Gépi tanulás alapú hiba és sérülékenység előrejelzés

Aladics Tamás

Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged, 2021

Témavezető:

Dr. Ferenc Rudolf

Ph.D. értekezés tézisei

Szegedi Tudományegyetem
Informatika Doktori Iskola

Bevezetés

Ebben az egyre inkább összekapcsolódott világban a szoftverrendszerek a modern társadalom gerincévé váltak, és mélyen befolyásolják a mindennapi életünk minden aspektusát, a mobilalkalmazásoktól a közlekedést és az egészségügyet kezelő komplex rendszerekig. A szoftveres megoldások elterjedése, valamint a képernyőidő exponenciális növekedése, különösen a COVID-19 világjárvány idején, jelentősen átalakította az életünket, munkaviszonyainkat és a kommunikációs módjainkat, kényelmet és hatékonyságot hozva [20]. Ez a gyors integráció azonban ezeket a rendszereket a kibertámadások fő célpontjaivá is tette, a sebezhetőségek növekedése pedig a biztonsági incidensek és adatszivárgások riasztó növekedéséhez vezetett. A International Business Machines Corporation (IBM) X-Force Threat Intelligence Index 2020-as jelentése 2019-ben körülbelül 8,5 milliárd kibertámadást azonosított, ami a megelőző évhez képest több mint 200 százalékos növekedést jelent [29]. Ezek az incidensek gyakran a szoftverfejlesztés hiányos felügyeléséből és a nem megfelelő biztonsági protokollokból erednek. Ez kiemeli a fejlesztők számára a biztonság priorizálásának, valamint a szervezetek számára a rendszerek folyamatos monitorozásának és frissítésének szükségességét, különösen mivel egyre több területen az alapvető, kritikus infrastruktúra a digitális technológiákra támaszkodik [28].

Felismerve a szoftverrendszereket veszélyeztető sérülékenységeket, a tudományos és ipari közösség a biztonságközpontú megközelítés felé mozdult el mind a szoftverfejlesztésben, mind a szoftverfelügyeletben, túllépve a hagyományos, manuális beavatkozásokra épülő megközelítéseket, mint például a kódellenőrzések és a penetrációs tesztelés. Bár ezeknek a hagyományos módszereknek megvan a maguk haszna, jelentős időt és erőforrásokat igényelnek, és előfordulhat (főként a kódellenőrzések esetén), hogy nem veszik észre a rejtettebb problémákat. Válaszul erre jelentek meg az automatizált szoftverelemzési technikák, amelyek kiegészítik a manuális megközelítéseket. A statikus forráskód elemzés, amely a végrehajtás nélküli kódellenőrzésekkel előre definiált mintákon keresztül azonosítja a sebezhetőségeket, valamint a dinamikus forráskód elemzés, amely valós időben figyeli a szoftver viselkedését az anomáliák észlelése érdekében, a hagyományos vizsgálatokkal együtt hatékony stratégiát nyújt a szoftverbiztonság növelésére.

Annak ellenére, hogy az automatizált módszerek jelentős előrelépést jelentenek a manuális vizsgálatokkal szemben, a skálázhatóság és az alkalmazkodóképesség kihívásaival néznek szembe a szoftveres sebezhetőségek változó természete miatt. Az ezekben alkalmazott minta alapú megoldások az új sebezhetőségek megjelenésével szemben gyorsan elavulttá válhatnak. Emellett a modern szoftverrendszerekben található hatalmas mennyiségű kód egyre célszerűtlenebbé teszi a kézi és a minta alapú automatizált elemzéseket.

A gépi tanulás (ML) integrálása az információs technológiába, különösen a szoftverbiztonság területén, kulcsfontosságú előrelépést jelent, olyan megoldásokat kínálva, amelyek alkalmazkodhatnak a minta alapú elemzéseken túlmutató adatokhoz és tanulnak azokból. Az ML-algoritmusok azon képessége, hogy a múltbeli sebezhetőségekből és incidensekből tanulva képesek jelezni és felismerni az új fenyegetéseket, jelentős előrelépés lehet. Ebben a disszertációban bemutatom ezen területhez kapcsolódó hozzájárulásaimat, ami magában foglalja a DeepWater-Framework fejlesztésében való részvételt, amely egy olyan eszköz, amely az alapvető ML pipeline folyamatok, például a végrehajtás, a hiperparaméter-optimalizálás és az értékelés automatizálására szolgál elosztott rendszereken [14]. Ez a keretrendszer kulcsfontosságú szerepet játszott kutatásainkban, megkönnyítve a szoftverbiztonsághoz kapcsolódó különféle ML-megközelítések fejlesztését és összehasonlítását, demonstrálva az ML lehetőségét az automatizált analízisben és a proaktívabb, robusztusabb biztonsági megoldások kidolgozásában.

A gépi tanulás szoftverbiztonságban való alkalmazása kifejezetten hatékony összetett minták megértésében és reprezentálásában, például a forráskód beágyazásán keresztül. Ez a technika,

amely kulcsfontosságú az ML alkalmazások számára, a kódot olyan numerikus vektorokká alakítja át, amelyek rögzítik a szemantikai összefüggéseket. Az olyan algoritmusok alkalmazásával, mint a Doc2Vec, az ML modellek proaktívan megjósolhatják a sebezhetőségeket, lehetővé téve a korai beavatkozást. Kutatásunk konkrétan a Doc2Vec-et alkalmazta Java forráskódra, és olyan numerikus vektorokat állított elő, amelyek alapján az ML modellek szemantikai mintákat ismerhetnek fel és potenciális hibákat jósolhatnak meg, bemutatva a gépi tanulás területében rejlő lehetőségeket, például javítja a sebezhetőségek észlelését, és hozzájárul a biztonságosabb szoftverrendszerek fejlesztéséhez [6].

A szoftveres sebezhetőségek növekvő tendenciája rávilágít a korai felismerés fontosságára, különösen a kommit bejegyzés időpontjában – amikor a kód megosztásra kész, de még nem teljesen integrált. A Just-In-Time (JIT, "valós idejű") sebezhetőség előrejelzés, mint terület, kiemelten kezeli az időszerű és proaktív biztonsági szempontokat. A JIT előrejelzés közeponti eleme a kódban bekövetkező változások, átalakítások hatékony reprezentálása kommit reprezentációkon keresztül. Abból fakadóan, hogy a sebezhetőséget kiváltó kommitokhoz tartozó adatkészletek hiánya kihívást jelent, kutatásunk egy olyan módszert mutat be, amellyel ilyen adathalmazokat hozhatunk létre a meglévő hibajavítási kommit adatkészletekből [4]. Az SZZ algoritmust egy relevancia alapú szűrési lépéssel követve egy Java sebezhetőségekre összpontosító adathalmazt hoztunk létre, amely elősegíti a valós idejű sebezhetőség észlelési módszerek fejlesztését és azok értékelését.

