Programok kompatibilitásának és biztonságának növelése kódelemzés és gépi tanulási technikák segítségével

Antal Gábor

Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged, 2021

Témavezető:

Dr. Ferenc Rudolf

Ph.D. értekezés tézisei

Szegedi Tudományegyetem
Informatika Doktori Iskola

Bevezetés

Manapság gyakorlatilag mindent szoftverek működtetnek. Mindenki használ szoftvereket, akár a napi munkája során, akár szabadidőben. Emiatt egyre több szoftver készül.

Az új koronavírus kitörése után a napi kibertámadások száma 300%-kal nőtt, naponta több, mint 4000 támadást okozva [5]. A kiberbűnözők könyörtelenül célba vesznek mindenkit. Remek példa erre a Zoomban található sérülékenység kihasználása (Zoombombing). A sérülékenység lehetővé tette a támadók számára, hogy csatlakozzanak tetszőleges konferenciahívásba, ami a Zoom videókonferencia szoftverben zajlott. Ez a példa is azt mutatja, hogy figyelnünk kell a szoftvereink minőségére és biztonságára. Mivel emberek vagyunk, időnként hibákat véthetünk a kódunkban, amelyek sokáig észrevétlenek maradhatnak. Ideális esetben elegendő időnk van arra, hogy a legjobbat hozzuk ki magunkból, és szabadon választhatjuk meg az eszközkészletünket annak érdekében, hogy minőségi szoftvereket készítsünk. A valóságban azonban ez aligha történik így.

A megrendelő különböző megszorításokat tehet, például a programozási nyelv pontos változatát, a rendelkezésre álló könyvtárakat és eszközöket is megkötheti. A fejlesztők azonban szeretnek új dolgokat tanulni, és naprakészek szeretnének maradni a legújabb technológiákkal. Ez nem csak egy természetes igény, az újabb nyelvi verziók használata segíthet hatékonyabb, kifejezőbb és kevésbé hibaérzékeny kódot írni.

Annak érdekében, hogy a fejlesztők nagyobb szabadságot kapjanak az újabb nyelvi szabványok használatában, megvizsgáltuk ezt a területet, és azt állapítottuk meg, hogy nincsenek olyan eszközök, amelyek segítenék a fejlesztőket a C++ újabb szabványainak használatában oly módon, hogy a kód automatikusan kompatibilis legyen a nyelv korábbi szabványaival. Megterveztünk és elkészítettünk egy teljes megoldást erre a problémára. Az eszközünk képes átalakítani C++11 forráskódot úgy, hogy az megfeleljen a C++03 szabványnak.

Természetesen nem ez az egyetlen probléma, amellyel a fejlesztők időről időre szembesülnek.. A dinamikusan típusos nyelvek (például a JavaScript és a Python) egyre szélesebb körű használatával az elemzésük is fontosabb, mint valaha. Mivel nincs fordítási fázis, nincsenek statikus típusellenőrzések. Ráadásul ezekben a nyelvekben gyakori a reflexió használata, ami még egy ember számára is megnehezíti a forráskód megértését. Számos kódelemző eszköz a program hívási gráfjára támaszkodik. A hívási gráf az alapja számos összetettebb adatszerkezetnek (például a vezérlési folyam gráf), amelyek elengedhetetlenek a programban található problémák felderítéséhez. Éppen ezért a hívási gráfok pontossága rendkívül fontos. Mivel számos eszköz és algoritmus segítségével készíthetünk hívási gráfot, alapvető fontosságú, hogy tudjuk, melyik helyzetben melyik eszköz/algoritmus használata a legelőnyösebb. Összehasonlító vizsgálatot végeztünk a manapság legnépszerűbb statikus hívási gráfot építő algoritmusok között, melyek eredményét bemutatjuk az értekezésben.

