Kodo gavybos pagrindu išvestų požymių tyrimas programinės įrangos pažeidžiamumų aptikimui

Vestina Bertašiūtė, Asta Slotkienė

Vilniaus Gedimino technikos universitetas,
Saulėtekio al. 11, 10223 Vilnius
Vestina.bertasiute@gmail.com

Santrauka. Šiame straipsnyje nagrinėjama, kaip iš skirtingų kodo reprezentacijų - abstrakčių sintaksės medžių (AST), valdymo srautų grafų (CFG) ir programos priklausomybių grafų (PDG) - išvesti požymiai lemia mašininio mokymosi modelių gebėjimą aptikti programinės įrangos saugumo pažeidžiamumus. Tyrimo tikslas - sistemiškai įvertinti šių reprezentacijų informatyvumą skirtingoms „Common Weakness Enumeration“ (CWE) pažeidžiamumų klasėms. Tyrimo metu nustatyta, kad pritaikius kodo reprezentaciją konkrečiam pažeidžiamumo mechanizmui, ne tik reikšmingai padidėja klasifikavimo tikslumas, bet ir optimizuojami skaičiavimo resursai atsisakant neinformatyvių požymių.

Raktiniai žodžiai: statinė analizė, CWE, pažeidžiamumų aptikimas.

1 Įvadas

Programinės įrangos saugumo užtikrinimas išlieka vienu didžiausių iššūkių informacinių technologijų srityje, tačiau tradicinių statinės analizės įrankių efektyvumas praktikoje dažnai yra nepakankamas [1]. Literatūros apžvalgos rodo, kad 35-91 % standartinių įrankių generuojamų pranešimų yra klaidingi arba neinformatyvūs, o didelis klaidingai teigiamų rezultatų skaičius reikšmingai didina programinės įrangos audito sąnaudas [2]. Pagrindinė šio ribotumo priežastis - tradicinių įrankių orientacija į paviršinę kodo sintaksę, neįvertinant sudėtingų semantinių ryšių tarp programos komponentų [3].

Siekiant spręsti šią problemą, vis dažniau taikomas jungtinio kodo savybių grafo (angl. Code Property Graph, CPG) modelis [4]. Ši reprezentacija vieningoje struktūroje integruoja abstrakčius sintaksės medžius, valdymo srauto ir duomenų priklausomybių grafus, taip leisdama gauti išsamius kodo požymius. Nustatyta, kad pasitelkus CPG modelį, pažeidžiamumų aptikimo tikslumas gali būti padidintas iki 33 %, lyginant su metodais, naudojančiais tik pavienes kodo reprezentacijas [5].

DOI: https://doi.org/10.15388/LMITT.2026.3

Šiame straipsnyje keliamas klausimas, ar universalus kodo požymių rinkinys yra vienodai informatyvus visoms pažeidžiamumų klasėms. Tyrimo metu, naudojant specializuotą analizės įrankį „Joern“, siekiama gauti specifinius grafų savybių požymius ir eksperimentiškai įvertinti jų diagnostinę vertę 16-ai skirtingų „Common Weakness Enumeration“ (CWE) kategorijų. Taip nustatoma, kaip kodo reprezentacijos informatyvumas kinta priklausomai nuo pažeidžiamumo prigimties ir mechanizmo.