Ezt követően, a JIT sebezhetőség előrejelzéssel kapcsolatos vizsgálatódásaink során azonosítottuk a forráskód strukturális információit beépítő módszerek hiányát, amelyek gyakran olyan formákban jelennek meg, mint az absztrakt szintaxisfák (AST-k). Definiáltuk a Code Change Tree (CCT) struktúrát, amely az AST-ket használva hoz létre egy forráskód változási fát, amely rögzíti a strukturális különbségeket az AST szintjén a kommit előtti és utáni állapotok között. Megközelítésünket gépi tanulási modellek segítségével értékeltük, miután Doc2Vec segítségével a fastruktúrákat vektorokká alakítottuk [3]. A pontosabb értékelés érdekében a CC2Vec-kel és a DeepJIT-tel szembeni összehasonlító vizsgálat során elemeztük teljesítményüket a JIT sebezhetőség előrejelzésben, a fals pozitív arány és előrejelzéseik granularitása területén [5]. Eredményeink azt mutatták, hogy a CCT bár testreszabhatóbb részletességű előrejelzésre ad lehetőséget, alacsonyabb teljesítményt mutat a CC2Vec-hez és a DeepJIT-hez képest, amelyek jobb prediktív hatékonyságot mutattak a kommit szintű előrejelzésekben. Bízunk benne, hogy ezek az eredmények hasznosnak bizonyulnak a granularitás és a prediktív pontosság közötti kompromisszumok vizsgálatában a különböző kommit reprezentációs módszerek esetén.

A disszertáció négy tézispontot tartalmaz. Jelen tézisfüzetben összegezzük a tézispontokban elért eredményeket.

№	[6]	[4]	[3]	[5]
I.	◆			
II.		◆		
III.			◆	
IV.				◆

1. táblázat. A tézispontokhoz tartozó publikációk

I. Hiba előrejelzés Doc2Vec alapú forráskód beágyazással

Bár kritikus fontosságú a szoftverminőség biztosítása szempontjából, a hibák felderítése továbbra is kihívást jelentő feladat a statikus kódelemző eszközök korlátai miatt. Ezek az eszközök gyakran nem elég robusztusak, mivel előre definiált mintákra támaszkodnak. A gépi tanulás ígéretes alternatívát kínál, mivel nagy mennyiségű adatból képes adaptív módon mintákat tanulni, potenciálisan javítva a hibajelzést. Azonban a gépi tanulási megközelítések esetében kritikus aspektus, hogy hogyan reprezentálják a bemeneti adatokat, milyen jellemzőkön történik a tanulás. A szoftverelemzés összefüggésében ezek a jellemzők közvetlenül a forráskódból származnak, és különböző formákat ölthetnek, például lehetnek kódmetrikák, függvények vagy osztályok halmazai, sőt, akár teljes programok is. Ezek a reprezentációk eltérhetnek struktúrájukban (tokenek, függvények stb.) és az alkalmazott formában (szöveg, absztrakt szintaxisfa stb.). Tekintve a kódreprezentáció fontosságát ezen a területen, ez a tézispont azt vizsgálja, hogy mennyire hatékony az absztrakt szintaxisfából (AST) származó jellemzők használata a hiba előrejelzésben a kódmetrikákhoz képest.

Pontosabban, egy olyan hibajelzési módszert javasolunk, amely egy Doc2Vec-en alapuló kódbeágyazást használ [21], ami egy hatékony eszköz az elosztott dokumentum reprezentációk tanulásához. A Doc2Vec a jól ismert Word2Vec algoritmus továbbfejlesztése, amely az adott szavak beágyazását a szöveg korpuszban a környező szavak figyelembevételével alakítja ki. A Doc2Vec erre a koncepcióra épít, azzal a töblettel, hogy egy dokumentumvektort is tanul, amely egy dokumentum (szavak tetszőleges halmaza) szemantikai jelentését rögzíti. E tézipoint kontextusában a Java osztályokat (beleértve az enumerációkat és interfészeket is) kezeljük alapvető egységként, tokenek sorozatát generálva belőlük. Ezeket az osztályszekvenciákat ezután dokumentumként kezeljük egy Doc2Vec modellben, az egyes tokenek pedig a dokumentumon belüli szavakként funkcionálnak. A Doc2Vec fix méretű vektorreprezentációt generál minden egyes osztályhoz, amely ezután a hiba előrejelzési feladatoknál jellemzővektorként használható.

Kísérletezéseink során különböző Doc2Vec paraméterezéseket vizsgáltunk, majd a generált vektorokat kétféleképpen használjuk fel a hibajelzésre:

- Önálló jellemzők: A vektorokat közvetlenül bementként használjuk a gépi tanulási modellekhez.
- Kombinált jellemzők: A vektorokat hagyományos kódmetrikákkal kombináljuk, hogy egy átfogóbb jellemző halmazt alkossunk.

A forráskód strukturális információinak kinyerése érdekében köztes reprezentációként az absztrakt szintaxisfát (AST) használjuk. Az AST-t mélységi kereséssel járjuk be, és minden egyes megtalált csomópont típusát (pl. osztálydefiníció, változóhasználat) hozzáadjuk az AST-hez tartozó szekvenciához. A kódon belüli scope változások figyelembevétele érdekében egy speiális konstans értéket adunk a sorozathoz, amikor a fában visszalépünk. Ez segít megkülönböztetni az olyan kódblokkokat, amik a scope információkon kívül nem különböznének.

Elmondható tehát, hogy a Doc2Vec alkalmazása melletti döntésünk két fő szempontból ered. Először is, intuitív módon az osztályokat „dokumentumoknak”, kód elemeiket pedig „szavaknak” tekinthetjük. Másodszor, a Doc2Vec rögzített hosszúságú vektorokat produkál, ami egyszerűsíti a különböző gépi tanulási modellekkel való integrációjukat. A megközelítés értékeléséhez az Unified Bug Dataset-et ([13]) használtuk, amely egy nagyméretű, Java kódhibákból álló adatbázis. Ez az adathalmaz különféle forrásokból származik, és a különböző forráskód elemekkel társított hibák gyűjteményét nyújtja. Az osztályokra, interfészekre és felsorolásokra összpontosít: 48,719 entitást ölel fel, amiből 8,242-t hibásnak azonosít.

