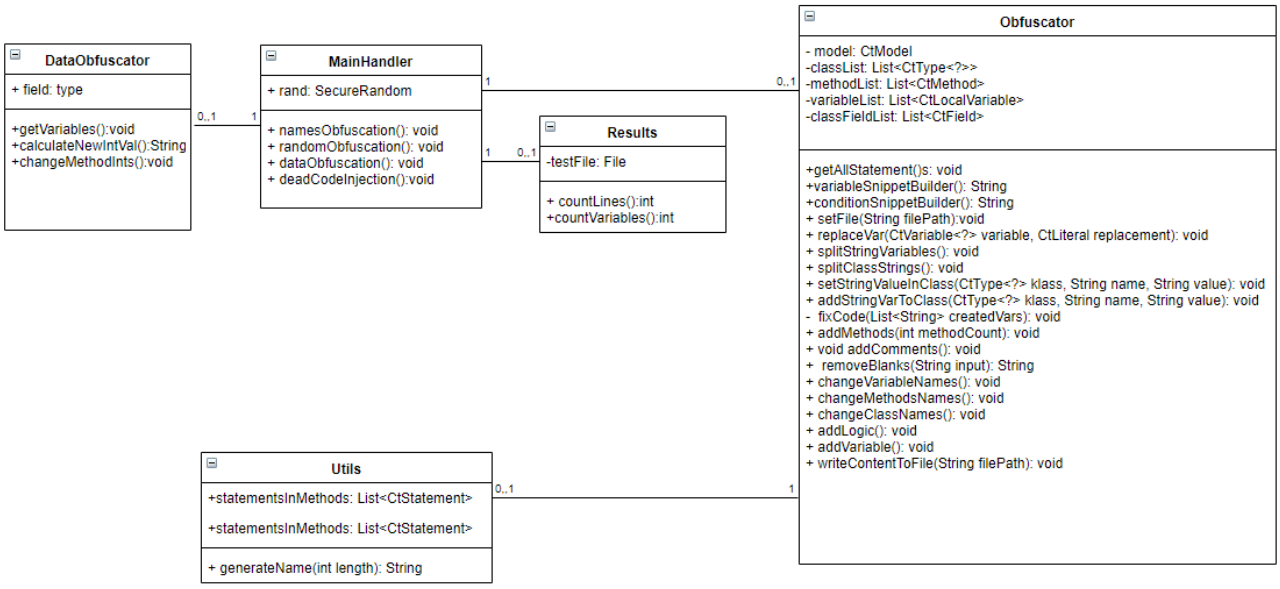
Projekte naudojamos programavimo kalbų pirminio kodo užtemdymo programos algoritmas pateiktas 2.3 paveiksle.



pav. 1

2.3 pav. Programavimo kalbų pirminio kodo užtemdymo programos algoritmas

Java kodo apdorojimui ir transformacijom naudojama papildoma Java Spoon biblioteka, kuri suteikia įrankius, reikalingus nuskaitytam iš failo kodui paversti į abstrakčios sintaksės medį, bei atlikti tam tikras jo modifikacijas. Programos prototipas sudarytas iš kelių klasių. Pagrindinė programos klasė *"Obfuscator"*, joje yra nuskaitomas programos kodo failas ir suformuojamas sintaksės medis. Tai pat naudojamos papildomos klasės, kuriose yra metodai kodo užtemdymo transformacijoms bei rezultatų skaičiavimui. Programavimo kalbų pirminio kodo užtemdymo programos prototipo klasių diagrama pateikta 2.4 paveiksle.



2.4 pav. Programavimo kalbų pirminio kodo užtemdymo programos prototipo klasių diagrama

2.3. Naudojami įrankiai

Šiame skyriuje apžvelgiami projekte reikalingi įrankiai ir jų savybės. Pagrindiniai įrankiai reikalingi Java kodo užtemdymo programos kūrimui ir jos efektyvumo tyrimui yra Java programavimo aplinka, pagalbinė biblioteka skirta Java kodo analizei, bei Java dekompiliatorius, kodo užtemdymo efektyvumo įvertinimui.

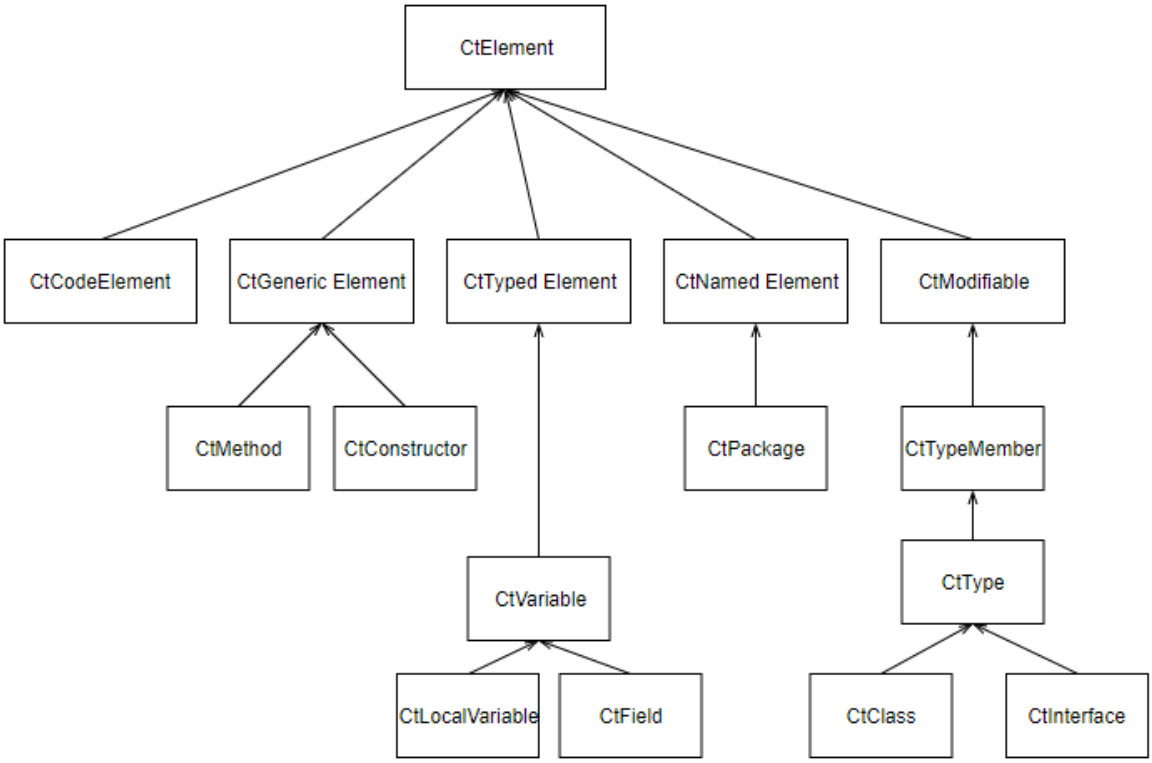
2.3.1. Java spoon biblioteka

Spoon biblioteka skirta padėti programų inžinieriams rašyti programos kodo analizės įrankius [27]. Biblioteka naudoja Eclipse JDT biblioteka kaip pagrindą, kartu pridėdama papildomų abstrakcijų ir funkcionalumo, kad kodo manipuliacija taptų lengvesnė. Spoon bibliotekos funkcionalumas leidžia filtruoti ir išrinkti kodo elementus. Šis funkcionalus gali būti naudojamas, norint surasti ir išrinkti tam tikrus kodo elementus kaip klases, deklaruotus metodus, kintamuosius, funkcijų iškvietimus. Kad būtų įmanoma kodo analizė yra sukuriamas programos meta modelis. Spoon biblioteka nuskaičius programos kodą, suformuoja abstrakčios sintaksės medį. Suformuotas sintaksės medis yra supaprastinamas į tokį, kuriuo būtų galima lengvai manipuliuoti. Programos modelis gali būti keičiamas, transformuojamas ir vėl perverčiamas į pirminį programos kodą.

Naudojant Spoon galimas kodo reorganizavimas, Spoon leidžia įterpti į esamą kodą naujus kodo elementus – metodus, kintamuosius. Taip pat biblioteka turi galimybę, keisti surinktų klasių pavadinimus, metodų pavadinimus ir lokalių kintamųjų pavadinimus bei tipą.

2.3.2. Java programos meta modelis

Naudojant Spoon biblioteką Java meta-modelis yra sukuriamas taip, kad būtu suprantamas Java programuotojų, kurie galėtų jį naudoti programų analizei, bei transformacijoms. Šis meta modelis savyje turi visa informaciją reikalingą išgauti sukompiliuojama Java kodą. Spoon meta-modelį galima išskaidyt į tris dalis. Struktūrinė dalyje yra aprašyti tokie programos elementai kaip klasės, metodai, kintamųjų deklaravimas, anotacijos. Struktūriniai Java Spoon modelio elementai pateikti 2.5 paveiksle.



2.5 pav. Struktūriniai elementai Java Spoon modelyje

Kodo dalyje yra programos loginėje dalis, Java vykdomasis kodas, kaip, kad aprašytas metodų blokuose. Tai gali būti įvairios logikos formuluotės, kintamųjų priskyrimai. Kodo elementai, kurie pasiekiami naudojant Spoon biblioteką pavaizduoti 21 paveiksle. Trečia Spoon meta modelio dalis sumodeliuoja nuorodas į programos elementus, vidines, bei trečiųjų šalių bibliotekas.

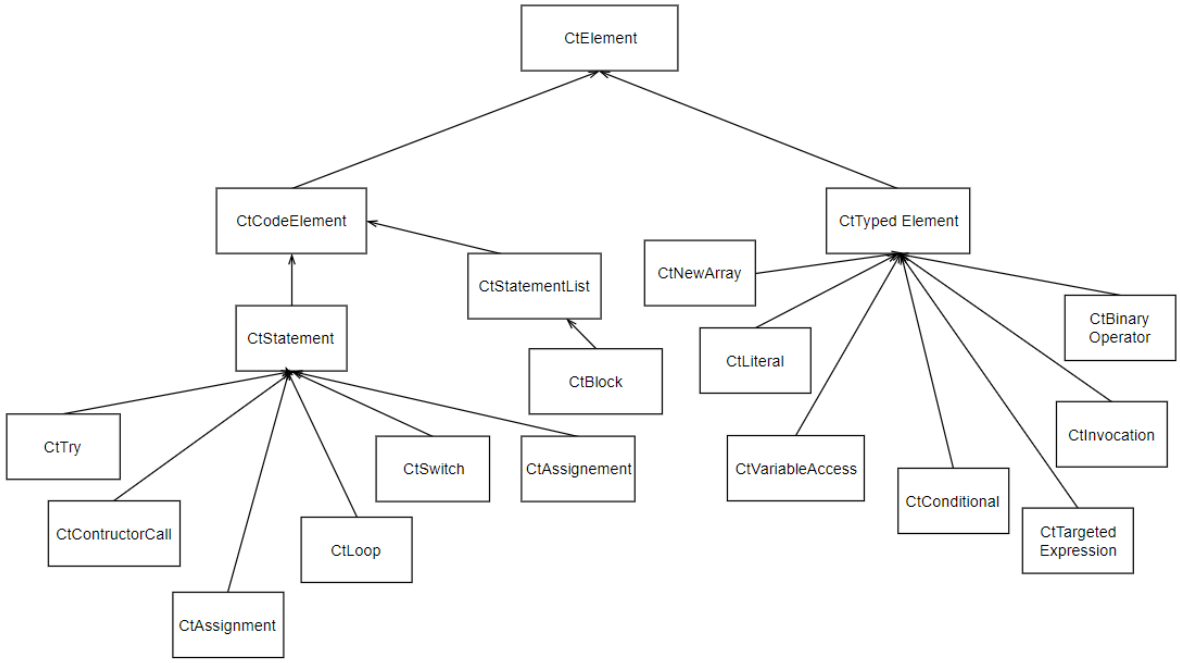
2.3.3. Java kodo modifikacijos naudojant Spoon biblioteką

Naudojant Spoon biblioteką yra galima modifikuoti kodo elementus, sukurti naujus kodo elementus ir įterpti juos į esamą Java kodą. Naudojant Spoon bibliotekos klasę *Factory* galima sukurti bet kurį meta modelio elementą [27], paveiksle 2.6 pateikiamas klasės kūrimo pavyzdys naudojant Java Spoon bibilioteką.

```
CtClass newClass = factory.Core().createClass();
```

2.6 pav. Klasės elemento sukūrimas naudojant Java Spoon biblioteką

Taip pat gali būti sukuriami metodai, kintamieji, ciklų blokai, kintamųjų priskyrimai, kodo intarpai. Sukurti modelio elementai gali būti įterpiami į meta modelį ir vėliau išvedami kaip programos kodas. Šis bibliotekos funkcionalumas keičiant esamus kodo elementus, bei sukuriant naujus kodo intarpus gali būti panaudojamas darbe, atliekant programos pirminio kodo užtemdymo transformacijas. Kodo elementai esantys Java Spoon modelyje pateikiami 2.7 paveiksle.