2 Tyrimo metodika ir duomenų paruošimas

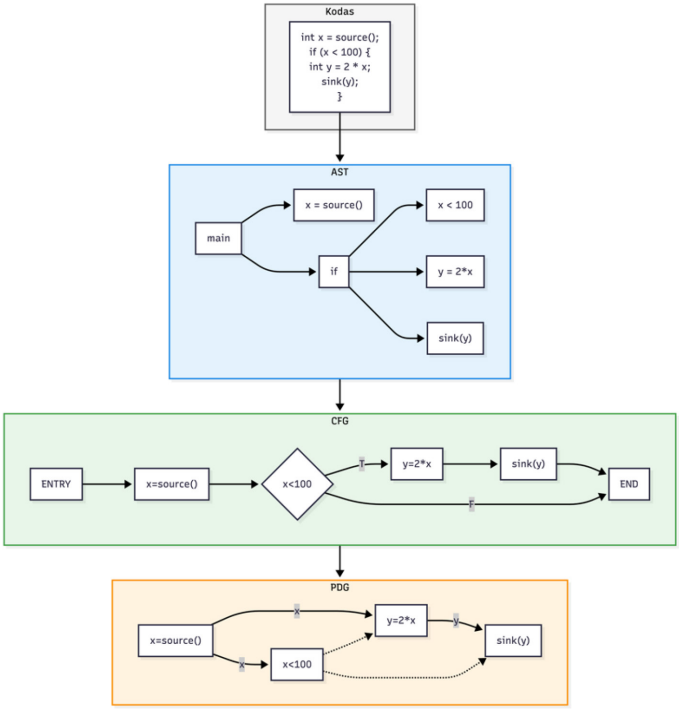
Eksperimentiniam vertinimui pasirinktas JAV Nacionalinės saugumo agentūros (angl. National Security Agency, NSA) sukurtas pažeidžiamumų testavimo rinkinys „Juliet Test Suite v1.3“, skirtas C/C++ programavimo kalbai [6]. Šis rinkinys pasirinktas dėl jo metodologinio pagrįstumo: jame pateikiami sužymėti kodo pavyzdžiai, kuriuose aiškiai atskiriamos pažeidžiamos ir saugios, pagal gerosios praktikos principus sutvarkytos funkcijų versijos.

Siekiant apimti kuo platesnį pažeidžiamumų spektrą, tyrimui tikslingai atrinkta 16 CWE klasių, kurios pagal veikimo principus suskirstytos į keturias pagrindines kategorijas: atminties rėžių klaidas (CWE-121, 122, 124, 126, 127), atminties valdymo klaidas (CWE-415, 416, 476), įvesties klaidas (CWE-78, 90, 134) bei logines klaidas (CWE-190, 680, 367, 426, 427). Pasirinktos kategorijos yra aktualios šiuolaikinių grafinių kodo reprezentacijų tyrimų kontekste: jų analizė dominuoja naujausiuose grafų neuroninių tinklų [7] bei kodo modelių informatyvumo palyginimo darbuose [8], [9].

Eksperimentui atlikti pradinė duomenų imtis buvo nuosekliai filtruojama naudojant „Joern“ analizės platformą. Pagrindinis šio etapo tikslas - užtikrinti, kad mašininio mokymosi modelis būtų apmokomas naudojant tik semantiškai turtingą ir tyrimui aktualų programinį kodą. Iš pradinio rinkinio pašalinti visi pagalbiniai programos elementai bei techniniai artefaktai, neturintys tiesioginio ryšio su pažeidžiamumų mechanizmais. Atrinktų funkcijų ženklinimas pagrįstas duomenų rinkinio pavadinimų konvencija: pažeidžiamoms funkcijoms priskirta reikšmė 1, saugioms - 0. Galutinę tyrimo imtį sudarė 25 951 funkcija: 8 246 pažeidžiamos ir 17 705 saugios. Toks sisteminis paruošimas leidžia modeliui efektyviau identifikuoti esminius skirtumus tarp saugaus ir pažeidžiamo kodo variantų, eliminuojant informacinį triukšmą.