A forráskód-metrikák használata a szoftverproblémák előrejelzésére meglehetősen kiforrott technika [9, 10, 3, 7]. Azonban a dinamikus nyelvek esetén történő előrejelzés területe viszonylag új, és ezek általában fájlszintű predikcióra képesek. Ha létezne egy olyan predikciós modell, amely függvény (vagy alacsonyabb) szinten működik, a fejlesztők használhatnák azt a kódjukban felmerülő problémák megelőzésére. Az előrejelző modellek gyakorlati alkalmazása azonban a valós helyzetekben való alkalmazásuktól, és a hamis pozitív találatok arányától is függ. Először is létrehoztunk egy függvény-szintű JavaScript sérülékenység adathalmazt, amely 12 125 JavaScript-funkció statikus elemzésének eredményeit tartalmazza, kiegészítve azzal, hogy tartalmaznak-e sérülékenységet vagy sem. Bemutattnk 8 jól ismert gépi tanulási algoritmus átfogó összehasonlítását sebezhető JavaScript funkciók előrejelzésében. Előzetes eredményeink meglehetősen jók voltak annak ellenére, hogy csak statikus forráskódmetrikákat használtunk, ezért előrejelzési

modellünket dinamikus elemzési eredményekkel bővítettük, és a sérülékenységekről az általános szoftverhibákra terjesztettük ki a kutatást. A modellünk teljesítménye már attól javult, hogy a statikus függvényhívási metrikákat kicseréltük statikus és dinamikus elemzés egyvelegéből származó megfelelőikkel.

A kódelemző eszközök és előrejelző modellek használata kiváló automatizált módszerek arra, hogy segítsenek észrevenni a kódban lévő problémákat, mielőtt a változások a szoftver éles verziójában megjelennének. Mindazonáltal az emberi tényező sem elhanyagolható. A kódban nagy valószínűséggel előfordulnak hibák, amikre nem lehetünk eléggé felkészülve, azonban a már létező biztonsági problémákat alaposan megvizsgálhatjuk. A kódbázisban időről-időre előfordulnak sérülékenységek; ezek megértése segíthet abban, hogy továbbfejleszthessük az előrejelző modelljeinket. Továbbá segíthet abban is, hogy fokozott figyelmet fordíthassunk a jellemző hibák elkerülésére az oktatási anyagokban is. Az adatbányászathoz a Software Heritage Graph Datasetet [8] használtuk, hogy segítsünk a JavaScript és Python fejlesztőknek jobban megismerni az adott nyelvre jellemző biztonsági problématípusokat és azok jellemzőit. Definiáltunk egy módszertant arra vonatkozóan is, hogy hogyan lehet adatokat bányászni erre a célra. Elkészítettünk több eszközt is, valamint bemutattuk a tipikus biztonsági problémákat és azok különböző jellemzőire vonatkozó megállapításinkat számos programozási nyelv tekintetében.

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

№	[11]	[12]	[16]	[15]	[13]	[14]
I.	◆					
II.		◆				
III.			◆	◆		
IV.					◆	◆

1. táblázat. A tézispontokhoz tartozó publikációk

I. C++11 kód átalakítása C++03-ra örökölt fordítási környezetek támogatása érdekében

Az újabb technológiák (például programozási nyelvek, környezetek, függvénykönytárak) nagyon gyorsan változnak. Azonban a különböző belső és külső megszorítások gyakran megakadályozzák, hogy a projektek (és a fejlesztők) naprakészek maradjanak a gyakori változásokkal szemben. Az újabb technológiákkal való lépéstartás miatt a szoftverek kevésbé lesznek hibaérzékenyek és javul a teljesítményük, mivel minden egyes változtatással újabb nyelvi elemek, függvények kerülnek bevezetésre. Mindezek ellenére előfordulhat, hogy a megrendelők például adott platformmal való kompatibilitást követelnek meg egy szoftvergyártótól. Ez a tézispont éppen ezzel a kérdéssel foglalkozik a C++ programozási nyelv területén. A Szegedi Tudományegyetem Szoftverfejlesztés Tanszékének egyik ipari partnere olyan szoftverfejlesztő készletet (SDK) köteles használni, amelyek csak a C++ nyelv korábbi változatát támogatják. Ők azonban szeretnék lehetővé tenni a fejlesztőik számára, hogy újabb nyelvi konstrukciókat használhassanak a kódjukban, és természetesen a fejlesztők is szívesen használnák az újabb C++ nyelvi változatok lehetőségeit.