2.7 pav. Kodo elementai Java Spoon modelyje

2.3.4. Java Eclipse aplinka

Eclipse yra programų kūrimo programinis paketas susidedantis integruotos programavimo aplinkos, bei papildomų įrankių (*ang. plug-ins*), kurie suteikia paketui didesnį funkcionalumą [25]. Eclipse gali būti naudojamas programos kodo rašymui, kompiliavimui, bei dokumentacijos kūrimui. Šis paketas daugiausia yra naudojamas Java programų rašymui, tačiau taip pat gali palaikyti ir kitas programavimo kalbas, naudojant papildomus įrankius. Eclipse SDK (*ang. software development kit)* yra atviro kodo, todėl galima rasti nemažai įrankiu, kurie gali būti integruoti į ši paketą, taip pat ir Java Spoon analizės įrankį. Dėl šių priežasčių, bei todėl, kad programos kodo užtemdymo įrankis bus kuriamas naudojant Java programavimo kalba, šiame projekte yra pasirinkta naudoti Java Eclipse aplinką.

2.3.5. JD Project Java dekompiliatorius

Norint įvertinti kodo užtemdymo strategijos efektyvumą, bei patikrinti užtemdyto kodo atsparumą prieš automatinius apgrąžos inžinerijos įrankius, reikalingas dekompiliatorius, kuris galėtų grąžinti pirminį programos kodą iš užtemdytos programos. Šiame projekte pasirinktas naudoti JD Project Java dekompiliatorius. Šis įrankis tinka naudoti .class, .jar, .zip, failų analizei, bei taip pat turi grafinę sąsają ir taip pat galimybe būti integruotas į Eclipse aplinką. Tiriant užtemdytos programos atsparumą prieš dekompiliatorių, bus naudojami sukompiliuotos užtemdytos programos .jar tipo failai.

2.4. Kodo užtemdymo eksperimento scenarijus

Programų pirminio kodo užtemdymo metodo tyrimui planuojama naudoti etaloninį (*ang. benchmark*) failą. Tyrimo pradžioje planuojama įvertinti atskirai naudojamų kodo užtemdymo metodų efektyvumą, pagal žinomas kodo užtemdymo efektyvumo metrikas. Kodo užtemdymo efektyvumas yra apskaičiuojamas pagal analizėje minėtas metrikas – stiprumą, atsparumą, bei kainą. Darbe yra siūlomas kombinuotas užtemdymo metodas, atsitiktinai kombinuojant atskirus kodo užtemdymo metodus. Tyrimo metu yra apskaičiuojami įvairių kombinuotų metodų efektyvumo įverčiai, taip gaunant rezultatus, kaip skirtinga užtemdymo metodų naudojimo tvarka, keičia užtemdymo efektyvumą.

2.5. Etaloninis failas

Tiriant programos pirminio kodo užtemdymo strategijų efektyvumą, bus naudojamas etaloninis failas, visiems užtemdymo metodams. Šis failas yra parinktas naudojant literatūroje [18][26] siūlomus kodo fragmentus, skirtus vertinti kodo užtemdymo efektyvumą. Toliau paveiksluose 2.8 ir 2.9 pateikiami kodo užtemdymo testavimo failai rasti literatūroje [18].

```
public class TestOBF{
 public Hello hello=new Hello();
 protected String good="";
 private int[] number = new int[10];
 public static void main(String args[]){
 new TestOBF();
 }

 public TestOBF(){  System.out.println(hello.getHello("HELLO"));
 System.out.println(getGood());
 System.out.println("number="+getArray());
 }

 protected String getGood(){
 good="GOOD";
 return good;
 }
 private int getArray(){
 number[0]=1;
 return number[0];
 }
}
class Hello{
 public Hello(){
 int number=1;
 }
 public String getHello(String helloname){
 return helloname;
 }
}
```

2.8 pav. Literatūroje naudojamas etaloninis kodas užtemdymo efektyvumo tyrimui [18]

Taip pat kode yra naudojami ciklai, bei kelios algoritmų funkcijos rekomenduojamos kodo užtemdymo efektyvumui vertinti, pateiktos 2.10 paveiksle. Šiame projekte, kodo užtemdymo efektyvumo tyrimui naudojamas failas yra sudarytas apjungiant literatūros šaltiniuose rasta kodą, kuris pateiktas 2.8, 2.9 ir 2.10 paveiksluose ir taip gaunant vieną kodo failą, kuris bus naudojamas kodo užtemdymui.

```
public class TestControl{
 public static void main( String
args[] ){
 int loopNum=0;
 // test for loop construct
 for ( int i=0; i<2; i++ )
 loopNum++;
 // test if.. else statement
 if ( loopNum>1 )
 System.out.print( " loopNum>1"
);
 else
 System.out.println(          "
loopNum<=1" );
 // test whilt loop construct
 while ( loopNum>1 )
 loopNum=loopNum-1;
 System.out.println(          "
loopNum="+loopNum );

 }
}
```

2. 9 pav. Literatūroje naudojamas etaloninis kodas užtemdymo efektyvumo tyrimui [18]

```
public static void simpleProgram() {
      int i;
      int sum = 0;

      for (i = 0; i < 127; i++)
         sum += i;

      System.out.println("Sum is: " + sum + "\n");

   }

   static int fib(int n) {
      int a = 1;
      int b = 1;
      int i;
      for (i = 3; i <= n; i++) {
         int c = a + b;
         a = b;
         b = c;
      };
      return b;
   }
```

2. 10 pav. Programų kodo fragmentai siūlomi kodo užtemdymo testavimui [26]

Etaloninis failas sudarytas taip, kad galėtų matuoti kodo išdėstymo, programos vykdymo eigos, bei duomenų užtemdymo transformacijas. Testuojamas failas sudarytas iš klasės, kurioje yra keli kintamieji, bei klasės metodai. Suformuotas etaloninis failas, kuris bus naudojamas užtemdymo tyrimui pateiktas 2.11 paveiksle.

```
public class Test {
   //public Test hello = new Test();
   protected String good = "empty";
   private int[] number = new int[10];
   public Test() {
      System.out.println(getHello("HELLO"));
      System.out.println(getGood());
      System.out.println("number=" + getArray());
   }
   protected  String getGood() {
      return good;
   }
   private int getArray() {
      number[0] = 111;
      return number[0];
   }
   public void Hello() {
      int number = 789456;
   }
   public String getHello(String helloname) {
      return helloname;
   }
   public static void main(String args[]) {
      int loopNum = 0;
      new Test();
      // test methods
      System.out.println("FIB 5 :" + fib(5));
      simpleProgram();
      // test for loop construct
      for (int i = 0; i < 2; i++)
         loopNum++;
      // test if.. else statement
      if (loopNum > 1)
         System.out.print(" loopNum>1");
      else
         System.out.println(" loopNum<=1");
      // test while loop construct
      while (loopNum > 1)
         loopNum = loopNum - 1;
      System.out.println(" loopNum=" + loopNum);
   }
   public static void simpleProgram() {
      int i;
      int sum = 0;
      for (i = 0; i < 127; i++)
         sum += i;
      System.out.println("Sum is: " + sum + "\n");
   }
   static int fib(int n) {
      int a = 1;
      int b = 1;
      int i;
      for (i = 3; i <= n; i++) {
         int c = a + b;
         a = b;
         b = c;
      };
      return b;}
}
```

2. 11 pav. Suformuotas etaloninis failas kodo užtemdymo tyrimui

2.6. Naudojamos programos kodo transformacijos

Tiriant kodo užtemdymo metodų efektyvumą šiame projekte bus naudojami keturi kodo užtemdymo metodai: atsitiktinis užtemdymas, duomenų užtemdymas, bereikšmio kodo įterpimas, bei kintamųjų vardų, bei funkcijų pavadinimų keitimas. Naudojant atsitiktinį užtemdymą yra įterpiama papildomų kintamųjų, komentarų, pašalinami buvę komentarai, pakeičiamas kodo formatavimas. Duomenų užtemdymui pasirinkta taikyti eilutės išskaidymą į keletą kintamųjų, bei sveikų skaičių kintamųjų reikšmių užtemdymą. Kodo įterpimui naudojami keli Java kodo fragmentai su metodais, kuriuose naudojami sąlyginiai *if* , bei ciklų blokai. Naudojant pavadinimų užtemdymą yra pakeičiami kode esančių kintamųjų, bei funkcijų pavadinimai.

2.7. Kodo užtemdymo efektyvumo įvertinimas

Vertinant tiriamų kombinuotų užtemdymo metodų efektyvumą projekte naudojamos prieš tai nagrinėtoje literatūroje [1][15] aptartos kodo užtemdymo efektyvumo įvertinimo metrikos. Pradžioje atskirai apskaičiuojamas kiekvieno užtemdymo metodo efektyvumas, toliau yra paskaičiuojamas kombinuotų užtemdymo metodų efektyvumas. Vertinant kodo užtemdymo efektyvumo metrikas naudojami literatūroje [15] siūlomi įverčių svorių koeficientai.

Skaičiuojant kodo efektyvumą pirmiausiai yra paskaičiuojami trys kodo užtemdymo efektyvumo įverčiai: stiprumas, atsparumas, bei naudojamų resursų kaina. Kodo stiprumas apskaičiuojamas pagal 1.3.2 skyrelyje aprašytas formules, reikšmės reikalingos įverčio apskaičiavimui apskaičiuojamos rankiniu būdu. Kodo atsparumui vertinti naudojamas JD Project dekompiliatorius, kuriuo yra bandoma išgauti pirminį programos kodą iš sukompiliuotos programos, grąžintas kodas yra vertinamas pagal 1.3.3 skyrelyje pateiktas metrikas. Naudojamos resursų kainos įvertis vertinamas pagal 1.3.4 skyrelyje aprašytas metrikas. Programos naudojama atmintis, bei užimama vieta, apskaičiuojama rankinių būdu, programos vykdymo greitis apskaičiuojamas naudojant papildomą funkciją programos kode, kuri išveda programos vykdymo laiką. Programos sunaudojama atmintis bus vertinama naudojant *YourKit Java profiler* įrankį, kuris analizuoja programos metu sunaudotus resursus ir gali parodyt kiek buvo naudojama *heap* ir *neheap* atminties.

2.8. Išvados

- Pasiūlytas kombinuotas kodo užtemdymo metodas.
- Sukurta programinės įrangos architektūra ir prototipas tyrimams atlikti.
- Remiantis etaloniniais programų kodais kombinuoto užtemdymo metodo tyrimui pasiūlytas etaloninės programos pirminis kodas.

3. Programų pirminio kodo užtemdymo metodų tyrimas

Šiame skyriuje aprašomas programų pirminio kodo užtemdymo metodų kombinuojant įvairias kodo užtemdymo transformacijas tyrimo scenarijus, bei pateikiami gauti rezultatai.

Tyrimui naudojamas failas

Tyrimui naudojamas etaloninis failas sudarytas iš algoritmų, kurie yra siūlomi naudoti kodo užtemdymo testavimui [26], bei dviejų etaloninių failų rastų literatūros šaltiniuose [18] skirtuose kodo užtemdymo testavimui. Visi kodo fragmentai yra sudėti į vieną tekstinį Java failą. Kadangi rasti užtemdymo tyrimams skirti failai buvo parašyti C++ kalba, o tyrime bus naudojama Java kalba, sudarant tyrimui reikalinga failą buvo reikalinga išversti kodą iš C++ į Java kalbą.

Tyrimui naudojamo failo turinys

Tyrime naudojame tekstiniame java faile yra tokie kodo elementai:

- Pagrindinė klasė pavadinimu „Test“.
- „Test“ klasė turi du kintamuosius – eilute pavadinimu „*good“*, kurios reikšmė „empty“, bei dešimties sveikų skaičių masyvą pavadinimų *number*.
- „Test“ klasėje yra konstruktorius bei septyni klasės metodai.
- Du metodai iš algoritmų skirtų kodo užtemdymo testavimui – fibonačio n elemento paskaičiavimo funkcija, bei sumą skaičiuojanti bet rezultatą išvedanti funkcija.
- Kiti klasės metodai grąžina arba modifikuoja klasėje esančius kintamuosius.
- Faile iš viso yra vienuolika kintamųjų.
- Naudojami trys *for* ciklai ir vienas *while* ciklas.
- Iš viso faile 76 eilutės.