Kaip pavaizduota 1 pav., funkcijos transformavimo procesas prasideda nuo C/C++ kodo, kuris pirmiausia konvertuojamas į abstraktų sintaksės medį (AST). Šiame etape kodas išskaidomas į hierarchinę struktūrą, kurioje

kiekvienas mazgas atitinka konkrečią programos konstrukciją. Toliau generuojamas valdymo srauto grafas (CFG), pavaizduotas vidurinėje schemos dalyje. Šiame grafike programa reprezentuojama kaip galimų vykdymo kelių visuma, kur šakos (pvz., if sąlygos) ir perėjimai tarp blokų atspindi skirtingus vykdymo scenarijus. Taip pat sudaromas duomenų priklausomybių grafas (PDG), susiejantis duomenų srautus su CFG elementais. Šiame etape nustatomi duomenų perdavimo tarp operacijų ryšiai, identifikuojant, kurie kintamieji veikia kitus ir kaip reikšmės sklinda programoje.



1 pav. C/C++ programinio kodo funkcijos transformacija į AST, CFG ir PDG grafus.

3 Eksperimentinis tyrimas

Atliktas eksperimentas atskleidžia dėsningumus, kaip skirtingos kodo grafinės reprezentacijos veikia mašininio mokymosi modelių gebėjimą identifikuoti specifines pažeidžiamumų klases. Tyrimas susideda iš trijų etapų,

kurių metu gauti rezultatai interpretuojami remiantis požymių svarbos rodikliais, modelių generalizacijos galimybėmis bei struktūrinės informacijos įtaka klasifikavimo procesui.

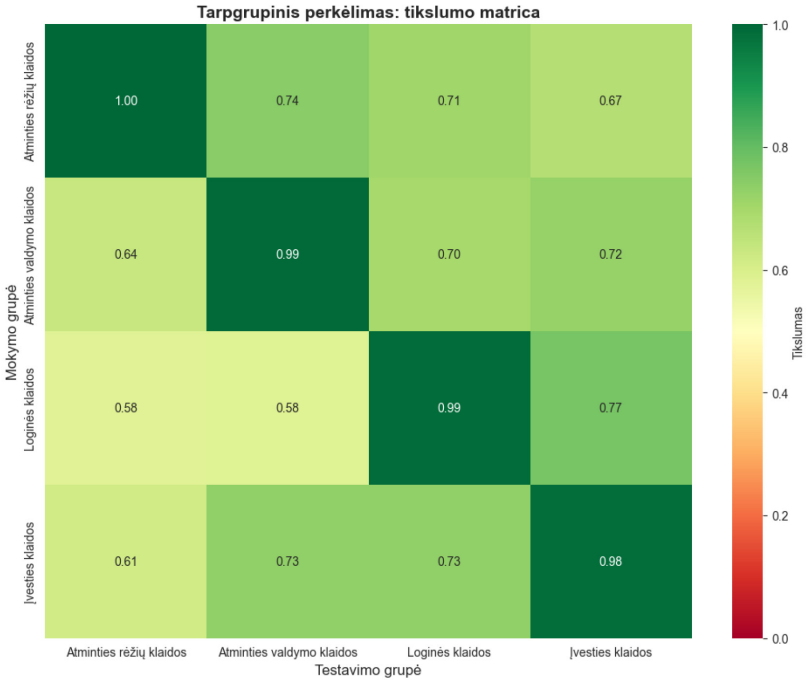
3.1 Individualių CWE modelių analizė ir požymių informatyvumo matrica

Pirmajame etape kiekvienai iš 16 CWE kategorijų buvo apmokyti atskiri „Random Forest" klasifikatoriai, o naudojant permutacinę požymių svarbos analizę (angl. Permutation Importance), nustatytas kiekvieno požymio indėlis į modelio priimamus sprendimus. Gauti rezultatai, susisteminti CWE ir požymių informatyvumo matricoje (žr. 2 pav.), atskleidžia, kad struktūriniai baziniai požymiai, tokie kaip literal_count, ast_nodes ir control_count, išlieka svarbūs daugumoje kategorijų. Rezultatai rodo, kad modeliai bazinę kodo struktūrą ir mazgų kiekį naudoja kaip pirminį filtrą pažeidžiamumams identifikuoti. Tačiau analizuojant specifines klaidų grupes, išryškėja reikšmingi požymių informatyvumo skirtumai. Pavyzdžiui, įterpties klaidų grupėje (CWE-78, CWE-90, CWE-134) esminę reikšmę įgyja taint_reachable ir sink_call_count rodikliai, kurie atminties valdymo klaidų grupėse neturi jokios diagnostinės vertės.