2. táblázat. Különböző gépi tanuló modellek összehasonlítása ugyanazon forráskódbeágyazás esetén F_1-értékekben mérve)

Model név	Beágyazás	Kód metrika	Kombinált
Bayes	0.301	**0.325**	**0.325**
Lineáris	0.298	0.401	**0.418**
Logisztikus	0.311	0.412	**0.430**
Döntési Fa	0.374	**0.475**	0.461
Random Forest	0.423	0.515	**0.522**
CDNNC	0.451	0.474	**0.502**
SDNNC	0.467	0.520	**0.533**
KNN	0.463	0.502	**0.524**

A kódmetrikák frissítéséhez és bővítéséhez az OpenStaticAnalyzer eszközt használtuk [8], amely olyan bevált szoftvermetrikákat tartalmaz, mint a sorok száma (LOC), a metódusok száma (NM) stb. Emellett kutatásunkban kulcsfontosságú szerepet játszott a Deep Water Framework (DWF) [14], amely segítségével a különböző Doc2Vec paraméterezésekkel és gépi tanulási modellekkel kísérleteztünk. Modelleink halmaza hagyományos algoritmusokat tartalmaz, mint például a random forest, a döntési fák, a k-legközelebbi szomszéd (KNN), az SVM, a naiv Bayes, a lineáris osztályozó, logisztikus regresszió, valamint két neurális hálózati architektúra: standard mély neurális hálózat (SDNNC) és egyéni mély neurális hálózat (CDNNC). Emellett 10-fold keresztvalidációt és előfeldolgozási stratégiákat implementáltunk, mint például a normalizálás, standardizálás és a felülmintavételezés.

A 2. táblázatban látható eredmények alapján megállapítható, hogy bár a Doc2Vec beágyazások összehasonlítható teljesítményt nyújtanak a hagyományos kódmetrikákkal, az utóbbiak általában kissé jobban teljesítenek. Egyetlen Doc2Vec beállítás sem bizonyult univerzálisan optimálisnak az összes gépi tanulási modell esetében, ami kiemeli a paraméterhangolás szükségességét. A Doc2Vec beágyazások kódmetrikával való kombinálásakor az eredmények következetesen javultak, ami arra utal, hogy a beágyazások értékes szemantikai információkat tartalmaznak, amelyek a kódmetrikából önmagukban hiányoznak. A forráskód beágyazások és a kódmetrikák közötti szinergia nemcsak a hiba előrejezés pontosságát növeli, hanem azt is megerősíti, hogy a Doc2Vec fontos betekintést nyújt az adatokba.

A hiba előrejezéás céljából létrehozott adatkhalmazt is tartalmazó replikációs csomag elérhető a következő címen: `http://doi.org/10.5281/zenodo.4724941`

A szerző hozzájárulása

A disszertáció szerzője részt vett a forráskód reprezentációs algoritmus definiálásában. A szerző végezte el a modellbetanítást és hiperparaméter hangolást. Megtervezte a modell kiértékelésének a módszertanát. Dimenziócsökkentést végzett el az egyes forráskód példákon, hogy bemutassa a beágyazás hatékonyságát. A statikus kódelemző eszközt is ő futtatta le az adatbázis elemeire, így állítva elő a forráskód metrikákat.

- **Tamás Aladics**, Judit Jász, Rudolf Ferenc. Bug Prediction Using Source Code Embedding Based on Doc2Vec. Computational Science and Its Applications 21st International Conference (ICCSA 2021), Cagliari, Italy, September 13-16, 2021, Proceedings, Part VII, volume 12955 of Lecture Notes in Computer Science, pages 382–397. Springer, 2021. `https://link.springer.com/chapter/10.1007/978-3-030-87007-2_27`

II. Sérülékenységeket bevezető Java kommit adathalmaz és továbbfejlesztett SZZ alapú módszer

A szoftveres sérülékenységek jelentős és folyamatosan növekvő fenyegetést jelentenek. Ennek következménye, hogy az gyorsan fejlődő gépi tanulási technikákat egyre inkább alkalmazzák a szoftverfejlesztési feladatokban, például a minőségbiztosításban, hogy kezeljék ezeket a kihívásokat. Ezen technikák sikere nagymértékben függ a megfelelő adatkészletek elérhetőségétől a képzéshez és az értékeléshez. Bár a sebezhetőségjavító kommitokat (Vulnerability Fixing Commit - VFC) tartalmazó adatkészletek viszonylag könnyen találhatóak, a sebezhetőséget bevezető kommitok (Vulnerability Introducing Commit - VIC) esetében rosszabb a helyzet. Ez a hiány korlátokat állít olyan kritikus területek kutatásában, mint például a valós idejű sebezhetőség észlelés és lokalizáció [7, 12, 22].

A meglévő sérülékenység javító adathalmazok gyakran nyilvánosan közzétett adatbázisokra épülnek, mint például a Common Vulnerabilities and Exposures (CVE) [1] és a National Vulnerability Database (NVD) [10]. Ezek az források jellemzően részleteket tartalmaznak a sérülékenységről, és linkeket is szolgáltatnak a releváns javító patchekhez. A VFC adatkészletek elérhetősége ellenére a sérülékenységeket bevezető megfelelő kommitok pontos meghatározása kihívás marad. A folyamat automatizálására tett jelenlegi kísérletek gyakran heurisztikák alapján működnek, és nehezen skálázhatóak.

A probléma megoldására egy kétfázisú módszert javasolunk a VIC adatkészletek szisztematikus előállítására, a könnyen elérhető VFC adatkészletekből:

- **1. fázis: Kandidátus kommitok kiválasztása** Megközelítésünk középpontjában az SZZ algoritmus egy finomított implenetációja, az SZZ Unleashed [11] áll. Az SZZ alapú algoritmusok kiválóan alkalmasak arra, hogy egy sérülékenységjavító commit (VFC) sorait visszavezessék a repository előzményein keresztül, hogy meghatározzák azokat a kommitokat, amelyek potenciálisan bevezették a hibás kódot. Az SZZ Unleashed VFC-k halmazára való alkalmazásával megkapjuk a jelölt (kandidátus) VIC-k kezdeti halmazát.