3.1. Tyrimo eiga

Atliekant tyrimą, paruoštas testinis failas pradžioje yra užtemdomas atskirai naudojant keturis užtemdymo metodus: atsitiktinį užtemdymą, bereikšmio kodo įterpimą, duomenų užtemdymą, bei funkcijų bei kintamųjų pavadinimų užtemdymą. Tyrimo metu naudoto kompiuterio specifikacijos pateikiamos 3.1 lentelėje

3.1 lentelė. Tyrime naudoto kompiuterio specifikacija

Procesorius	Interl Core i7-6700HQ 2,60GHz
Operatyvioji atmintis	8GB
Kietasis diskas	Samsung SSD 512GB
Grafinė plokštė	NVIDIA GeForce GTX 960M
Operacinė sistema	Windows 10 64-bit

3.1.1. Atsitiktinis užtemdymas

Naudojant šį užtemdymo metodą projekte naudojamas užtemdymo prototipas į pradinį programos kodą atsitiktinai įterpia papildomų kintamųjų, komentarų, bei nereikšmingos programos logikos. Užtemdyta programa yra sukompiliuojama, patikrinamas jos vykdymo laikas. Turint užtemdyto kodo tekstinį failą yra paskaičiuojami duomenys reikalingi kodo užtemdymo efektyvumui įvertinti. Kodo užtemdymo efektyvumas apskaičiuojamas pagal 1.3 skyriuje aprašytas vertinimo metrikas. Kodo efektyvumui apskaičiuoti naudojame formulę (3.1):

$$S_{efek} = 0{,}4 \cdot S_{stipr} + 0{,}6 \cdot S_{atsp} - S_{kain} \quad (3.1)$$

Kodo užtemdymo efektyvumo skaičiavimo formulė. S_{efekt} – užtemdymo efektyvumo įvertis, S_{atsp} – užtemdymo atsparumo įvertis, S_{kain} – naudojamų resursų kainos įvertis

Pagal minėtą formulę (3.1) norint gauti kodo užtemdymo efektyvumo įvertį, pirmiausia apskaičiuojami stiprumo, atsparumo, bei resursų kainos įverčiai.

Stiprumo įverčio apskaičiavimas

Stiprumo įvertis susideda iš devynių dedamųjų, tačiau šiame tyrime analizuojant Java kalbos užtemdyme nebuvo naudojami programos šuoliai su *„go-to“* instrukcijomis, bei sąlygų pavadinimai (*ang. labels)*, todėl 1.3 skyriuje pateikta formulę galima supaprastinti. Šiame tyrime stiprumo įvertis skaičiuojamas pagal formulę (3.2).

$$S_{stipr} = 12{,}5 \cdot S_{cikl} + 6{,}25 \cdot S_{neskait} + 6{,}25 \cdot S_{kart} + \\ +6{,}25 \cdot S_{ekstra} + 6{,}25 \cdot S_{pras} + 12{,}5 \cdot S_{enc} + 12{,}5 \cdot S_{ilg} \quad (3.2)$$

Stiprumo įverčio skaičiavimo formulė. S_{stipr} – užtemdymo stiprumo įvertis, S_{cikl} – ciklų įvertis, $S_{neskait}$ – neskaitomų eilučių įvertis, S_{kart} – pasikartojančių kintamųjų įvertis, S_{ekstra} – papildomų kintamųjų įvertis, S_{pras} – prasmingų pavadinimų įvertis, S_{enc} – šifruotų kintamųjų įvertis, S_{ilg} – programos ilgio įvertis.

Ciklų sudėtingumo įverčio apskaičiavimas

Norint apskaičiuoti ciklų sudėtingumo įverti, paskaičiuojamas ciklų įvertis pradinėje programoje ir užtemdytoje programoje. Tam reikia žinoti ciklų kiekį, bei jų gylį abiejose programose. Ciklų įverčius paskaičiuojame pagal formulę (3.3)

$$S_c = \sum_{i=1}^{N} lygis_i \cdot kiekis_i \quad (3.3)$$

Ciklų įverčio skaičiavimo formulė. S_c – užtemdymo stiprumo įvertis, N – didžiausias ciklo gylis programoje, $lygis_i$ – ciklo *i* lygis, $kiekis_i$ – ciklų kiekis gylyje *i*.

Pradinėje programoje turime keturis ciklus, visi ciklai yra pirmo lygio. Užtemdytame kode yra aštuoni ciklai, visi pirmo lygio. Žinodami ciklų kiekį ir gylį naudojame (3.3) formulę ir apskaičiuojame ciklų įverti pradinėje, bei užtemdytoje programoje.

3.2 lentelė. Ciklų vertinimo metrikos

	Pradinis kodas	Užtemdytas kodas
Ciklų kiekis	4	1
Didžiausias ciklų gylis	8	1

$$Pradinio\ S'_c = 1 \cdot 4 = 4$$
$$Užtemdytos\ S_c = 1 \cdot 8 = 8$$

Turėdami pradinės ir užtemdytos programos ciklų įverčius, galima apskaičiuoti užtemdymo metodo ciklų sudėtingumo įvertį pagal formulę (3.4):

$$S_{cikl} = \frac{S_c - S'_c}{S'_c} = \frac{8-4}{4} = 1 \quad (3.4)$$

S_{cikl} – ciklų gylio sudėtingumo įvertis, S'_c – ciklų gylio įvertis pradinėje programoje, S_c – ciklų gylio įvertis užtemdytoje programoje.

Neskaitomų eilučių įvertis

Šis įvertis nurodo kiek užtemdyto kodo eilučių dekompiliatorius nesugebėjo grąžinti. Įverčiui apskaičiuoti reikia žinoti kodo eilučių skaičių užtemdytoje programoje ir negrąžintų eilučių skaičių. Naudojant šį užtemdymo metodą, dekompiliatorius grąžino visas kodo eilutes. Užtemdytas kodo failas turi 182 eilutes. Turint šias reikšmes neskaitomų eilučių įvertį apskaičiuojame pagal (3.5) formulę:

$$S_{neskait} = \frac{c}{c'} = \frac{0}{182} = 0 \quad (3.5)$$

$S_{neskait}$ – neskaitomumo įvertis, c – negrąžinamų eilučių skaičius, c' – eilučių skaičius užtemdytoje programoje

Pasikartojančių kintamųjų įvertis

Šis įvertis nurodo koks procentas kintamųjų užtemdytame programos kode turi tokius pačius pavadinimus, tačiau skirtingas reikšmes. Norint apskaičiuoti šį įvertį reikia žinoti visų kintamųjų skaičių užtemdytame kode, bei pasikartojančių kintamųjų skaičių. Turint reikšmes įvertis apskaičiuojamas pagal formulę (3.6):

3.3 lentelė. Pasikartojančių kintamųjų skaičius programos kode

	Pradinis kodas	Užtemdytas kodas	
Kintamųjų skaičius	11	29	
Pasikartojančių kintamųjų skaičius	-	6	

$$S_{kart} = \frac{k_{kart}}{k'} = \frac{6}{29} = 0.2068 \quad (3.5)$$

S_{kart} – pasikartojančių kintamųjų įvertis, k' – visų kintamųjų skaičius užtemdytoje programoje,
k_{kart} – pasikartojančių kintamųjų skaičius.

Papildomų kintamųjų įvertis

Papildomų kintamųjų įvertis nurodo koks procentas kintamųjų užtemdytame kode turi skirtingus pavadinimus, bet tas pačias reikšmes. Žinant kiek kintamųjų turi tas pačias reikšmes ir visą kintamųjų skaičių, ši įvertį galime paskaičiuoti pagal formulę (3.7):

3.4 lentelė. Papildomų kintamųjų skaičius programos kode

	Pradinis kodas	Užtemdytas kodas
Kintamųjų skaičius	11	29
Papildomų	-	1

$$S_{ekstra} = \frac{k_{ekstra}}{k'} = \frac{1}{29} = 0.0344 \quad (3.7)$$

S_{ekstra} – papildomų kintamųjų įvertis, k' – visų kintamųjų skaičius užtemdytoje programoje, k_{ekstra} – papildomų kintamųjų skaičius.

Prasmingų kintamųjų įvertis

Šis įvertis nurodo ar kodo užtemdymo metodas pakeitė kintamųjų vardus į tokius, kurie neturi jokios reikšmės. Įvertis gali turėti dvi reikšmes 0 – jei vardai nepakeisti, bei 1 – jei transformacija vardus pakeitė į nereikšmingus. Naudojant šį metodą kintamųjų vardai buvo nepakeisti, todėl:

$$S_{pras} = 0$$

S_{pras} – prasmingų kintamųjų pavadinimų įvertis

Šifruotų kintamųjų įvertis

Šio įverčio reikšmė priklauso nuo to ar kintamieji užtemdant kodą buvo šifruojami ir kur yra aptinkamas arba neaptinkamas jų iššifravimo metodas. Šiame tyrime pasirinktos kodo užtemdymo transformacijos, neatlieka kintamųjų šifravimo, todėl šis įvertis lygus nuliui.

$$S_{enc} = 0$$

S_{enc} – šifruotų kintamųjų įvertis

Programos ilgio įvertis

Apskaičiavimui kiek kodo užtemdymo metodas pakeitė programos ilgį skaičiuojamas kodo eilučių tarp pradinės programos ir užtemdyto programos kodo. Programos ilgio įvertis apskaičiuojamas pagal formulę (3.8)

3.5 lentelė. Eilučių skaičius programos kode

	Pradinis kodas	Užtemdytas kodas
Eilučių skaičius	76	183

$$S_{ilg} = \frac{i}{i'} = \frac{76}{183} = 1{,}4078 \quad (3.8)$$

S_{ilg} – programos ilgio įvertis, i' – eilučių skaičius užtemdytoje programoje, i – eilučių skaičius pradinėje programoje.

Apskaičiavę visus reikalingus įverčius galima suskaičiuoti šio metodo kodo užtemdymo stiprumą, naudojant (3.2) formulę:

$$S_{stipr} = 12{,}5 \cdot 1 + 6{,}25 \cdot 0 + 6{,}25 \cdot 0{,}20689 + 6{,}25 \cdot 0{,}03448 + 6{,}25 \cdot 1 + 12{,}5 \cdot 0 + \\ + 12{,}5 \cdot 1{,}4078 = 31{,}6073$$

Atsparumo įvertis

Atsparumo įvertis gaunamas pagal taip kaip sukompiliuotas programos kodas atsilaikė prieš dekompiliavimo įrankius. Šiuo atveju tyrime naudotas dekompiliatorius JD Project Java pateikus sukompiliuotą programą, pirminį programos kodą grąžino be klaidų, todėl atsparumo įvertis lygus nuliui.

$$S_{atsp} = 0$$

S_{atsp} – užtemdyto kodo atsparumo įvertis

Resursų kainos įvertis

Skaičiuojant šį įverti reikalinga žinoti programos vykdymo laiką, sunaudojamos atminties kiekį, bei užimamą vietą. Tuomet resursų kainos įvertis apskaičiuojamas pagal formulę (3.9):

$$S_{kain} = 0{,}4 \cdot S_m + 0{,}14 \cdot S_{viet} + 0{,}45 \cdot S_{laik} \quad (3.9)$$

S_{kain} – resursų kainos įvertis, S_m – atminties įvertis, S_{viet} – užimamos vietos įvertis, S_{laik} – laiko įvertis

3.6 lentelė. Naudojamos atminties įverčiai

	Pradinė programa	Užtemdyta programa
Vidutinė naudojama heap atmintis (MB)	7,2	6,5
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Programos užimama vieta (KB)	1,59	2,1
Programos vykdymo laikas (ns)	2412700	4842400

Turint visus duomenis reikalingus kainos įverčio apskaičiavimui, pradžiai suskaičiuojami tarpiniai įverčiai pagal (3.10) ir (3.11) formules:

$$S_{heap} = \frac{m'_{heap} - m_{heap}}{m_{heap}} = \frac{6{,}5 - 7{,}2}{7{,}2} = -0{,}09722 \quad (3.10)$$

S_{heap} –naudojamos heap atminties įvertis, m'_{heap} – naudojama heap atmintis užtemdytoje programoje, m_{heap} – naudojama heap atmintis pradinėje programoje.

$$S_{ne-heap} = \frac{m'_{ne-heap} - m_{ne-heap}}{m_{ne-heap}} = \frac{9{,}2 - 9{,}2}{9{,}2} = 0 \quad (3.11)$$

$S_{ne\text{-}heap}$ –naudojamos ne *heap* atminties įvertis, $m'_{ne\text{-}heap}$ – naudojama ne *heap* atmintis užtemdytoje programoje, $m_{ne\text{-}heap}$ – naudojama ne *heap* atmintis pradinėje programoje.