3.2 Kryžminės grupinės generalizacijos eksperimentas

Vertinant kryžminį modelių suderinamumą (žr. 3 pav.), pastebėta, kad pažeidžiamumų aptikimo sėkmė tiesiogiai priklauso nuo klaidų grupių struktūrinio panašumo. Analizė parodė, kad modeliai, apmokyti atpažinti tam tikros grupės klaidas, pasižymi aukštu tikslumu testuojant tos pačios grupės pavyzdžius (tikslumas siekia 0,98-1,00). Tai patvirtina, kad tos pačios grupės pažeidžiamumai pasižymi panašia grafine struktūra, todėl modelis juos efektyviai identifikuoja. Klasifikavimo kokybė ženkliai sumažėja bandant modelį pritaikyti skirtingo pobūdžio pažeidžiamumams, pavyzdžiui, loginių klaidų grupėje tikslumas siekia 0,59-0,73. Tai pagrindžia prielaidą, jog duomenų srautų pagrindu suformuoti požymiai nėra tiesiogiai pritaikomi sintaksinio pobūdžio klaidoms aptikti. Reikšmingas skirtumas tarp klaidų grupių rodo, kad kiekvienas pažeidžiamumo tipas jungtiniame grafe pasižymi jam būdingais ryšiais.

2 pav. Požymių svarbos pasiskirstymas pagal CWE kategorijas.



3 pav. Kryžminio grupių klasifikavimo tikslumo matrica.

3.3 Struktūrinių kodo apimties požymių įtakos vertinimas

Paskutiniame etape vertinta, kiek modelio sprendimai priklauso nuo tiesioginių kodo apimties rodiklių (angl. size bias). Tam tikslui iš analizės pašalinti požymiai, nusakantys funkcijos dydį, tokie kaip mazgų kiekis (ast_nodes) ar santykinis iškvietimų tankis (calls_per_ast_node). Atliktas eksperimentas parodė, kad pašalinus šiuos rodiklius bendras modelio tikslumas sumažėjo 17,6 % (nuo 0,972 iki 0,801), tačiau išliko reikšmingai aukštesnis už atsitiktinio spėjimo ribą, o tai patvirtina kodo grafinių savybių vertę nepriklausomai nuo jo apimties.

Detalesnė analizė pagal atskiras CWE kategorijas atskleidžia nevienodą struktūrinės informacijos įtaką skirtingiems pažeidžiamumų tipams. Pastebėta, kad tokioms kategorijoms kaip CWE-427, CWE-78 ar CWE-680 kodo dydžio pašalinimas neturėjo reikšmingos įtakos - ROC-AUC rodiklis išliko artimas 1,000 (atitinkamai 0,998, 0,992 ir 0,984). Rezultatai rodo, kad šiems

pažeidžiamumams identifikuoti modelis naudoja unikalius CPG briaunų dėsningumus ir loginę programos seką, o ne funkcijos apimtį. Priešingas dėsningumas stebimas CWE-121, CWE-126 ir ypač CWE-122 bei CWE-426 atvejais, kur ROC-AUC rodiklis ženkliai sumažėjo (pvz., CWE-122 atveju nuo 0,923 iki 0,407). Toks nuosmukis rodo, kad šių specifinių pažeidžiamumų atvejais modelis dažniau remiasi kodo sudėtingumo metrikomis, o ne grynąja semantika.