- **2. fázis: Szűrés és relevanciaértékelés** Az SZZ algoritmusok által generált nagymennyiségű adat miatt kimenetük gyakran fals pozitív eredményeket tartalmaz, továbbá az SZZ nem nyújt segítséget az eredmények rangsorolásában, szűrésében. E problémák orvoslárásra egy szűrési fázist javaslunk, amely minden jelölt VIC-hez egy relevancia pontszámot rendel. Ez a pontszám számszerűsíti, hogy a VIC változásai milyen mértékben korrelálnak a vonatkozó VFC-ben végrehajtott módosításokkal, így biztosítva metrikát a priorizáláshoz.

Módszerünk fontos eleme a relevancia pontszám kiszámítása. Minden egyes jelölt VIC esetében végigiterálunk a módosított fájlokon, és megkeressük a megfelelő fájlokat a hozzá tartozó VFC-ben. Ha egyezést találunk, kiszámítunk egy hozzájárulási (kontribúciós) pontszámot, amely két tényezőt ötvöz:

- Fájlhasonlóság: Megmérjük a VIC és VFC megfelelő fájljai közötti hasonlóságot, például az azonos sorok arányát, a szóközöket figyelmen kívül hagyva a robusztusság érdekében.

- Alap pontszám: A fájl relatív fontosságát becsüljük meg a VFC módosításain belül, kiszámítva a fájlban módosított sorok arányát a VFC-ben végrehajtott összes módosításhoz képest. Ily módon a javításban nagyobb szerepet játszó fájlok magasabb alap pontszámokat kapnak.

Egy VIC-hez tartozó végső relevancia pontszám az összes olyan fájl hozzájárulási pontszámának összege, amelyeknek vannak megfelelőik a VFC-ben (azaz csak olyan fájlok esetén számítjuk, ami a VIC-ben és VFC-ben is metalálható). Ez a pontszám indikátorként használható annak becslésére, hogy az adott VIC valóban sérülékenységet előidéző kommithoz vezet-e.

Módszerünk praktikusságának bemutatására kifejlesztettük a *BugIntroducerMiner* és a *FilterBugIntroducer* eszközöket. Ezeket használtuk egy új Java VIC-ket tartalmazó adatkészlet generálására a projekt-KB adatbázisból. Ez az adathalmaz, amely 564 VFC-bejegyzést tartalmaz, mindegyikhez legfeljebb két, de legalább egy VIC-t hozzárendelve, értékes eszköz lehet a szoftverbiztonsági kutatások számára. Módszerünk jelentősen finomítja a nyers SZZ-kimenetét, a VFC-nkénti 1-700 kandidátus kommitról kezelhetőbb számra (ez a módszer választható paramétere, ebben az esetben 2-re állítottuk) csökkentve azt. A generálási folyamat során 198 nyílt forráskódú projekt több mint 110 000 fájlját vizsgáltuk a kétlépcsős folyamat részeként.

Az adathalmaz és a használt eszközök elérhetőek: `https://doi.org/10.5281/zenodo.5785239`

A szerző hozzájárulása

A szerző részt vett a kapcsolódó munkák feltárásában és áttekintésében. Részt vett a módszertan megtervezésében. Felismerte a második, szűrési fázis szükségességét. Definiálta a relevancia pontszámot. A szerző részt vett a rendelkezésre álló SZZ implementációk vizsgálatában és a szükséges környezet felállításában. Ő volt a sérülékenység bevezető adathalmaz generálásában használt eszközökök fejlesztője. Részt vett az eredmények értékelésében és értelmezésében.

- **Tamás Aladics**, Péter Hegedűs, Rudolf Ferenc. A Vulnerability Introducing Commit Dataset for Java: An Improved SZZ based Approach. In Proceedings of the 17th International Conference on Software Technologies - ICSOFT, pages 68–78. INSTICC, SciTePress, 2022. `https://www.scitepress.org/Link.aspx?doi=10.5220/0011275200003266`

III. AST-alapú forráskód változás reprezentáció és annak valós idejű sebezhetőség előrejelzés teljesítménye

A szoftveres sérülékenységek eszkalálódó száma kritikus kihívást jelent a szoftverbiztonság szempontjából. Ez a gyors növekedés, amelyet 2020-ban a nyílt forráskódú sérülékenységek 50%-os növekedése is kiemel [19], megerősíti a terület fontosságát. A "valós idejű" (JIT) sérülékenység előrejelzés célja, hogy ezt a kihívást azáltal kezelje, hogy a sérülékenységeket már a kódba való bevezetésükkor észleli. Ez a proaktív megközelítés lehetőséget kínál a biztonsági kockázatok minimalizálására és a hibajavítással járó költségek csökkentésére.

A hagyományos sérülékenység előrejelző modellek (VPM-ek) gyakran statikus elemző eszközökre és szoftvermetrikákra támaszkodnak. Ezeket a technikákat azonban korlátozhatják a skálázhatósági problémák, a magas hamis pozitív arányok és az új sérülékenységi mintákhoz való alkalmazkodás nehézségei [18][9][25]. A gépi tanulás (ML) ígéretes alternatívát kínál, de sikere nagymértékben függ a képzéshez használt adatok minőségétől. A just-in-time (JIT, "valós idejű") sérülékenység előrejelzés egyik alapvető kihívása a commit előtti és utáni kódállapotok közötti különbségek hatékony reprezentálása. A legújabb kutatások azt mutatják, hogy a meglévő metrika és szöveges jellemzőkön alapú módszerek nem képesek hatékonyan erre a repreztációra [23]. Ennek következtében lépett fel az olyan új kódváltoztatási reprezentációs technikák iránti igény, amelyek alkalmasabbak lehetnek a gépi tanulással elősegített sérülékenység előrejelzésre.

Ebben a szakaszban egy új megközelítést mutatunk be a forráskód változások reprezentálására, amit Forráskód Változási Fának (Code Change Tree - "CCT") nevezünk. A CCT-k célja, hogy hatékonyan reprezentálják a forráskód változásokat "valós idejű" (JIT) sérülékenység előrejelzés céljából. Felépítésük több lépést foglal magában. Először, absztrakt szintaxisfákat (AST) generálunk mind a változás előtti (S_{pre}), mind a változás utáni (S_{post}) kódokra. Ezután minden AST-t egyedi gyökér-levél ("gyökérutak") utak halmazává alakítunk. Ezt követően elvetjük S_{pre} azon gyökérutait, amelyek megegyeznek S_{post} gyökérutaival, így csak a módosításokat tartjuk meg. A fennmaradó, eltérő gyökérutak alkotják a Forráskód Változási Fát, hatékonyan reprezentálva a szerkezeti módosításokat.