Apskaičiavus šiuos atminties įverčius, bendras naudojamos atminties yra paskaičiuojamas pagal formulę:

$$S_m = 0{,}375 \cdot S_{ne-heap} + 0{,}625 \cdot S_{heap} = 0{,}625 \cdot (-0{,}09722) = -0{,}06076$$

S_m– naudojamos atminties įvertis, $S_{ne\text{-}heap}$ –naudojamos ne heap atminties įvertis, S_{heap} –naudojamos heap atminties įvertis

Naudojamos vietos įvertis parodo kiek pasikeitė programos dydis lyginant su pradine programa ir yra apskaičiuojamas pagal formulę (3.12):

$$S_{viet} = \frac{v' - v}{v} = \frac{2{,}1 - 1{,}59}{1{,}59} = 0{,}32075 \quad (3.12)$$

S_{laik}– užimamos vietos įvertis, v' – užtemdytos programos užimama vieta,
v – pradinės programos užimama vieta

Vykdymo laiko įvertis parodo kiek pasikeitė programos vykdymo laikas lyginant su pradine programa ir yra apskaičiuojamas pagal formulę (3.13):

$$S_{laik} = \frac{t' - t}{t} = \frac{5629200 - 2412700}{2412700} = 1{,}333153728 \quad (3.13)$$

S_{laik}– programos vykdymo laiko įvertis, t' – užtemdytos programos vykdymo laikas, t – pradinės programos vykdymo laikas

Apskaičiavus tarpinius įverčius galima įvertinti bendra užtemdytos programos naudojamų resursų kainos įvertį, naudojant (3.9) formulę:

$$S_{kain} = 0{,}4 \cdot (-0{,}06076) + 0{,}15 \cdot 0{,}32075 + 0{,}45 \cdot 1{,}33315 = 0{,}62372683$$

Bendras užtemdymo efektyvumo apskaičiavimas
Apskaičiavus stiprumo, atsparumo, bei resursų kainos įverčius, galima apskaičiuoti bendra užtemdymo efektyvumą pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot S_{stipr} + 0{,}6 \cdot S_{atsp} - S_{kain} = 0{,}4 \cdot 31{,}6073 + 0 - 0{,}62372683 = 12.01919513$$

3.1.2. Bereikšmio kodo įterpimas

Naudojant šį kodo užtemdymo metodą į pradinį kodą yra įterpiama bereikšmio kodo kuris nėra vykdomas. Atsitiktinai į kodą įterpiami metodai, kuriuose yra kintamųjų, sąlygų, ciklų, kurie nekeičia programos rezultato. Panaudojus bereikšmio kodo įterpimo metodą yra suskaičiuojami reikalingi duomenys, bei apskaičiuojami efektyvumo įverčiai taip pat kaip prieš tai aprašytam metodui. Toliau lentelėse pateikiami šio metodo kodo užtemdymo efektyvumo įverčiai.

3.7 lentelė. Bereikšmio kodo įterpimo metodo stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	6
Ciklų gylio sudėtingumo įvertis	1
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	18
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	5
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,2777
Papildomų kintamųjų įvertis	0,111
Prasmingų kintamųjų įvertis	0
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	169
Programos ilgio įvertis	1,2236
Bendras stiprumo įvertis	**23.97660819**

Šis užtemdymo metodas nedarė pakeitimų kintamųjų varduose, bei nenaudojo kintamųjų šifravimo, todėl šie įverčiai lygus nuliui. Taip dekompiliatorius grąžino visas kodo eilutes, todėl neskaitomumo įvertis taip lygus nuliui. Šiuo atveju stiprumui didžiausią įtaką darė padidėjęs programos ilgis.

Suskaičiavus reikalingas vertes kodo užtemdymui stiprumui vertinti, bendras šio metodo stiprumo įvertis apskaičiuojamas pagal (3.2) formulę:

$$S_{stipr} = 12{,}5 \cdot 1 + 6{,}25 \cdot 0 + 6{,}25 \cdot 0{,}2777 + 6{,}25 \cdot 0{,}111 + 6{,}25 \cdot 0 + 12{,}5 \cdot 0 + \\ + 12{,}5 \cdot 1{,}2236 = 23{,}97660$$

Sukompiliuotą programa bandant analizuoti įrankiu „JD Project Java decompiler“ gauname tokį pat pradinį kodą koks ir buvo prieš kompiliaciją, taigi įrankis kodą grąžino be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Naudojant šį užtemdymo metodą, nebuvo pokyčių programos sunaudojamoje atmintyje, tačiau padidėjo programos vykdymo laikas, bei užimama vieta. Naudojamų resursų kainos įverčiai pateikiami 8 lentelėje.

3.8 lentelė. Bereikšmio kodo įterpimo metodo naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	3,03
Naudojamos vietos įvertis	0,9056	
Programos vykdymo laikas (ns)	2412700	3247900
Vykdymo laiko įvertis	0,346168193	
Bendras resursų kainos įvertis	0,291624744	

Suskaičiavus reikalingas vertes kodo užtemdymo metodo naudojamų resursų kainai vertinti, bendras šio metodo resursų kainos įvertis apskaičiuojamas pagal (3.9) formulę:

$$S_{kain} = 0{,}4 \cdot (0) + 0{,}15 \cdot 0{,}9056 + 0{,}45 \cdot 0{,}346168193 = 0{,}291624744$$

Turint stiprumo, atsparumo, bei resursų kainos įverčius, bendra užtemdymo metodo efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 23.97660819 + 0{,}6 \cdot 0 - 0{,}29162 = 9{,}299018531$$

3.1.3. Duomenų užtemdymas

Naudojant šį metodą kode esantys *String* tipo kintamųjų reikšmės buvo išskaidytos į keletą dalių ir priskirtos naujai sukurtiems kintamiesiems, o pirminė kintamojo reikšmė tampa šių naujai sukurtų kintamųjų junginys. Sveikų skaičių tipo kintamųjų reikšmės buvo pakeistos matematinėmis funkcijomis, bei perverčiami į dvejetainį formatą. Kadangi šis metodas nedaug pakeičia programos ilgį, nekeičia kintamųjų pavadinimų, bei neprideda papildomų ciklų, pagal tyrime naudojamas metrikas, metodas turi žemesnį stiprumo įvertį, nei prieš tai tirti metodai. Toliau lentelėse pateikiami duomenų užtemdymo metodo kodo užtemdymo efektyvumo įverčiai:

3.9 lentelė. Duomenų užtemdymo metodo stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	4
Ciklų gylio sudėtingumo įvertis	0
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	14
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	3
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,214285
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	0
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	79
Programos ilgio įvertis	0,039473
Bendras stiprumo įvertis	**1,8327067**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Duomenų užtemdymo metodo naudojamų resursų kainos įverčiai pateikiami 10 lentelėje:

3.10 lentelė. Duomenų užtemdymo metodo naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	1,89
Naudojamos vietos įvertis	0,188679	
Programos vykdymo laikas (ns)	2412700	5325000
Vykdymo laiko įvertis	1,207070	
Bendras resursų kainos įvertis	0,571483	

Bendras duomenų užtemdymo metodo efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 1{,}832706767 + 0{,}6 \cdot 0 - 0{,}571483799 = 0.161598908$$

3.1.4. Funkcijų ir kintamųjų pavadinimų užtemdymas

Šis metodas pakeičia klasėje esančių funkcijų bei kintamųjų pavadinimus į atsitiktinai sugeneruotus. Šiame tyrime yra nekeičiamas pačios klasės pavadinimas, bei *main* funkcija, kadangi pakeitus šiuos pavadinimus, programa negrąžindavo tokių pat rezultatų kaip pradinė. Toliau lentelėse pateikiami kintamųjų pavadinimų užtemdymo metodo kodo užtemdymo efektyvumo įverčiai:

3.11 lentelė. Funkcijų ir kintamųjų pavadinimų užtemdymo metodo stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	4
Ciklų gylio sudėtingumo įvertis	0

Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	11
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	3
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,272727273
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	76
Programos ilgio įvertis	0
Bendras stiprumo įvertis	**7,954545455**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas taip pat grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Funkcijų ir kintamųjų pavadinimų užtemdymo metodo naudojamų resursų kainos įverčiai pateikiami 12 lentelėje:

3.12 lentelė. Funkcijų ir kintamųjų pavadinimų užtemdymo metodo naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	1,7
Naudojamos vietos įvertis	0,06918239	
Programos vykdymo laikas (ns)	2412700	5878600
Vykdymo laiko įvertis	1,436523397	
Bendras resursų kainos įvertis	0,656812887	

Bendras funkcijų ir kintamųjų pavadinimų užtemdymo metodo efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 7{,}954545455 + 0{,}6 \cdot 0 - 0{,}65681288 = 2{,}525005295$$

3.1.5. Kombinuoti užtemdymo metodai

Ištyrus keturių atskirai naudojamų metodų užtemdymo efektyvumą, toliau yra tiriama kombinuoti užtemdymo metodai. Šie užtemdymo metodai gauti kombinuojant ir taikant prieš tai nagrinėtus užtemdymo metodus, skirtinga tvarka.

Metodas Nr.1 – Duomenų užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas

3.13 lentelė. Kombinuoto metodo nr.1 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13

Ciklų gylio sudėtingumo įvertis	2,25
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	55
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	3
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,054545
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	333
Programos ilgio įvertis	3,3815789
Bendras stiprumo įvertis	**76,98564593**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.1 naudojamų resursų kainos įverčiai pateikiami 3.14 lentelėje:

3.14 lentelė. Kombinuoto metodo nr.1 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,482
Naudojamos vietos įvertis	1,8188679	
Programos vykdymo laikas (ns)	2412700	653700
Vykdymo laiko įvertis	1,70390019	
Bendras resursų kainos įvertis	1,0395827	

Bendras kombinuoto metodo nr.1 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 76{,}98564593 + 0{,}6 \cdot 0 - 1{,}0395827 = 29{,}7546731$$

Metodas Nr.2 – Duomenų užtemdymas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas

Kombinuoto metodo nr.2 stiprumo įverčiai pateikiami 3.15 lentelėje:

3.15 lentelė. Kombinuoto metodo nr.2 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	11
Ciklų gylio sudėtingumo įvertis	1,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	36
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11

Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	244
Programos ilgio įvertis	2,210526316
Bendras stiprumo įvertis	**55,75657895**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.2 naudojamų resursų kainos įverčiai pateikiami 3.16 lentelėje:

3.16 lentelė. Kombinuoto metodo nr.2 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	3,73
Naudojamos vietos įvertis	1.34591195	
Programos vykdymo laikas (ns)	2412700	6113100
Vykdymo laiko įvertis	1,533717412	
Bendras resursų kainos įvertis	0,892059628	

Bendras kombinuoto metodo nr.2 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 55{,}75657895 + 0{,}6 \cdot 0 - 0{,}892059628 = 21{,}41057195$$

Metodas Nr.3 – Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas

Kombinuoto metodo nr.3 stiprumo įverčiai pateikiami 3.17 lentelėje:

3.17 lentelė. Kombinuoto metodo nr.3 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	52
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	8
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,153846154
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76

Kodo eilučių skaičius užtemdytoje programoje	374
Programos ilgio įvertis	3,921052632
Bendras stiprumo įvertis	**84,34969636**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.3 naudojamų resursų kainos įverčiai pateikiami 3.18 lentelėje:

3.18 lentelė. Kombinuoto metodo nr.3 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	3,99
Naudojamos vietos įvertis	1,509433962	
Programos vykdymo laikas (ns)	2412700	6563100
Vykdymo laiko įvertis	1,720230447	
Bendras resursų kainos įvertis	1,000518796	

Bendras kombinuoto metodo nr.3 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 84{,}34969636 + 0{,}6 \cdot 0 - 1{,}000518796 = 32{,}73935975$$

Metodas Nr.4 – Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas

Kombinuoto metodo nr.4 stiprumo įverčiai pateikiami 3.19 lentelėje:

3.19 lentelė. Kombinuoto metodo nr.4 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	7
Ciklų gylio sudėtingumo įvertis	0,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	51
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	2
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,039215686
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	268
Programos ilgio įvertis	2,526315789
Bendras stiprumo įvertis	**47,44904541**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.4 naudojamų resursų kainos įverčiai pateikiami 3.20 lentelėje:

3.20 lentelė. Kombinuoto metodo nr.4 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,8
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0,083333333	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0,052083333	
Programos užimama vieta (KB)	1,59	3,94
Naudojamos vietos įvertis	1,477987421	
Programos vykdymo laikas (ns)	2412700	6221200
Vykdymo laiko įvertis	1,578521988	
Bendras resursų kainos įvertis	0,952866341	

Bendras kombinuoto metodo nr.4 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 47{,}44904541 + 0{,}6 \cdot 0 - 0{,}952866341 = 18{,}02675182$$

Metodas Nr.5 – Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas

Kombinuoto metodo nr.5 stiprumo įverčiai pateikiami 3.21 lentelėje:

3.21 lentelė. Kombinuoto metodo nr.5 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	53
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	285
Programos ilgio įvertis	2,75
Bendras stiprumo įvertis	**68,75**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.5 naudojamų resursų kainos įverčiai pateikiami 3.22 lentelėje:

3.22 lentelė. Kombinuoto metodo nr.5 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,33
Naudojamos vietos įvertis	1,72327044	
Programos vykdymo laikas (ns)	2412700	5718500
Vykdymo laiko įvertis	1,370166204	
Bendras resursų kainos įvertis	0,875065358	

Bendras kombinuoto metodo nr.5 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 68{,}75 + 0{,}6 \cdot 0 - 0{,}875065358 = 26{,}62493464$$

Metodas Nr.6 – Duomenų užtemdymas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas

Kombinuoto metodo nr.6 stiprumo įverčiai pateikiami 3.23 lentelėje:

3.23 lentelė. Kombinuoto metodo nr.6 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	41
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0,195121951
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	369
Programos ilgio įvertis	3,855263158
Bendras stiprumo įvertis	**82,56578947**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.6 naudojamų resursų kainos įverčiai pateikiami 3.24 lentelėje:

3.24 lentelė. Kombinuoto metodo nr.6 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,1
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,010869565	
Bendras atminties įvertis	-0,004076087	
Programos užimama vieta (KB)	1,59	3,74
Naudojamos vietos įvertis	1,352201258	
Programos vykdymo laikas (ns)	2412700	6109500
Vykdymo laiko įvertis	1,532225308	
Bendras resursų kainos įvertis	0,890701142	

Bendras kombinuoto metodo nr.6 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 82{,}56578947 + 0{,}6 \cdot 0 - 0{,}89070114 = 32{,}13561465$$

Metodas Nr.7 – Bereikšmio kodo įterpimas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas

Kombinuoto metodo nr.7 stiprumo įverčiai pateikiami 3.25 lentelėje:

3.25 lentelė. Kombinuoto metodo nr.7 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	65
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	8
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0.123076923
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	329
Programos ilgio įvertis	3,328947368
Bendras stiprumo įvertis	**76,75607287**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.7 naudojamų resursų kainos įverčiai pateikiami 3.26 lentelėje:

3.26 lentelė. Kombinuoto metodo nr.7 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,8
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0,083333333	
Naudojamos neheap atminties įvertis	0	

Bendras atminties įvertis	0,052083333	
Programos užimama vieta (KB)	1,59	4,94
Naudojamos vietos įvertis	2,106918239	
Programos vykdymo laikas (ns)	2412700	5846200
Vykdymo laiko įvertis	1,423094458	
Bendras resursų kainos įvertis	0,977263576	

Bendras kombinuoto metodo nr.7 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 76{,}75607287 + 0{,}6 \cdot 0 - 0{,}977263576 = 29{,}72516557$$

Metodas Nr.8 – Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas

Kombinuoto metodo nr.8 stiprumo įverčiai pateikiami 3.27 lentelėje:

3.27 lentelė. Kombinuoto metodo nr.8 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	16
Ciklų gylio sudėtingumo įvertis	3
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	74
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	319
Programos ilgio įvertis	3,197368421
Bendras stiprumo įvertis	**83,71710526**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.8 naudojamų resursų kainos įverčiai pateikiami 3.28 lentelėje:

3.28 lentelė. Kombinuoto metodo nr.8 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	5,22
Naudojamos vietos įvertis	2,283018868	
Programos vykdymo laikas (ns)	2412700	6113800
Vykdymo laiko įvertis	1,534007543	

Bendras resursų kainos įvertis	1,032756225

Bendras kombinuoto metodo nr.8 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 83{,}71710526 + 0{,}6 \cdot 0 - 1{,}032756225 = 32.45408588$$

Metodas Nr.9 – Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Duomenų užtemdymas

Kombinuoto metodo nr.9 stiprumo įverčiai pateikiami 23.9 lentelėje:

3.29 lentelė. Kombinuoto metodo nr.9 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	15
Ciklų gylio sudėtingumo įvertis	2,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	78
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	9
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,115384615
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	337
Programos ilgio įvertis	3,434210526
Bendras stiprumo įvertis	**84,27378543**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.9 naudojamų resursų kainos įverčiai pateikiami 3.30 lentelėje:

3.30 lentelė. Kombinuoto metodo nr.9 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	6,5
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	-0,097222222	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	-0,060763889	
Programos užimama vieta (KB)	1,59	5,39
Naudojamos vietos įvertis	2,389937107	
Programos vykdymo laikas (ns)	2412700	7827500
Vykdymo laiko įvertis	2,244290629	
Bendras resursų kainos įvertis	1,344115793	

Bendras kombinuoto metodo nr.9 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 84{,}27378543 + 0{,}6 \cdot 0 - 1{,}344115793 = 32{,}36539838$$

Metodas Nr.10 – Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas

Kombinuoto metodo nr.10 stiprumo įverčiai pateikiami 3.31 lentelėje:

3.31 lentelė. Kombinuoto metodo nr.10 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	11
Ciklų gylio sudėtingumo įvertis	1,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	53
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	7
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,132075472
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	268
Programos ilgio įvertis	2,526315789
Bendras stiprumo įvertis	**60,52941907**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.10 naudojamų resursų kainos įverčiai pateikiami 3.32 lentelėje:

3.32 lentelė. Kombinuoto metodo nr.10 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,3
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0,010869565	
Bendras atminties įvertis	0,004076087	
Programos užimama vieta (KB)	1,59	4,26
Naudojamos vietos įvertis	1,679245283	
Programos vykdymo laikas (ns)	2412700	8387700
Vykdymo laiko įvertis	2,476478634	
Bendras resursų kainos įvertis	1,367932612	

Bendras kombinuoto metodo nr.10 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 60{,}52941907 + 0{,}6 \cdot 0 - 1{,}367932612 = 22{,}84383501$$

Metodas Nr.11 – Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas

Kombinuoto metodo nr.11 stiprumo įverčiai pateikiami 3.33 lentelėje:

3.33 lentelė. Kombinuoto metodo nr.11 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	11
Ciklų gylio sudėtingumo įvertis	1,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	53
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	258
Programos ilgio įvertis	2,394736842
Bendras stiprumo įvertis	**58,05921053**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.11 naudojamų resursų kainos įverčiai pateikiami 3.34 lentelėje:

3.34 lentelė. Kombinuoto metodo nr.11 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	6,5
Vidutinė naudojama neheap atmintis (MB)	9,2	9,3
Naudojamos heap atminties įvertis	-0,097222222	
Naudojamos neheap atminties įvertis	0,010869565	
Bendras atminties įvertis	-0,056687802	
Programos užimama vieta (KB)	1,59	4,77
Naudojamos vietos įvertis	2	
Programos vykdymo laikas (ns)	2412700	5860400
Vykdymo laiko įvertis	1,428979981	
Bendras resursų kainos įvertis	0,920365871	

Bendras kombinuoto metodo nr.11 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 58{,}05921053 + 0{,}6 \cdot 0 - 1{,}428979981 = 22{,}30331834$$

Metodas Nr.12 – Atsitiktinis užtemdymas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas

Kombinuoto metodo nr.12 stiprumo įverčiai pateikiami 3.35 lentelėje:

3.35 lentelė. Kombinuoto metodo nr.12 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	11
Ciklų gylio sudėtingumo įvertis	1,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0

Kintamųjų skaičius užtemdytoje programoje	51
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	4
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,078431373
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	376
Programos ilgio įvertis	3,947368421
Bendras stiprumo įvertis	**77,95730134**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.12 naudojamų resursų kainos įverčiai pateikiami 3.36 lentelėje:

3.36 lentelė. Kombinuoto metodo nr.12 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,48
Naudojamos vietos įvertis	1,817610063	
Programos vykdymo laikas (ns)	2412700	6295001
Vykdymo laiko įvertis	1,60911054	
Bendras resursų kainos įvertis	0,996741252	

Bendras kombinuoto metodo nr.12 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 77{,}95730134 + 0{,}6 \cdot 0 - 0.996741252 = 30{,}18617928$$

Metodas Nr.13 – Atsitiktinis užtemdymas → Duomenų užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas

Kombinuoto metodo nr.13 stiprumo įverčiai pateikiami 3.37 lentelėje:

3.37 lentelė. Kombinuoto metodo nr.13 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	12
Ciklų gylio sudėtingumo įvertis	2
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	51
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0

Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	296
Programos ilgio įvertis	2,894736842
Bendras stiprumo įvertis	**67,43421053**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.13 naudojamų resursų kainos įverčiai pateikiami 3.38 lentelėje:

3.38 lentelė. Kombinuoto metodo nr.13 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	6,5
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	-0.097222222	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	-0.060763889	
Programos užimama vieta (KB)	1,59	4,6
Naudojamos vietos įvertis	1,893081761	
Programos vykdymo laikas (ns)	2412700	5654199
Vykdymo laiko įvertis	1,343515149	
Bendras resursų kainos įvertis	0,864238526	

Bendras kombinuoto metodo nr.13 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 67{,}43421053 + 0{,}6 \cdot 0 - 0{,}864238526 = 26{,}10944568$$

Metodas Nr.14 – Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Kintamųjų pavadinimų keitimas

Kombinuoto metodo nr.14 stiprumo įverčiai pateikiami 3.39 lentelėje:

3.39 lentelė. Kombinuoto metodo nr.14 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	11
Ciklų gylio sudėtingumo įvertis	1,75
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	48
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	243
Programos ilgio įvertis	2,197368421
Bendras stiprumo įvertis	**55,59210526**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.14 naudojamų resursų kainos įverčiai pateikiami 3.40 lentelėje:

3.40 lentelė. Kombinuoto metodo nr.14 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,6
Naudojamos vietos įvertis	1,893081761	
Programos vykdymo laikas (ns)	2412700	6595500
Vykdymo laiko įvertis	1,733659386	
Bendras resursų kainos įvertis	1,064108988	

Bendras kombinuoto metodo nr.14 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 55{,}59210526 + 0{,}6 \cdot 0 - 1{,}064108988 = 21{,}17273312$$

Metodas Nr.15 – Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas

Kombinuoto metodo nr.15 stiprumo įverčiai pateikiami 3.41 lentelėje:

3.41 lentelė. Kombinuoto metodo nr.15 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	12
Ciklų gylio sudėtingumo įvertis	2
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	69
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	253
Programos ilgio įvertis	2,328947368
Bendras stiprumo įvertis	**60,36184211**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.15 naudojamų resursų kainos įverčiai pateikiami 3.42 lentelėje:

3.42 lentelė. Kombinuoto metodo nr.15 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,82
Naudojamos vietos įvertis	2,031446541	
Programos vykdymo laikas (ns)	2412700	5765701
Vykdymo laiko įvertis	1,389729763	
Bendras resursų kainos įvertis	0,930095375	

Bendras kombinuoto metodo nr.15 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 60{,}36184211 + 0{,}6 \cdot 0 - 0{,}930095375 = 23{,}21464147$$

Metodas Nr.16 – Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas

Kombinuoto metodo nr.16 stiprumo įverčiai pateikiami 3.43 lentelėje:

3.43 lentelė. Kombinuoto metodo nr.16 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	12
Ciklų gylio sudėtingumo įvertis	2
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	47
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	6
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	0
Pasikartojančių kintamųjų įvertis	0,127659574
Papildomų kintamųjų įvertis	0
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	260
Programos ilgio įvertis	2,421052632
Bendras stiprumo įvertis	**62,31103024**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.16 naudojamų resursų kainos įverčiai pateikiami 3.44 lentelėje:

3.44 lentelė. Kombinuoto metodo nr.16 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	

Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,13
Naudojamos vietos įvertis	1,597484277	
Programos vykdymo laikas (ns)	2412700	6116500
Vykdymo laiko įvertis	1,535126622	
Bendras resursų kainos įvertis	0,930429621	

Bendras kombinuoto metodo nr.16 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 62{,}31103024 + 0{,}6 \cdot 0 - 0{,}930429621 = 23{,}99398247$$

Metodas Nr.17 – Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas

Kombinuoto metodo nr.17 stiprumo įverčiai pateikiami 3.45 lentelėje:

3.45 lentelė. Kombinuoto metodo nr.17 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	9
Ciklų gylio sudėtingumo įvertis	1,25
Neskaitomų eilučių skaičius	0
Neskaitomumo įvertis	0
Kintamųjų skaičius užtemdytoje programoje	40
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	8
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,2
Papildomų kintamųjų įvertis	0,05
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	287
Programos ilgio įvertis	2,776315789
Bendras stiprumo įvertis	**58,14144737**

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.17 naudojamų resursų kainos įverčiai pateikiami 3.46 lentelėje:

3.46 lentelė. Kombinuoto metodo nr.17 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	9,2
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	0	
Bendras atminties įvertis	0	
Programos užimama vieta (KB)	1,59	4,25
Naudojamos vietos įvertis	1,672955975	
Programos vykdymo laikas (ns)	2412700	5090600
Vykdymo laiko įvertis	1,109918349	

Bendras resursų kainos įvertis	0,743884914

Bendras kombinuoto metodo nr.17 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 58{,}14144737 + 0{,}6 \cdot 0 - 0{,}743884914 = 22{,}51269403$$

Metodas Nr.18 – Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Kintamųjų pavadinimų keitimas → Duomenų užtemdymas

Kombinuoto metodo nr.18 stiprumo įverčiai pateikiami 3.47 lentelėje:

3.47 lentelė. Kombinuoto metodo nr.18 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Kintamųjų skaičius užtemdytoje programoje	65
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	0
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0
Papildomų kintamųjų įvertis	0,030769231
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	256
Programos ilgio įvertis	2,368421053
Bendras stiprumo įvertis	64,17257085

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.18 naudojamų resursų kainos įverčiai pateikiami 3.48 lentelėje:

3.48 lentelė. Kombinuoto metodo nr.18 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,9
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,032608696	
Bendras atminties įvertis	-0,012228261	
Programos užimama vieta (KB)	1,59	5,05
Naudojamos vietos įvertis	2,176100629	
Programos vykdymo laikas (ns)	2412700	5883499
Vykdymo laiko įvertis	1,438553902	
Bendras resursų kainos įvertis	0,968873046	