A probléma megoldására megterveztünk és megvalósítottunk egy forráskód-transzformációs keretrendszert, amely képes automatikusan visszaalakítani a C++11 szabvány szerinti forráskódot funkcionálisan megegyező C++03 szerinti forráskódra, az LLVM és clang infrastruktúra segítségével. A keretrendszerünk segítségével a fejlesztők szabadon kihasználhatják a legújabb nyelvi funkciók egy részét úgy, hogy az átalakítás utáni (éles) rendszerváltozat csak olyan nyelvi elemeket tartalmaz, amelyekkel a kód fordítható marad bármely szabványos C++03 fordítóval. A transzformációs keretrendszer két fő részből áll: az első az inkrementalitást biztosító motor, míg a második a transzformációk végrehajtásáért felel. Az inkrementalitást biztosító motor fájlszinten figyeli a változásokat a kódban, és meghatározza, hogy a projekt mely fájljait kell átalakítani. Ezen lista alapján a transzformációs motor elvégzi a szükséges módosításokat. Figyelembe kellett vennünk, hogy a C++11 számos új nyelvi elemét nem lehet egy lépésben átalakítani (pl. lambdákba ágyazott lambda kifejezések), és néhány transzformáció függ egymástól, így ezeket több iterációban, előre meghatározott sorrendben kell végrehajtani. A keretrendszer alapvető működése:

- A keretrendszer a `compile_commands.json` fájlt várja bemenetként, ami tartalmazza a projekt fordítási egységeinek információit.
- Karbantartunk egy adatbázist, amely a forráselemek legutóbbi módosítás idejének, valamint az elemek közötti függőségek eltárolásával támogatja az inkrementalitást. Az előfeldolgozás során a transzformációs keretrendszer elemzi a fordítási egységek közötti függőségeket, és az adatbázis alapján kiválasztja azokat a fájlokat, amelyeket át kell alakítani.
- A keretrendszer végrehajtja a szükséges transzformációkat.
- Miután az összes érintett fájl átalakításra került, a keretrendszer elmenti a változásokat, és az inkrementalitást biztosító motor frissíti az adatbázist.

A keretrendszerben megvalósított transzformációk: Osztálydeklarációban történő adattag inicializálás; Fordításidejű típus következtetés (auto kulcsszó); Lambda függvények; Attribútumok; Final és override kulcsszavak; Intervallum-alapú számláló ciklus (for); Konstruktor delegálás; Típus elnevezés; *Kísérleti jelleggel megvalósított transzformációk*. Egy lambda függvény átalakítását mutatja be az 1. ábra.

```
1  std::vector<int> v(6);
2  int inc = 7;
3
4
5
6
7
8
9
10 std::for_each(
11   v.begin(),
12   v.end(),
13   [&inc](int &n) {
14     n += inc;
15   }
16 );
```

⇒

```
1  std::vector<int> v(6);
2  int inc = 7;
3  class LambdaFunctor__12_1{
4    int& inc;
5  public:
6    LambdaFunctor__12_1(
7      int& inc) : inc(inc) {}
8    void operator()(int & n){
9      n += inc;
10   }
11 };
12 std::for_each()
13   v.begin(),
14   v.end(),
15   (LambdaFunctor__12_1(inc))
16 );
```

1. ábra. Példa lambda függvény átalakítására

A transzformációs keretrendszert két szempont szerint értékeltük: a transzformált kód helyessége, és az eszköz teljesítménye (futásidő). A fejlesztés és a kiértékelés korai szakaszában egyszerű kódrészletek halmazát használtuk, amelyek tartalmaztak átalakítandó nyelvi elemeket. Később pedig olyan rendszereken futtattuk az eszközt, amelyek tartalmaznak C++11 nyelvi elemeket, és nem triviális méretűek. Négy nyílt forráskódú és két zárt forrású rendszert elemeztünk.

Annak érdekében, hogy javítsuk a keretrendszerünk alkalmazhatóságát nagy rendszereken, változatos módokon próbáltuk gyorsítani a rendszert, a teljes feldolgozási idő csökkentése érdekében: a transzformációk párhuzamosan, több szálon futnak; az inkrementális transzformációk csak a szükséges lépéseket hajtják végre, az alapján, hogy mi változott a legutóbbi transzformáció óta; azonosítjuk, hogy milyen nyelvi jellemzőket használnak az egyes fordítási egységek annak érdekében, hogy a rendszer kiküszöbölje a felesleges feldolgozást a nem kapcsolódó transzformációs körökben; a MultipleTransforms fázis egyetlen körben hajtja végre az egymástól független nyelvi elemek átalakítását.

A megvalósításunk nyílt forráskódú és szabadon elérhető a GitHubon: `https://github.com/sed-szeged/cppbackport`.