A CCT-k egyik döntő szempontja az AST-n belül használt csomópont azonosítás. Az általunk használt séma figyelembe veszi a csomópontok típusát, értékét és kontextuális információit (pozíció az AST-n belül), hogy lehetővé tegye az összehasonlításokat különböző AST-k között is. Továbbá, a CCT-k rugalmasak a granularitás tekintetében - képesek különböző szinteken (utasítás, metódus, osztály, stb.) reprezentálni a változásokat, mivel ez a módszer alkalmazható bármely AST-vel rendelkező kódelemre.

A CCT reprezentációk (és hasonlóképpen a nyers AST reprezentációk) gépi tanulási modellekbe való integrálásához Doc2Vec beágyazást alkalmazunk. Egy Doc2Vec modell egy nagyméretű Java metódus gyűjteményen (korpuszon) kerül betanításra, hogy megtanulja a Java specifikus szemantikákat. Ez a modell ezután a kilapított CCT-ket (és AST-ket) fix hosszúságú vektorokká transzformálja, alkalmassá téve őket gépi tanuló modellek számára.

Összefoglalva, ebben a tézispontban három kódváltozás reprezentációs módszert vizsgálunk:

- Metrika-alapú: A változásban érintett függvényekre szoftvermetrikákat számítunk, majd az S_{pre} és S_{post} állapotokhoz tartozóakat összefűzzük.
- Egyszerű kódváltozás: Az S_{pre} és S_{post} állapotokhoz AST-ket generálunk, majd tokenszekvenciákká lapítjuk őket, végül azokat összefűzzük.
- Kódváltozási Fa (CCT): Gyökérutakat vonunk ki minden AST-ből. Az S_{pre} S_{post}-hoz viszonyított változásait gyökérút különbségek alapján azonosítjuk, így kialakítva a CCT-t. A már bemutatott csomópont azonosítási sémát alkalmazunk az így kapott, különböző AST-k közötti összehasonlításokhoz.

A kiértékeléshez az előző tézispontban generált SZZ alapú sérülékenység bevezető kommit (VIC) adathalmazt használjuk [4]. Ezt az adathalmazt a project-KB VFC adatbázisból származtattuk SZZ algoritmussal generált kandidátus kommitok találatával, majd relevanciapontszám alapú szűrés segítségével.

Kísérleteink a Kódváltozási Fák (CCT-k), mint kódváltozások reprezentálására szolgáló módszer erősségeit bizonyítják. Az AST alapú reprezentációs formák előnye a metrika alapúakkal szemben nyilvánvaló, mivel az átlagos F_1-érték legalább 8%-kal nő. Ez a megállapítás tovább erősíti, hogy az AST alapú reprezentációk olyan (valószínűleg szerkezeti) információkat ragadnak meg, amelyeket a metrikák nem. A két AST-alapú megközelítés között nem jelentős, de mégis észrevehető különbség látható, mivel a Kódváltozási Fa közel 2%-kal jobb teljesítményt nyújt (lásd 3. táblázatot). Ez kiemeli a strukturális változások felismerésének fontosságát – ami a CCT-k egyik kiemelt erőssége, amely potenciálisan túlmutat a metrikák vagy az egyszerű AST alapú laposítás által szolgáltatott információkon.

Ezenkívül a CCT-k jelentősen csökkentették a kódváltozás reprezentációk átlagos méretét az egyszerű kódváltozás módszerhez képest. 59 340 függvényt tartalmazó adathalmazunkon a CCT-k átlagos csomópontszáma 51 volt, szemben az egyszerű változás reprezentáció 174 csomópontjával

3. táblázat. Különböző forráskód beágyazási módszerek eredményei (F1-érték)

Véletlen találgatás	20%							
	Adaboost	**CDNNC**	**Forest**	**KNN**	**Logistic**	**SDNNC**	**Tree**	**Átlag**
Metrikák	37%	32%	38%	38%	19%	29%	37%	33%
Simple	41%	40%	42%	**47%**	30%	**44%**	40%	41%
Change Tree	**41%**	**44%**	**43%**	43%	**38%**	44%	**42%**	**42%**

- ami több, mint 70%-os csökkenést jelent. Ez kiemeli a CCT-k azon képességét, hogy izolálják a változás legfontosabb aspektusait, ami előnyös a későbbi gépi tanulási feladatok számára.

A pontosság javulása és a kompakt reprezentáció kombinációja a CCT-ket ígéretes választássá teszi a sérülékenység észleléshez. Az a képességük, hogy a kódváltozás alapvető elemeire összpontosítanak, jelentősen javíthatja a "valós idejű" sérülékenység előrejelző rendszerek sebességét és megbízhatóságát.

A szerző hozzájárulása

A szerző részt vett a kapcsolódó szakirodalom áttekintésében. A módszertan kialakítása is főként a szerző munkája volt. A kiértékelésénél használt forráskód reprezentációkat is ő definiálta és implementálta. A modellek tanítási és hiperparaméter hangolási környezetét (DeepWaterFramework) a szerző állította össze és futtatta. Részt vett az eredmények kiértékelésében és értelmezésében.

- **Tamás Aladics**, Péter Hegedűs, and Rudolf Ferenc. An AST-based Code Change Representation and its Performance in Just-in-time Vulnerability Prediction. Software Technologies, pages 169–186, Cham, 2023. Springer Nature Switzerland.
  `https://link.springer.com/chapter/10.1007/978-3-031-37231-5_8`

IV. Kommit reprezentációk valós idejű sebezhetőség előrejelző képességeinek vizsgálata

A szoftverrendszerek növekvő komplexitása és összekapcsoltsága következtében egyre több szoftver sérülékenység jelenik meg, ami jelentős fenyegetést jelent. Ez a tendencia jól megfigyelhető a Tenable 2021-es jelentésében, amely a bejelentett CVE-k számottevő növekedését emeli ki [2]. Ezen folyamat megállításához, illetve lassításához elengedhetetlen a korai azonosítás a fejlesztési folyamat során. Erre próbál megoldást nyújtani a "valós idejű" (JIT) sérülékenység előrejelzés, amely a forráskód a kódbázishoz való hozzáadáskor (kommit időben) történő elemzését jelenti [16].