Bendras kombinuoto metodo nr.19 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 64{,}17257085 + 0{,}6 \cdot 0 - 0{,}968873046 = 24{,}70015529$$

Metodas Nr.19 – Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas

Kombinuoto metodo nr.19 stiprumo įverčiai pateikiami 3.49 lentelėje:

3.49 lentelė. Kombinuoto metodo nr.19 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	15
Ciklų gylio sudėtingumo įvertis	2,75
Kintamųjų skaičius užtemdytoje programoje	87
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	15
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,172413793
Papildomų kintamųjų įvertis	0,022988506
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	336
Programos ilgio įvertis	3,421052632
Bendras stiprumo įvertis	84,60942226

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.19 naudojamų resursų kainos įverčiai pateikiami 3.50 lentelėje:

3.50 lentelė. Kombinuoto metodo nr.19 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,9
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,032608696	
Bendras atminties įvertis	-0,012228261	
Programos užimama vieta (KB)	1,59	5,3
Naudojamos vietos įvertis	2,33333	
Programos vykdymo laikas (ns)	2412700	6424899
Vykdymo laiko įvertis	1,662949807	
Bendras resursų kainos įvertis	1,093436109	

Bendras kombinuoto metodo nr.19 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 84{,}60942226 + 0{,}6 \cdot 0 - 1{,}093436109 = 32{,}7503328$$

Metodas Nr.20 – Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas

Kombinuoto metodo nr.20 stiprumo įverčiai pateikiami 3.51 lentelėje:

3.51 lentelė. Kombinuoto metodo nr.20 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4

Ciklų gylio įvertis užtemdytoje programoje.	15
Ciklų gylio sudėtingumo įvertis	2,75
Kintamųjų skaičius užtemdytoje programoje	70
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	11
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,157142857
Papildomų kintamųjų įvertis	0,028571429
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	346
Programos ilgio įvertis	3,552631579
Bendras stiprumo įvertis	86,19360902

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.20 naudojamų resursų kainos įverčiai pateikiami 3.52 lentelėje:

3.52 lentelė. Kombinuoto metodo nr.20 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,9
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,032608696	
Bendras atminties įvertis	-0,012228261	
Programos užimama vieta (KB)	1,59	4,87
Naudojamos vietos įvertis	2.062893082	
Programos vykdymo laikas (ns)	2412700	5716300
Vykdymo laiko įvertis	1,369254362	
Bendras resursų kainos įvertis	0,920707121	

Bendras kombinuoto metodo nr.20 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 86{,}19360902 + 0{,}6 \cdot 0 - 0{,}92070712 = 33{,}55673649$$

Metodas Nr.21 – Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Duomenų užtemdymas → Bereikšmio kodo įterpimas

Kombinuoto metodo nr.21 stiprumo įverčiai pateikiami 3.53 lentelėje:

3.53 lentelė. Kombinuoto metodo nr.21 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Kintamųjų skaičius užtemdytoje programoje	46
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	8
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,173913043
Papildomų kintamųjų įvertis	0,043478261
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0

Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	273
Programos ilgio įvertis	2,592105263
Bendras stiprumo įvertis	68,13501144

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.21 naudojamų resursų kainos įverčiai pateikiami 3.54 lentelėje:

3.54 lentelė. Kombinuoto metodo nr.21 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,9
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,032608696	
Bendras atminties įvertis	-0,012228261	
Programos užimama vieta (KB)	1,59	4,22
Naudojamos vietos įvertis	1,65408805	
Programos vykdymo laikas (ns)	2412700	6301900
Vykdymo laiko įvertis	1,611969992	
Bendras resursų kainos įvertis	0,9686084	

Bendras kombinuoto metodo nr.21 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 68{,}13501144 + 0{,}6 \cdot 0 - 0{,}9686084 = 26{,}28539618$$

Metodas Nr.22 – Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas

Kombinuoto metodo nr.22 stiprumo įverčiai pateikiami 3.55 lentelėje:

3.55 lentelė. Kombinuoto metodo nr.22 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	10
Ciklų gylio sudėtingumo įvertis	1,5
Kintamųjų skaičius užtemdytoje programoje	55
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	8
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,145454545
Papildomų kintamųjų įvertis	0,036363636
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	246
Programos ilgio įvertis	2,236842105
Bendras stiprumo įvertis	54,09688995

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.22 naudojamų resursų kainos įverčiai pateikiami 3.56 lentelėje:

3.56 lentelė. Kombinuoto metodo nr.22 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,9
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,032608696	
Bendras atminties įvertis	-0,012228261	
Programos užimama vieta (KB)	1,59	4,68
Naudojamos vietos įvertis	1,943396226	
Programos vykdymo laikas (ns)	2412700	5756500
Vykdymo laiko įvertis	1,385916193	
Bendras resursų kainos įvertis	0,910280417	

Bendras kombinuoto metodo nr.22 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 54{,}09688995 + 0{,}6 \cdot 0 - 0{,}91028041 = 20{,}72847556$$

Metodas Nr.23 – Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas

Kombinuoto metodo nr.23 stiprumo įverčiai pateikiami 3.57 lentelėje:

3.57 lentelė. Kombinuoto metodo nr.23 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	13
Ciklų gylio sudėtingumo įvertis	2,25
Kintamųjų skaičius užtemdytoje programoje	44
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	9
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,204545455
Papildomų kintamųjų įvertis	0,045454545
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	263
Programos ilgio įvertis	2,460526316
Bendras stiprumo įvertis	66,69407895

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.23 naudojamų resursų kainos įverčiai pateikiami 3.58 lentelėje:

3.58 lentelė. Kombinuoto metodo nr.23 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,8
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0.043478261	

Bendras atminties įvertis	-0.016304348	
Programos užimama vieta (KB)	1,59	3,79
Naudojamos vietos įvertis	1,383647799	
Programos vykdymo laikas (ns)	2412700	6050000
Vykdymo laiko įvertis	1,50756414	
Bendras resursų kainos įvertis	0,879429294	

Bendras kombinuoto metodo nr.23 efektyvumas apskaičiuojamas pagal (3.1) formulę:

$$S_{efek} = 0{,}4 \cdot 66{,}69407895 + 0{,}6 \cdot 0 - 0{,}879429294 = 25{,}79820229$$

Metodas Nr.24 – Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas

Kombinuoto metodo nr.24 stiprumo įverčiai pateikiami 3.59 lentelėje:

3.59 lentelė. Kombinuoto metodo nr.24 stiprumo įverčiai

Ciklų gylio įvertis pradinėje programoje	4
Ciklų gylio įvertis užtemdytoje programoje.	9
Ciklų gylio sudėtingumo įvertis	1,25
Kintamųjų skaičius užtemdytoje programoje	41
Pasikartojančių kintamųjų skaičius užtemdytoje programoje	9
Kintamųjų kiekis pradinėje programoje	11
Papildomų kintamųjų skaičius užtemdytoje programoje	2
Pasikartojančių kintamųjų įvertis	0,219512195
Papildomų kintamųjų įvertis	0,048780488
Prasmingų kintamųjų įvertis	1
Šifruotų kintamųjų įvertis	0
Kodo eilučių skaičius pradinėje programoje	76
Kodo eilučių skaičius užtemdytoje programoje	268
Programos ilgio įvertis	2,526315789
Bendras stiprumo įvertis	55,13077664

Sukompiliuotą programą bandant analizuoti įrankiu „JD Project Java decompiler“ pirminis kodas grąžintas be klaidų, todėl atsparumo įvertis yra lygus nuliui.

$$S_{atsp} = 0$$

Kombinuoto metodo nr.24 naudojamų resursų kainos įverčiai pateikiami 3.60 lentelėje:

3.60 lentelė. Kombinuoto metodo nr.24 naudojamų resursų įverčiai

	Pradinis kodas	Užtemdytas
Vidutinė naudojama heap atmintis (MB)	7,2	7,2
Vidutinė naudojama neheap atmintis (MB)	9,2	8,9
Naudojamos heap atminties įvertis	0	
Naudojamos neheap atminties įvertis	-0,032608696	
Bendras atminties įvertis	-0,012228261	
Programos užimama vieta (KB)	1,59	3,82
Naudojamos vietos įvertis	1,402515723	
Programos vykdymo laikas (ns)	2412700	5880600
Vykdymo laiko įvertis	1,437352344	
Bendras resursų kainos įvertis	0,852294609	

Bendras kombinuoto metodo nr.24 efektyvumas apskaičiuojamas pagal (3.1) formulę:

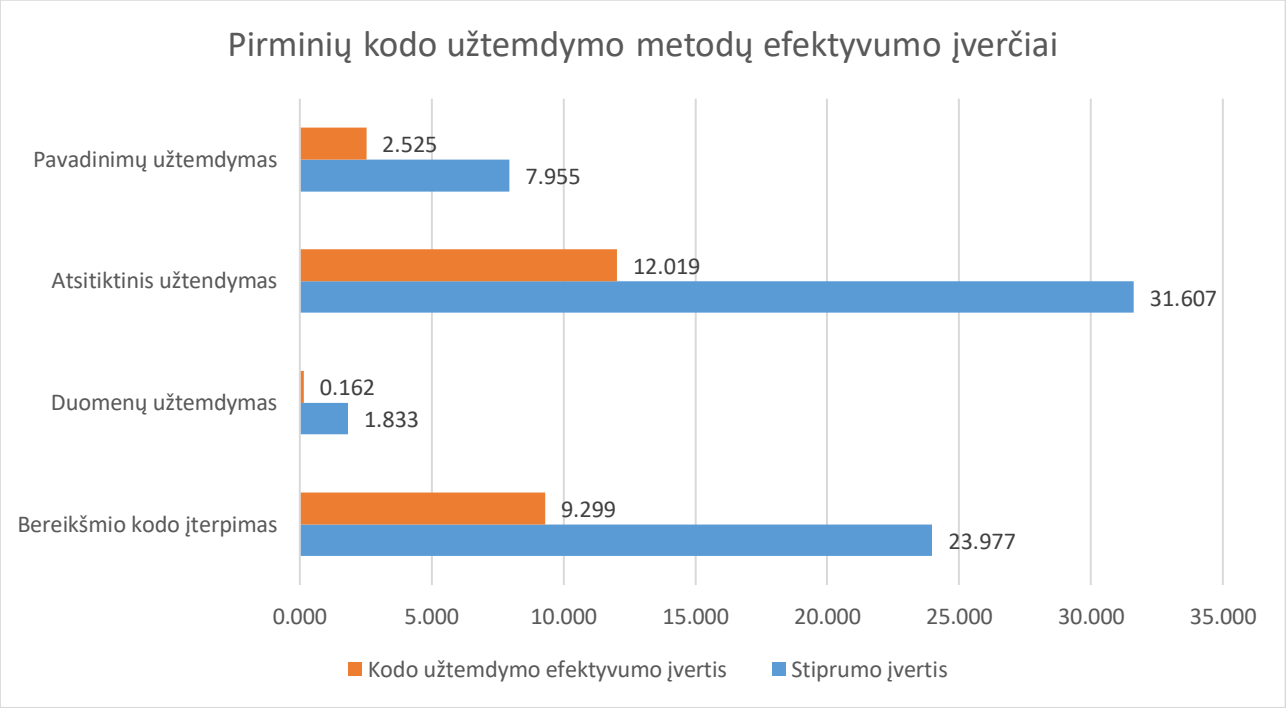
$$S_{efek} = 0{,}4 \cdot 55{,}13077664 + 0{,}6 \cdot 0 - 0{,}852294609 = 21{,}20001605$$

3.2. Tyrimo rezultatų įvertinimas

Šiame skyriuje yra pateikiami ir apibendrinami atlikto tyrimo rezultatai. Yra palyginami tirtų metodų stiprumo ir kainos įverčiai, bei bendras kodo užtemdo metodų efektyvumas. Toliau 3.61 lentelėje, bei 3.1 paveiksle pateikiami nekombinuotų užtemdymo metodų surinkti efektyvumo įverčiai.