Nors kodo apimties požymiai suteikia modeliui papildomos informacijos diferencijuojant funkcijas, išlikęs stabilumas daugumoje kategorijų patvirtina, kad grafų reprezentacija perteikia giliąją programos logiką. Tai leidžia teigti, kad CPG struktūra sėkmingai koduoja pažeidžiamumų mechanizmus, kurie yra nepriklausomi nuo funkcijos dydžio, tačiau tam tikriems atminties rėžių ir valdymo defektams aptikti struktūrinis kodo kontekstas išlieka svarbus diagnostinis elementas.

1 lentelė. ROC-AUC su ir be dydžio požymių pagal CWE tipą.

CWE	ROC-AUC (top-9)	ROC-AUC (be dydžio)	Skirtumas
CWE-367	1,000	1,000	-0,000
CWE-427	1,000	0,998	-0,002
CWE-90	1,000	0,998	-0,002
CWE-78	1,000	0,992	-0,008
CWE-680	1,000	0,984	-0,016
CWE-415	1,000	0,959	-0,041
CWE-416	0,991	0,956	-0,035
CWE-476	0,988	0,924	-0,064
CWE-190	0,989	0,924	-0,065
CWE-134	0,985	0,904	-0,081
CWE-127	0,992	0,888	-0,104
CWE-124	0,991	0,939	-0,052
CWE-121	0,980	0,667	-0,313
CWE-126	0,988	0,727	-0,261
CWE-426	0,962	0,508	-0,454
CWE-122	0,923	0,407	-0,517

4 Tyrimo ribotumai

Šis tyrimas turi keletą apribojimų, susijusių su duomenų rinkiniu, modelio pasirinkimu ir taikymo sritimi. Naudotas „Juliet Test Suite“ duomenų rinkinys yra sintetinis, todėl jame pateikiamos funkcijos yra paprastesnės nei realaus pasaulio programinis kodas. Realiuose projektuose dažnai pasitaiko sudėtingesnės valdymo struktūros, tarpmodulinės priklausomybės ir subtilesni pažeidžiamumai, todėl modelio našumas praktinėse situacijose gali būti žemesnis. Tyrime taikytas „Random Forest“ modelis negali tiesiogiai modeliuoti grafinių struktūrų ar ilgų nuotolių priklausomybių, nes kodo reprezentacijos buvo apibendrintos į požymių rinkinius, todėl dalis struktūrinės informacijos gali būti prarandama. Be to, tyrimas apsiriboja C/C++ programavimo kalba, todėl gauti rezultatai nėra tiesiogiai perkeliami į kitas kalbas, pasižyminčias skirtingais atminties valdymo principais ir pažeidžiamumų tipais. Galiausiai, tyrimas orientuotas tik į CWE pažeidžiamumus, todėl neapima kitų svarbių saugumo problemų kategorijų, tokių kaip kriptografinės silpnybės ar prieigos kontrolės pažeidimai.

5 Išvados

Atliktas tyrimas rodo, kad programinės įrangos saugumo patikros negali remtis vienu universaliu požymių rinkiniu, nes skirtingų tipų klaidų aptikimo sėkmė tiesiogiai priklauso nuo analizei pateikiamų kodo savybių. Gauti rezultatai rodo, kad iš jungtinio kodo savybių grafo (CPG), apjungiančio AST, CFG ir PDG reprezentacijas, išvesti požymiai leidžia efektyviai identifikuoti pažeidžiamumus, tačiau jų aptikimui reikalingi požymiai reikšmingai skiriasi. Taip pat nustatyta, kad skirtingoms CWE pažeidžiamumų klasėms yra svarbūs skirtingų kodo reprezentacijų pagrindu išvesti požymiai. Nustatyta, kad baziniai kodo struktūros rodikliai yra pakankami bendro pobūdžio klaidoms nustatyti, tačiau įterpties pažeidžiamumams aptikti būtina analizuoti duomenų srautus, kurių reikšmė kitose kategorijose yra mažesnė.