A kommitok értékes információkat tartalmaznak a sérülékenység elemzéshez, beleértve a hibajavításokat, a funkcionalitások hozzáadását, a kód refaktorálását és a metaadatokat. Ezen kommitok manuális elemzése azonban nehéz és könnyen elhibázható feladat, különösen nagyszabású projektekben [25]. Ezt orvosolva, a gépi tanulási megközelítések ígéretes alternatívát kínálnak az sérülékeny kommitok automatikus azonosítására [17].

Ezen megközelítések kritikus aspektusa a kommitok reprezentációja – a kommitok információinak olyan módon történő rögzítése, amely alkalmas a gépi tanulási algoritmusok számára. A reprezentációk gyakran a kommit üzenetekre [30], a kódváltozásokra (patchekre) [24], vagy a kettő kombinációjára [15, 16] támaszkodnak. Néhány megoldás kód metrikákat is beépít a kommit metaadatainak kiegészítésére [26].

A forráskód reprezentáció alapja többféle lehet, beleértve a nyers szöveget, köztes reprezentációkat (pl. AST-k), vagy kódmetrikák. A részletesség, granularitás szintén kulcsfontosságú –

befolyásolja a reprezentáció használhatóságát – ami terjedhet a teljes kommittól egészen az sor szintig. Ezen tézispont célja, hogy egy összehasonlító tanulmányt keretében ezeket a tényezőket vizsgálja a kommitok reprezentációjában, három különböző megközelítés összehasonlításával: CC2Vec, DeepJIT, és egy Kódváltozási Fa (CCT, Code Change Tree) alapú reprezentáció:

- DeepJIT [16]: A kommitok változásait elemzi a kommit üzenetek és a hozzájuk tartozó forráskód konvolúciós neurális hálózatokkal (CNN-ekkel) történő feldolgozásával. A nyers szöveges adatokat tömbösíti, és két dedikált CNN-t használ a fontos jellemzők kinyerésére – egyet a kommitok üzeneteihez és egyet a kódváltoztatásokhoz. A kapott vektorokat összesíti, hogy végleges reprezentációt hozzon létre a kommitra vonatkozóan.

- CC2Vec [15]: Vektor reprezentációkat tanul a patcheken belüli kódváltoztatásokról. Hierarchikus attention hálózatot (HAN) használ az adott patchen belül eltávolított és hozzáadott kód vektor reprezentációinak felépítésére. Végül, összehasonlító függvények segítségével az eltávolított és a hozzáadott kód közötti kapcsolatot reprezentáló jellemzőket hoz létre, amelyeket aztán összefűzünk, így kialakítva a patchen belüli kódváltoztatások végső reprezentációját.

- Code Change Tree (CCT) [4]: Az disszertáció III. pontjában bevezetett új struktúra, amely a forráskód két állapota közötti strukturális különbségeket ábrázolja Absztrakt Szintaxis Fák (AST-k) felhasználásával.

Az összehasonlító elemzés során két adathalmazt használtunk. Az első a Projekt-KB-ból származó adathalmaz [27], amely CVE azonosítókkal társított sérülékenységi bejegyzéseket tartalmaz, ahogyan azt a III. pontban bemutattuk. A második, Defectors adatbázis csak kisebb finomításokat igényelt. Mindkét adathalmaz hasonló felépítésű – kommit SHA, repository azonosító, fájlútvonal és sérülékenység jellemzőkkel – a kommit elemzéshez szükséges adatokat elérhetővé téve. A modell teljesítményének értékeléséhez ezeken a adatkészleteken olyan metrikákat használtunk, amelyek kiegyensúlyozatlan adatok esetén relevánsak maradnak, ideértve a pontosságot, a precizitást, a recall-t és az F-értékeket (F_1,F_2,$F_{0.5}$).

A modellek architektúrális különbségei miatt specifikus előfeldolgozásra volt szükség, hogy a bemeneteket megfelelően alakítsuk. Ahol lehetséges volt, a szerzők által készített implementációkat használtuk a DeepJIT-hez [1] és a CC2Vec-hez [2], míg a CCT modell esetében a következő lépéseket hajtottuk végre:

- Project-KB adathalmaz: A módosított függvényeket kigyűjtöttük a metódus szintű analízishez, majd a kialakított CCT-ket kilapítottuk és Doc2Vec-kel beágyaztuk. Egy random forest modellt tanítottunk és kiértékeltünk ezeken a vektorokon, hogy meghatározzuk a sérülékenységeket metódus szinten. Egy kommitot sérülékenynek címkéztünk, ha bármely benne lévő módosított metódust sérülékenynek találtunk.

- Defectors adathalmaz: Az adathalmaz alapvetően fájl szinten tárolja a sérülékenységeket, ezért ezt a magasabb szintű szemcsézetességet választottuk, illetve az előző ponthoz hasonló beágyazást alkalmaztunk. Ezután egy kétirányú LSTM réteget használtunk, követve egy teljesen összekapcsolt réteggel és egy szigmoid aktivációs függvénnyel, melynek kimenete értelmezhető a sérülékenységekhez tartozó valószínűségként.

[1] https://github.com/soarsmu/DeepJIT
[2] https://github.com/CC2Vec/CC2Vec

4. táblázat. A különböző metrikákkal mért teljesítmény modellenként, átlagolva a ProjectKB esetében egy 10-fold keresztvalidációs folyamat részeként, a Defectors adathalmaz esetében pedig egy tesztkészleten értékelve.