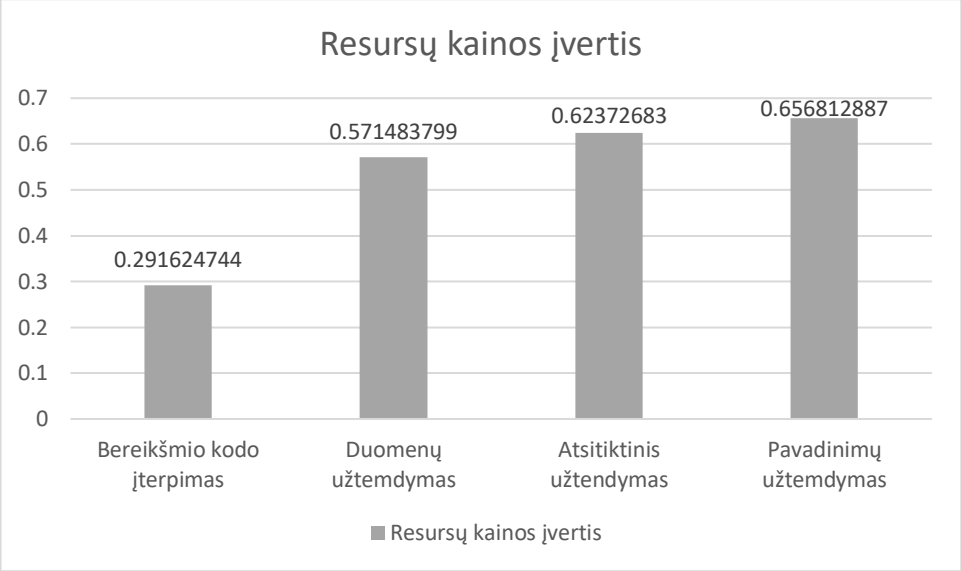
3. 61 lentelė. Pradinių kodo užtemdymo metodų efektyvumo įverčiai

Kodo užtemdymo metodas	**Stiprumo įvertis**	**Resursų kainos įvertis**	**Kodo užtemdymo efektyvumo įvertis**
Bereikšmio kodo įterpimas	23,97660819	0,291624744	9,299018531
Duomenų užtemdymas	1,832706767	0,571483799	0,161598908
Atsitiktinis užtemdymas	31,6073049	0,62372683	12,01919513
Pavadinimų užtemdymas	7,954545455	0,656812887	2,525005295



3.1 pav. Pirminių kodo užtemdymo metodų efektyvumo ir stiprumo įverčiai

Kaip pavaizduota 3.1 paveiksle, atsitiktinis užtemdymas turi didžiausią stiprumo įvertį iš nekombinuotų užtemdymo metodų. Tai galima paaiškinti tuo, kad šis metodas įterpia daugiau kodo eilučių, ciklų bei kintamųjų į pradinį programos kodą, nei likusieji metodai. 3.2 paveiksle pateikiami pirminių nekombinuotų užtemdymo metodų resursų kainos įverčiai.

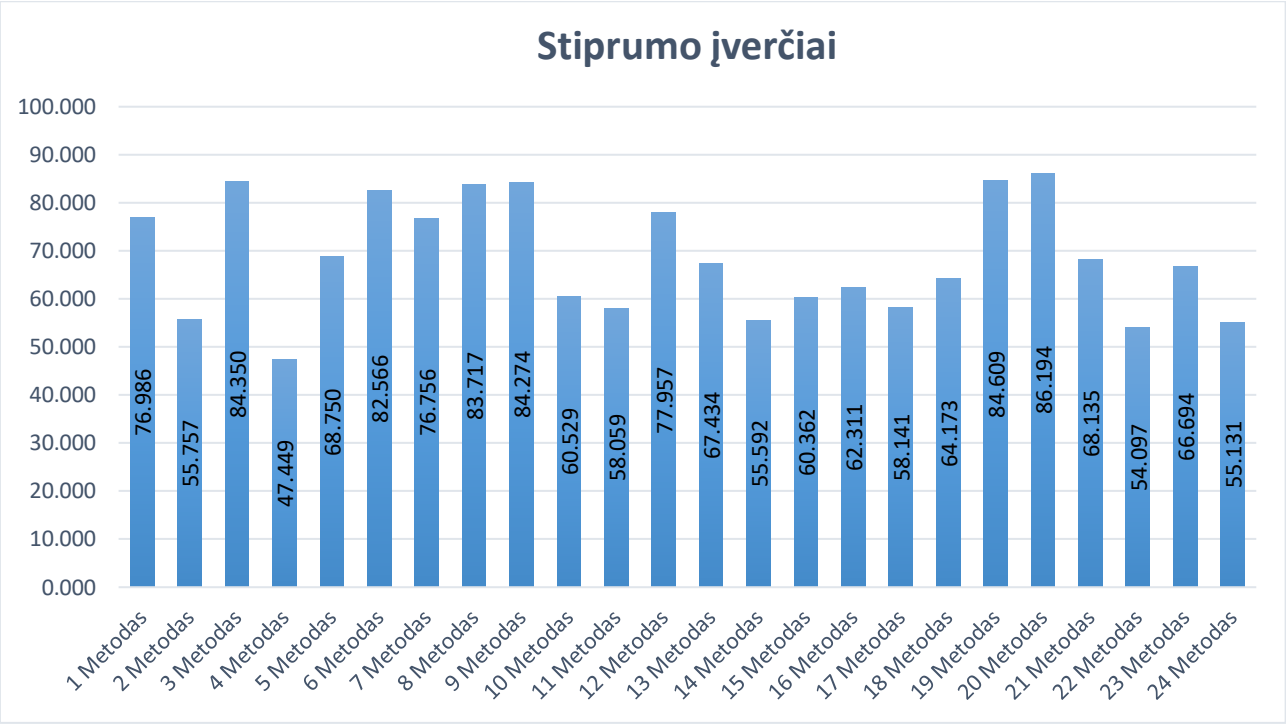


3.2 pav. Pirminių kodo užtemdymo metodų resursų kainos įverčiai

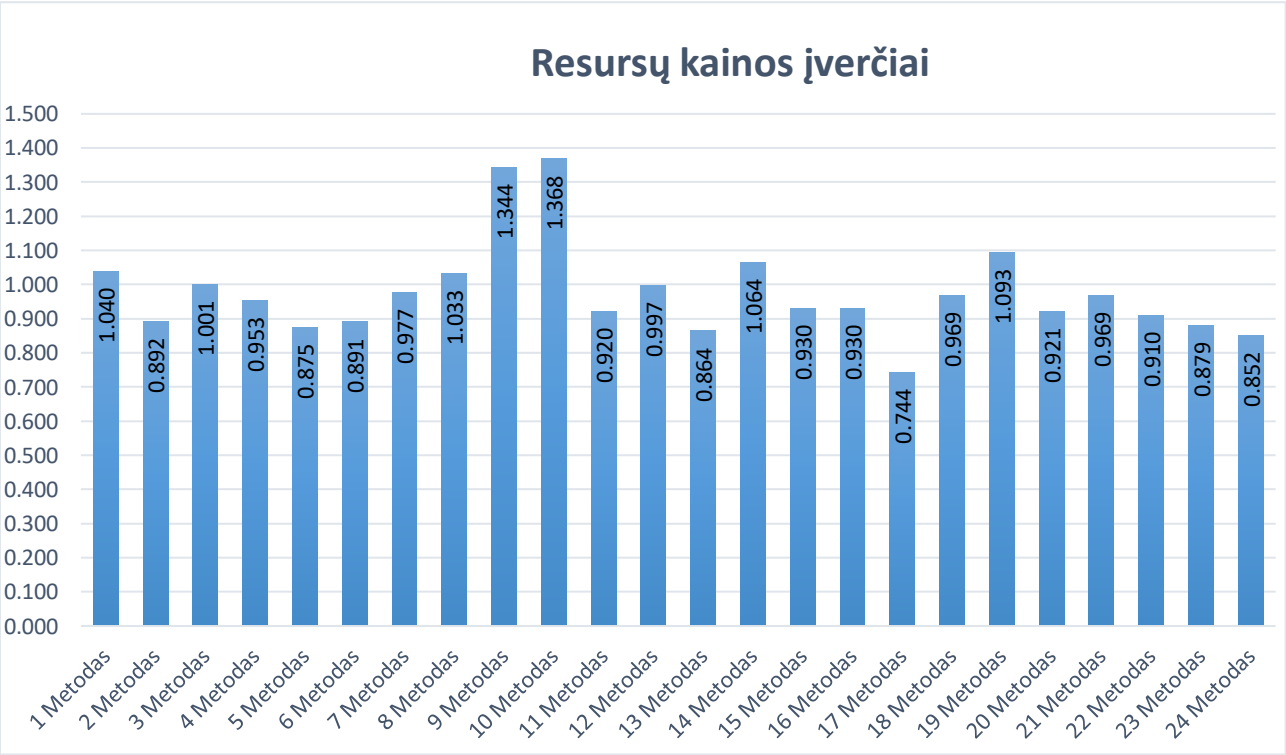
Pateiktuose paveiksluose yra palyginami nekombinuotų užtemdymo metodų naudojamų resursų, bei bendro užtemdymo efektyvumo įverčiai. Iš grafikų matosi, kad didžiausias kodo užtemdymo efektyvumo įvertis gautas naudojant atsitiktinio užtemdymo metodą, tuo tarpu mažiausias įvertis buvo duomenų užtemdymo. Taip yra dėl to, kad taikant duomenų užtemdymą, mažai pakeičiamas programos ilgis, nėra užtemdomi kintamųjų pavadinimai, o į žmogui sunkiau suprantama kintamųjų pateikimą kodo užtemdymo efektyvumo metrikos neatsižvelgia. Toliau 4.2 lentelėje, bei 4.3, 4.4 ir 4.5 paveiksluose pateikiami kombinuotų kodo užtemdymo metodų tyrimo rezultatai.

3.62 lentelė. Kombinuotų užtemdymo metodų efektyvumo įverčiai

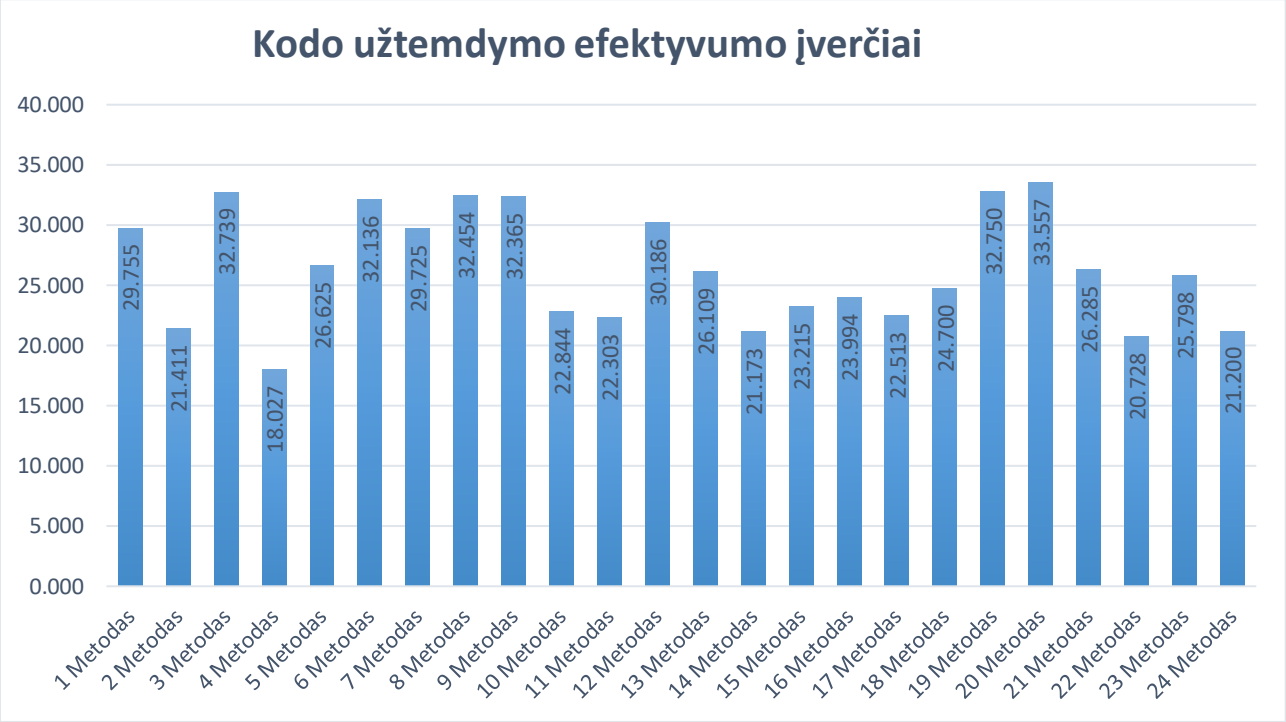
Kombinacija	Stiprumo įvertis	Resursų kainos įvertis	Kodo užtemdymo efektyvumo įvertis
1 Metodas	76.986	1.040	29.755
2 Metodas	55.757	0.892	21.411
3 Metodas	84.350	1.001	32.739
4 Metodas	47.449	0.953	18.027
5 Metodas	68.750	0.875	26.625
6 Metodas	82.566	0.891	32.136
7 Metodas	76.756	0.977	29.725
8 Metodas	83.717	1.033	32.454
9 Metodas	84.274	1.344	32.365
10 Metodas	60.529	1.368	22.844
11 Metodas	58.059	0.920	22.303
12 Metodas	77.957	0.997	30.186
13 Metodas	67.434	0.864	26.109
14 Metodas	55.592	1.064	21.173
15 Metodas	60.362	0.930	23.215
16 Metodas	62.311	0.930	23.994
17 Metodas	58.141	0.744	22.513
18 Metodas	64.173	0.969	24.700
19 Metodas	84.609	1.093	32.750
20 Metodas	86.194	0.921	33.557
21 Metodas	68.135	0.969	26.285
22 Metodas	54.097	0.910	20.728
23 Metodas	66.694	0.879	25.798
24 Metodas	55.131	0.852	21.200