Kryžminio grupių klasifikavimo rezultatai parodė, kad modelis, išmokytas atpažinti vienos grupės klaidas, sunkiai identifikuoja kitokio pobūdžio pažeidžiamumus. Tai rodo, kad skirtingos pažeidžiamumų klasės pasižymi skirtingais struktūriniais ir semantiniais dėsningumais, todėl kuriant automatizuotus saugumo analizės įrankius tikslinga taikyti specializuotus modelius atskiroms klaidų grupėms, o ne vieną universalų sprendimą. Toks požiūris leidžia pasiekti didesnį tikslumą ir efektyviau išnaudoti specifinius kodo požymius.

Taip pat nustatyta, kad kodo apimtis daro įtaką modelio tikslumui, tačiau nėra lemiamas veiksnys pažeidžiamumams aptikti. Eksperimentas, kuriame pašalinti dydžio rodikliai, parodė, kad modelis išlaiko gebėjimą atpažinti klaidas remdamasis kodo logine struktūra. Nors tam tikrais atvejais kodo dydžio informacija gali pagerinti rezultatus, daugumoje kategorijų iš grafinių reprezentacijų išvesti požymiai perteikia pakankamai informacijos pažeidžiamumams identifikuoti. Praktiniu požiūriu tai reiškia, kad analizės įrankiai gali būti patikimi ne dėl statistinių kodo rodiklių, o dėl gebėjimo suprasti pačią klaidos prigimtį, taip sumažinant nereikšmingų pranešimų kiekį saugumo patikros metu.

Tolesni tyrimai galėtų būti orientuoti į sudėtingesnių modelių, pavyzdžiui, grafinių neuroninių tinklų ar kitų gilesnių mokymosi metodų, taikymą siekiant geriau išnaudoti kodo struktūrinę informaciją ir tiesiogiai modeliuoti grafines reprezentacijas. Taip pat svarbu įvertinti modelio veikimą realaus pasaulio projektuose, siekiant nustatyti jo praktinį pritaikomumą. Be to, tikslinga tirti metodo pritaikymą kitoms programavimo kalboms, pritaikant požymių rinkinį pagal jų specifiką ir dažniausiai pasitaikančius pažeidžiamumus.

Literatūra

[1] M. Alqaradaghi and T. Kozsik, "Comprehensive Evaluation of Static Analysis Tools for Their Performance in Finding Vulnerabilities in Java Code," *IEEE Access*, vol. 12, p. 55824, 2024, doi: 10.1109/ACCESS.2024.3389955.

[2] Heckman, S., & Williams, L. (2011). A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, 53(4), 363-387.

[3] Omri, S., & Sinz, C. (2021). Machine learning techniques for software quality assurance: A survey. arXiv preprint arXiv:2104.14056.

[4] Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014, May). Modeling and discovering vulnerabilities with code property graphs. In 2014 IEEE symposium on security and privacy (pp. 590-604). IEEE.

[5] Wang, H., Ye, G., Tang, Z., Tan, S. H., Huang, S., Fang, D., ... & Wang, Z. (2020). Combining graph-based learning with automated data collection for code vulnerability detection. IEEE Transactions on Information Forensics and Security, 16, 1943-1958.

[6] NSA Center for Assured Software, "Juliet C/C++," 01 Oct 2017.

[7] de Kraker, W., Vranken, H., & Hommersom, A. (2025). MultiGLICE: Combining Graph Neural Networks and Program Slicing for Multiclass Software Vulnerability Detection. Computers, 14(3), 98.

[8] Chen, S. S., Hsu, Y. S., Lin, T. C., Chen, C. K., & Sun, C. Y. Enhancing Static Vulnerability Alert Validation using Large Language Models. Available at SSRN 5845441.

[9] Bin Mohd Illzam Elahee, M. (2025). A Software Vulnerabilities Detection Framework Based on Graph Neural Network with Graph-Based Feature Representations (Doctoral dissertation, Swinburne).