Adathalmaz	Model	Accuracy	F_1	$F_{0.5}$	F_2	Precision	Recall
ProjectKB	DeepJIT	0.74	0.47	0.41	0.56	0.38	0.64
	CC2Vec	0.59	0.37	0.3	0.51	0.32	0.64
	CCT + RF	0.7	0.33	0.3	0.37	0.29	0.4
	Baseline	0.65	0.22	0.22	0.22	0.22	0.22
Defectors	DeepJIT	0.71	0.35	0.27	0.47	0.24	0.63
	CC2Vec	0.75	0.39	0.31	0.50	0.28	0.63
	CCT + LSTM	0.66	0.3	0.23	0.42	0.20	0.58
	Baseline	0.78	0.13	0.13	0.13	0.13	0.13

Kísérleti eredményeink a 4. táblázaton láthatók, amelyek azt jelzik, hogy a kommit reprezentációk hasznosak lehetnek a valós idejű (JIT) sérülékenység előrejelzésben, de hatékonyságuk az adott felhasználási esettől függően változik. Míg az összes vizsgált modell (DeepJIT, CC2Vec és CCT) felülmúlja az egyszerű osztályozót, a valódi hatás függ a fals pozitív és a fals negatív eredmények relatív fontosságától, amelyet különböző F-értékekkel mérünk: az F_2 pontszám jó indikátor olyan esetekben, amikor a fals negatívok költségesek, és hasonlóan az $F_{0.5}$ jó indikátor, amikor a fals pozitív eredményekre fókuszálunk. A 4. táblázat megfelelő $F_{0.5}$ ésF_2 értékei alapján, azon esetekben, amikor a prioritás a fals negatívok minimalizálása (magas recall arány), ezek a reprezentációk igen igéretesek lehetnek. Azonban azokban a felhasználási esetekben, ahol a fals pozitív eredmények minimalizálása kulcsfontosságú (magas precision), a teljesítményük kevésbé lenyűgöző.

A CCT alapú modell, bár összességében valamivel alacsonyabb teljesítményt nyújt, kiemelkedik amikor lokalizáltabb sérülékenység előrejelzésre van szükség. Testreszabható részletessége miatt rugalmasabb és lehetőséget ad a sérülékeny függvények azonosítására. Ez értékessé teszi olyan esetekben, ahol a precision metrika kiemelt fontosságú, és finomabb szemcsézetű sérülékenység elemzésre van szükség.

A szerző hozzájárulása

A szerző elvégezte az kötődő irodalmak feltérképezését és kiválasztotta a kandidátus modelleket. Összegyűjtötte és szűrte a kiértékeléshez használt jelölt adatkészleteket. A módszertant a szerző közreműködésével tervezték meg. Implementálta a Code Change Tree modellt mind a Defectors, mind a Project-KB adatkészletekre. A független modellek (DeepJIT és CC2Vec) környezetét felállította, a kiértékelést futtatta. A szerző jelentős mértékben részt vett az eredmények értelmezésében és bemutatásában.

- **Tamás Aladics**, Péter Hegedűs, and Rudolf Ferenc. A Comparative Study of Commit Representations for JIT Vulnerability Prediction. Computers 13, no. 1: 22. `https://doi.org/10.3390/computers13010022`

Összefoglalás

Ebben a disszertációban négy, a szoftverminőség és -biztonság területéhez kapcsolódó gépi tanulással segített megközelítést mutatunk be. A tárgyalt témák közé tartozik egy Java forráskód

beágyazási módszer, egy új sérülékenységet bevezető adathalmazok generálási módszerének leírása, a forráskód változtatások beágyazásának megvitatása, illetve egy összehasonlító tanulmány, amely betekintést nyújt a valós idejű sérülékenység előrejelzés jelenlegi állapotába.

Először is, egy forráskód beágyazási algoritmust terveztünk, amelynek célja a forráskód strukturális információinak tárolása az Absztrakt Szintaxis Fa (AST) felhasználásával. Egy adott forráskód elemhez mélységi fa bejárással kilapítottuk a vonatkozó AST-t, és betanítottunk egy Doc2Vec beágyazási modellt, amelyet felhasználva fix hosszúságú vektorokat generáltunk. A megközelítés hatékonyságának validálásához összehasonlítottunk több gépi tanulási modellt egy hibás és hiba mentes forráskód elemekből álló adathalmazon.

Ezután, ahogy mélyebben beleástuk magunkat a szoftverhiba előrejelzés témájába, realizáltuk a sérülékenység bevezető adathalmazok hiányát – olyan adathalmazokét, amelyek bejegyzései sérülékenység kiváltó kommitok. A probléma orvoslására egy új módszert javasoltunk sérülékenység bevezető kommitok előállítására sérülékenységjavító adathalmazokból egy kétfázisú módszer alkalmazásával. Az első fázisban számos kandidátus kommitot generálunk, a második fázisban pedig egy általunk definiált relevancia-pontszám alapú szűrést végzünk a fals pozitív találatok csökkentése érdekében. Ezzel a módszerrel létrehoztunk, majd nyilvánosan közzétettünk egy új adathalmazt, amelyet a további munkáinkban is használtunk.

A *kommit reprezentáció* témakörnél célunk volt, hogy egy olyan megközelítést biztosítsunk, amely a forráskód két állapota közötti különbségeket ragadja meg, a strukturális információkra összpontosítva. E célból megalkottuk a Forráskód Változási Fát (Code Change Tree), egy olyan speciális struktúrát, amely csak a két AST közötti változásokat őrzi meg. Ezt a megközelítést függvény szinten alkalmazzuk a forráskód elemekre, és összehasonlítjuk egy metrikákon alapuló és egy egyszerű, nyers AST lapításon alapuló kódváltoztatási megközelítéssel. Előbbi statikus elemzőeszközökkel számított forráskód metrikákon alapul, utóbbi pedig a kommit előtti és utáni állapot teljes AST-jén – szemben a CCT alapú módszerrel, amelyek csak a változásokat tartalmazzák. Ehhez az összehasonlításhoz a korábbi munkánkban generált adathalmazt használtuk, és arra a következtetésre jutottunk, hogy a CCT alapú megközelítések hatékony és jól testreszabható megközelítést nyújtanak.

Köszönetnyilvánítás

Bár ez a disszertáció a saját hozzájárulásaimat emeli ki, ez a munka sosem lett volna lehetséges több személy támogatása és útmutatása nélkül.

Szeretném kifejezni hálámat témavezetőmnek, Dr. Ferenc Rudolfnak. Ő vezetett be a kutatás világába, és nagyszerű lehetőségeket biztosított a tanulásra és a fejlődésre a tanulmányaim során.

Hálás vagyok Dr. Hegedűs Péternek is a szoros együttműködésért és az állandó segítőkészségéért minden felmerülő kérdésemben. Meglátásai és támogatása kulcsfontosságú volt a fejlődésemben és a publikációimban.

Szeretném megköszönni Dr. Jász Juditnak a forráskód beágyazásokkal és egyébb területekkel kapcsolatos útmutatását, különösen az első publikációm során. Szakértelme sokat segített jobban megérteni ezt a területet.