3.3 pav. Kombinuotų užtemdymo metodų stiprumo įverčiai



3.4 pav. Kombinuotų užtemdymo metodų resursų kainos įverčiai



3.5 pav. Kombinuotų užtemdymo metodų efektyvumo įverčiai

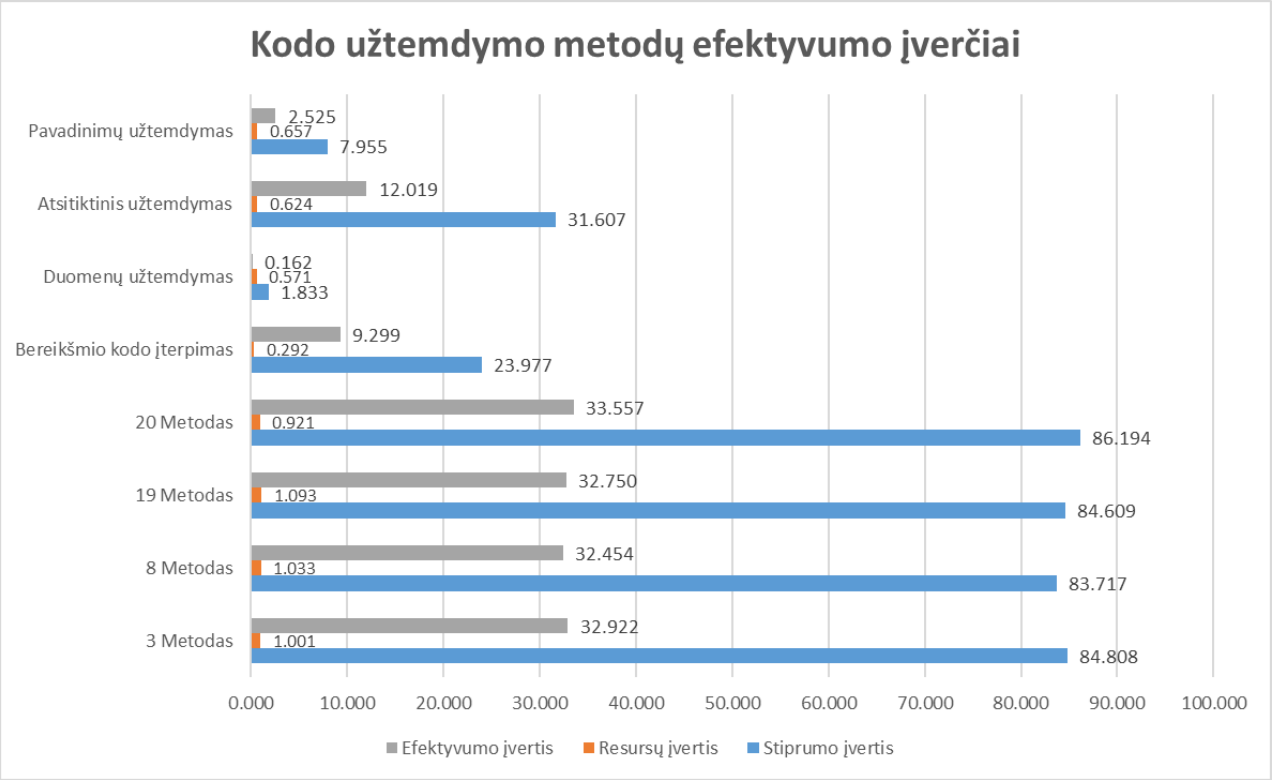
Iš tyrimo rezultatų matoma, kad didžiausias kodo užtemdymo efektyvumo įvertis pasiektas naudojant metodo nr.3 kodo užtemdymo taikymo tvarką. Taip pat artimi rezultatai gaunami ir metodo nr.6, metodo nr.8, bei metodo nr.9. Tuo tarpu mažiausi kodo užtemdymo efektyvumo įverčiai pasiekti taikant metodų nr.4, nr.14 ir nr.2 kodo užtemdymo metodų kombinacijas. Toliau 3.63 ir 3.64 lentelėse pateikiamos didžiausią bei mažiausią efektyvumo įvertį surinkusios metodų kombinacijos, 3.6 paveiksle pateikiamas keturių didžiausią efektyvumo įvertį surinkusių metodų palyginimas su nekombinuotais užtemdymo metodais.

3.63 lentelė. Didžiausią kodo užtemdymo efektyvumo įvertį surinkusių metodų taikymo kombinacijos

Užtemdymo metodas	**Užtemdymo metodų taikymo tvarka**
Metodas nr.20	Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas
Metodas nr.19	Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas
Metodas nr.3	Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas

3.64 lentelė. Mažiausią kodo užtemdymo efektyvumo įvertį surinkusių metodų taikymo kombinacijos

Užtemdymo metodas	**Užtemdymo metodų taikymo tvarka**
Metodas nr.4	Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas
Metodas nr.24	Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas
Metodas nr.22	Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas



3.6 pav. Kombinuotų ir nekombinuotų metodų palyginimas

3.3. Tyrimo rezultatų išvados

Atlikus eksperimentinį tyrimą su etaloniniu failu, taikant kodo užtemdymo transformacijas nustatyta:

a) Taikant kodo užtemdymo metodus atskirai, didžiausias kodo užtemdymo efektyvumas pasiektas naudojant atsitiktinį užtemdymą, mažiausias – duomenų užtemdymą.

b) Didžiausią kodo užtemdymo efektyvumo įvertį surinkę metodai kurių transformacijų užtemdymo tvarka tokia:

- Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas
- Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas
- Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas

c) Mažiausią kodo užtemdymo efektyvumo įvertį surinkę metodai kurių transformacijų užtemdymo tvarka tokia:

- Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas
- Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas
- Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas

Išvados

1. Šiuolaikinės programos susiduria su tokiomis grėsmėmis kaip apgrąžos inžinerija, kodo klastojimas ir piratavimas. Kodo užtemdymui naudojamos įvairios kodo transformacijos, kurios padaro kodą sunkiau suprantamą, tačiau kartu daro įtaką ir sunaudojamų resursų kiekiui. Remiantis atliktos analizės rezultatais nutarta ištirti kaip skirtinga kodo užtemdymo metodų taikymo tvarka, veikia užtemdymo efektyvumą.
2. Pasiūlytas kombinuotas kodo užtemdymo metodas, sukurtas programinės įrangos prototipo modelis tyrimams atlikti. Remiantis etaloniniais programų kodais kombinuoto užtemdymo metodo tyrimui pasiūlytas etaloninės programos pirminis kodas.
3. Atlikus eksperimentinį tyrimą su etaloniniu failu, taikant kodo užtemdymo transformacijas nustatyta:
    a) Taikant kodo užtemdymo metodus atskirai, didžiausias kodo užtemdymo efektyvumas pasiektas naudojant atsitiktinį užtemdymą, mažiausias – duomenų užtemdymą.
    b) Didžiausią kodo užtemdymo efektyvumo įvertį surinkę metodai kurių transformacijų užtemdymo tvarka tokia:
        - Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Duomenų užtemdymas → Atsitiktinis užtemdymas
        - Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas → Duomenų užtemdymas
        - Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas
    c) Mažiausią kodo užtemdymo efektyvumo įvertį surinkę metodai kurių transformacijų užtemdymo tvarka tokia:
        - Duomenų užtemdymas → Kintamųjų pavadinimų keitimas → Bereikšmio kodo įterpimas → Atsitiktinis užtemdymas
        - Kintamųjų pavadinimų keitimas → Duomenų užtemdymas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas
        - Kintamųjų pavadinimų keitimas → Atsitiktinis užtemdymas → Bereikšmio kodo įterpimas → Duomenų užtemdymas

Literatūros sąrašas

1. S. A. Sebastian, S. Malgaonkar, P. Shah, M. Kapoor, T. Parekhji, „A Study & Review on Code Obfuscation," 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare
2. S. Schrittwieser, S. Katzenbeisse, J. Kinder, G. Merzdovnik ir E. Weippl, „Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?," ACM Computing Surveys, t. 49, Nr. 1, Str. 4, balandis 2016.
3. P. Falcarin, M. Collberg, M. Atallah, M. Jakubowski „Software Protection", IEEE Software t. 28, Nr. 2, Kovas-Balandis 2011
4. S. Ghosh, J. D. Hiser, J. W. Davidson „A Secure and Robust Approach to Software Tamper Resistance", International Workshop on Information Hiding IH 2010: Information Hiding p. 33-47
5. Paolo Falcarin, Stefano Di Carlo, Alessandro Cabutto, Nicola Garazzino, Davide Barberis „Exploiting Code Mobility for Dynamic Binary Obfuscation", 2011 IEEE.
6. A. Viticchi, L. Regano, M. Torchiano, C. Basile, M. Ceccato, P. Tonella ir R. Tiella, „Assessment of Source Code Obfuscation Techniques," 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation.
7. N. P. Varnovskiya, V. A. Zakharovb, N. N. Kuzyurinc ir A. V. Shokurov, „The Current State of Art in Program Obfuscations: Definitions of Obfuscation Security," Programming and Computer Software, 2015, t. 41, Nr. 6, p. 361–372.
8. „Methods for Obfuscating Java Program"s Florin BUZATU IT&C Security Master Department of Economic Informatics and Cybernetics Bucharest University of Economic Studies, Kovas 2012
9. Babak Bashari Rad**,** Maslin Masrom, Suhaimi Ibrahim „Camouflage in Malware: from Encryption to Metamorphism", IJCSNS International Journal of Computer Science and Network Security, VOL.12 No.8, Rugpjūtis 2012
10. C. S. Collberg, C. D. Thomborson. „Watermarking, tamper-proofing, and obfuscation - tools for software protection." T. 28, p. 735–746, 2002.
11. W. Xu, F. Zhang ir S. Zhu, „The Power of Obfuscation Techniques in Malicious JavaScript Code: A Measurement Study," 2012 7th International Conference on Malicious and Unwanted Software
12. S. Qin, Z. Wang, Y. Wang, K. Xu, „A Method of JavaScript path obfuscation based on Collatz conjecture," 2015, 12th Web Information System and Application Conference.
13. V. Balachandran, S. Emmanuel, N.W. Keong, „Obfuscation by Code Fragmentation to Evade Reverse Engineering," 2014 IEEE International Conference on Systems, Man, and Cybernetics Spalis 5-8, 2014, San Diego, CA, JAV
14. A. K. Dalai, S. S. Das ir S. K. Jena „A Code Obfuscation Technique to Prevent Reverse Engineering," IEEE WiSPNET 2017.
15. Matthew et al. Karnick. „A qualitative analysis of java obfuscation.", 10th IASTED International Conference SOFTWARE ENGINEERING AND APPLICATIONS, p. 166–171, Dallas, TX, JAV, 2006.
16. „A Systematic Study on Static Control Flow Obfuscation Techniques in Java" Renuka Kumar, Anjana Mariam Kurian Amrita Center for Cybersecurity Systems & Networks, Amritapuri Campus, Amrita Vishwa Vidyapeetham, Rugsėjis, 2018

17. A. Gorji, M. Abadi, „Detecting Obfuscated JavaScript Malware Using Sequences of Internal Function Calls,“ ACM SE '14, Kovas 28–29, 2014, Kennesaw, GA, JAVJournal of Mobile, Embedded and Distributed Systems, vol. IV, no. 1, 2012 ISSN 2067 – 4074
18. Hongying Lai „A comparative survey of Java obfuscators available on the Internet”, Vasaris 22, 2001
19. Z. Yujia, P. Jianmin „A New Compile-time Obfuscation Scheme for Software Protection“, ‘16 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Spalis, 2016.

Informacijos šaltinių sąrašas

20. „JDProject Java decompiler,“ 03 sausio 2018, [Tinkle]. Available: http://java-decompiler.github.io/ [Kreiptasi 09 sausio 2018]
21. „AndroChef Decompiler“, [Tinkle]. Available: http://www.androiddecompiler.com/ [Kreiptasi 09 sausio 2018]
22. „DJ Java Decompiler“, [Tinkle]. Available: http://www.neshkov.com/ [Kreiptasi 09 sausio 2018]
23. „Procyon Java Decompiler“, https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler / [Kreiptasi 10 sausio 2018]
24. „The Java Language Environment“, [Tinkle]. Available: https://www.oracle.com/technetwork/java/intro-141325.html
25. „Eclipse IDE“, [Tinkle]. Available: https://www.eclipse.org/eclipseide/ [Kreiptasi 10 sausio 2018]
26. „Obfuscation benchmarks“, [Tinkle]. Available: https://github.com/tum-i22/obfuscation-benchmarks [Kreiptasi 15 kovo 2018]
27. „Spoon - Source Code Analysis and Transformation for Java“, [Tinkle]. Available: http://spoon.gforge.inria.fr/ [Kreiptasi 18 vasario 2018]