Végül köszönöm Viszkok Tamásnak a produktív és élvezetes közös munkát számos projekten, de különösen a Deep Water Framework fejlesztésében.

Az utóbbi években nyilvánalóvá vált, hogy a kutatás kollaboratív erőfeszítés, és hálás vagyok mindenkinek, akivel lehetőségem volt együtt dolgozni.

A disszertáció az Európai Unió RRF-2.3.1-21-2022-00004 azonosítójú Mesterséges Intelligen-

cia Nemzeti Laboratórium projekt, valamint a TKP2021-NVA-09 számú a Kulturális és Innovációs Minisztérium Nemzeti Kutatási Fejlesztési és Innovációs Alapból nyújtott, a TKP2021-NVA pályázati program keretében finanszírozott projekt támogatásával valósult meg.

Aladics Tamás, 2024

Hivatkozások

[1] The mitre corporation - common vulnerabilities and exposures: https://www.cve.org/, Sep 2021. nov. 20.

[2] The 2021 Threat Landscape Retrospective: Targeting the Vulnerabilities that Matter Most, 1 2022.

[3] Tamás Aladics, Péter Hegedűs, and Rudolf Ferenc. An ast-based code change representation and its performance in just-in-time vulnerability prediction. In Hans-Georg Fill, Marten van Sinderen, and Leszek A. Maciaszek, editors, *Software Technologies*, pages 169–186, Cham, 2023. Springer Nature Switzerland.

[4] Tamás Aladics., Péter Hegedűs., and Rudolf Ferenc. A vulnerability introducing commit dataset for java: An improved szz based approach. In *Proceedings of the 17th International Conference on Software Technologies - ICSOFT*, pages 68–78. INSTICC, SciTePress, 2022.

[5] Tamás Aladics, Péter Hegedűs, and Rudolf Ferenc. A comparative study of commit representations for jit vulnerability prediction. *Computers*, 13(1), 2024.

[6] Tamás Aladics, Judit Jász, and Rudolf Ferenc. Bug prediction using source code embedding based on doc2vec. In *Proceedings of the 21th International Conference on Computational Science and Its Applications (ICCSA 2021)*, pages 382–397, Cagliari, Italy, September 2021. Springer International Publishing.

[7] Amr Amin, Amgad Eldessouki, Menna Tullah Magdy, Nouran Abdeen, Hanan Hindy, and Islam Hegazy. Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach. *Information*, 10(10), 2019.

[8] OpenStaticAnalyzer Static Code Analyzer. `https://github.com/sed-inf-u-szeged/OpenStaticAnalyzer`, 2021.

[9] Nuno Antunes and Marco Vieira. Benchmarking vulnerability detection tools for web services. In *2010 IEEE International Conference on Web Services*, pages 203–210, 2010.

[10] Harold Booth, Doug Rike, and Gregory Witte. The national vulnerability database (nvd): Overview, 2013-12-18 2013.

[11] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. Szz unleashed: an open implementation of the szz algorithm - featuring example usage in a study of just-in-time bug prediction for the jenkins project. *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation - MaLTeSQuE 2019*, 2019.

[12] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei, and Bin Li. Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection. *Information and Software Technology*, 136:106576, 2021.

[13] Rudolf Ferenc, Zoltán Tóth, Gergely Ladányi, István Siket, and Tibor Gyimóthy. A public unified bug dataset for java and its assessment regarding metrics and bug prediction. *Software Quality Journal*, 28:1447–1506, 2020. Open Access.

[14] Rudolf Ferenc, Tamás Viszkok, Tamás Aladics, Judit Jász, and Péter Hegedűs. Deep-water framework: The swiss army knife of humans working with machine learning models. *SoftwareX*, 12:100551, 2020.

[15] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. Cc2vec: distributed representations of code changes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 518–529, New York, NY, USA, 2020. Association for Computing Machinery.

[16] Thong Hoang, Hoa Khanh Dam, Yasutaka Kamei, David Lo, and Naoyasu Ubayashi. Deepjit: An end-to-end deep learning framework for just-in-time defect prediction. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 34–45, 2019.

[17] Kevin Hogan, Noel Warford, Robert Morrison, David Miller, Sean Malone, and James Purtilo. The challenges of labeling vulnerability-contributing commits. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 270–275, 2019.

[18] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *2013 35th International Conference on Software Engineering (ICSE)*, pages 672–681, 2013.

[19] Patricia Johnson. The state of open source vulnerabilities 2021 - whitesource, Jun 2022.

[20] Mina Khan, Kathryn Wantlin, Zeel Patel, Elena Glassman, and Pattie Maess. Changing computer-usage behaviors: What users want, use, and experience. In *Asian CHI Symposium 2021*. ACM, may 2021.

[21] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1188–II–1196. JMLR.org, 2014.

[22] Hongzhe Li, Taebeom Kim, Munkhbayar Bat-Erdene, and Heejo Lee. Software vulnerability detection using backward trace analysis and symbolic execution. In *2013 International Conference on Availability, Reliability and Security*, pages 446–454, 2013.

[23] Francesco Lomio, Emanuele Iannone, Andrea De Lucia, Fabio Palomba, and Valentina Lenarduzzi. Just-in-time software vulnerability detection: Are we there yet? *Journal of Systems and Software*, 188:111283, 2022.

[24] Rocío Cabrera Lozoya, Arnaud Baumann, Antonino Sabetta, and Michele Bezzi. Commit2vec: Learning distributed representations of code changes. *SN Computer Science*, 2(3), mar 2021.

[25] Patrick Morrison, Kim Herzig, Brendan Murphy, and Laurie Williams. Challenges with applying vulnerability prediction models. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, HotSoS '15, New York, NY, USA, 2015. Association for Computing Machinery.

[26] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 426–437, New York, NY, USA, 2015. Association for Computing Machinery.

[27] Serena E. Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and C´edric Dangremont. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *Proceedings of the 16th International Conference on Mining Software Repositories*, May 2019.

[28] Hossein Rahimpour, Joe Tusek, Alsharif Abuadbba, Aruna Seneviratne, Toan Phung, Ahmed Musleh, and Boyu Liu. Cybersecurity challenges of power transformers, 2023.

[29] Sarah Sharifi. A novel approach to the behavioral aspects of cybersecurity, 2023.

[30] Yaqin Zhou and Asankhaya Sharma. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 914–919, New York, NY, USA, 2017. Association for Computing Machinery.