A szerző hozzájárulása

A disszertáció szerzője végezte el a kódtranszformációval kapcsolatos szakirodalom feldolgozását. A szerző részt vett a lehetséges transzformációs forgatókönyvek meghatározásában, valamint azok kiértékelésében. Az inkrementalitást és transzformációfuttatást biztosító keretrendszert is a szerző készítette. Az adatbázissémát is ő tervezte meg. A transzformációs algoritmusok megvalósításában is aktívan részt vett. A szerző részt vett a teszteléshez szükséges keretrendszer és az általa futtatott tesztek megtervezésében és megvalósításában. Megtervezte, megvalósította és tesztelte azt a módszertant, amellyel a keretrendszer pre-build lépésként integrálható a fejlesztési folyamatokba.

- **Gábor Antal**, Dávid Havas, István Siket, Árpád Beszédes, Rudolf Ferenc, and József Mihalicza. Transforming C++11 Code to C++03 to Support Legacy Compilation Environments In Proceedings of the IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM 2016), Raleigh, NC, USA. Pages 177–186, IEEE, October, 2016.

II. Statikus JavaScript hívási gráf építő algoritmusok összehasonlítása

A JavaScript népszerűsége és széles körű használata mind a kliens-, mind a szerveroldalon minden eddiginél fontosabbá teszi a JavaScript nyelvű kód elemzését. Számos kódelemző eszköz a program hívási gráf alapú reprezentációjára támaszkodik. A hívási gráf csomópontjai reprezentálják a programban található függvényeket, a közöttük lévő élek pedig azt jelentik, hogy egy adott függvény meghív egy másikat végrehajtása során. Ezen reprezentáció segítségével különböző minőségi és biztonsági problémákat észlelhetünk. A hívási gráfokat további elemzések alapjául használhatjuk, például a hívási gráf alapjain eljárások közötti vezérlési folyam gráf (ICFG) építhető. Mivel ilyen fontos adatstruktúrákról van szó, a hívási gráfok pontossága meghatározza a rájuk épülő kódelemző algoritmusok pontosságát is. Pontos hívási gráfok készítése JavaScript kódból elég nagy kihívás, mivel a nyelv eredendően dinamikus, gyengén típusos és aszinkron működésű. A program futtatását nem igénylő megközelítés hátránya az, hogy az *eval*(), *bind*() és

apply() (reflexió) nem triviális használatából adódó dinamikus hívási élek hiányozhatnak. A dinamikus elemzés néhány nyilvánvaló előnye ellenére a statikus algoritmusok is megfontolandóak a hívási gráf építéshez, mivel ezek nem igényelnek magas tesztlefedettséget a programokhoz; sem pedig azok költséges végrehajtását és nyomon követését. Ebben a tézispontban öt széles körben használt statikus algoritmus által épített hívási gráfokat hasonlítottunk össze szisztematikusan, melyeket az npm callgraph, az IBM WALA, a Google Closure Compiler, az Approximate Call Graph (ACG) és a Type Analyzer for JavaScript tools (TAJS) eszköz implementál. A vizsgálatot 26 WebKit SunSpider benchmark programon és 6 valós Node.js modulon végeztük el annak érdekében, hogy jobban megérthessük ezen algoritmusok tulajdonságait.

Néhány eszközt módosítanunk kellett, főként azért, hogy kinyerhessük a programok memóriájába épített hívási gráfokat. Az eszközök kimeneteit összegyűjtöttük, és egységes JSON formátumra konvertáltuk. A gráf összehasonlító programunk segítségével egy azonos szerkezetű, egyesített JSON-t hoztunk létre. Ez a JSON tartalmazza az összes csomópontot és élet, amit legalább egy eszköz megtalált, amihez hozzáadtunk egy új attribútumot, amely felsorolja, hogy az adott csomópontot/élt mely eszközök találták meg. Elvégeztük az elemzést, és statisztikákat számítottunk az egyedi és egyesített JSON fájlokon. Az eszközök mélyreható összehasonlításához három tesztbemeneti csoportot határoztunk meg: valós, egyfájlos példák (a WebKit böngészőmotor SunSpider benchmarkja [2]); valós, többfájlos Node.js modulok (6 valós, széles körben használt Node.js modul, amely újabb nyelvi funkciókat is használ); és generált nagyméretű példák (különböző számú függvényt és azok közötti hívásokat tartalmaznak).

A kvalitatív elemzéshez létrehoztunk egy – Lhoták és társai [6] munkája által inspirált – gráf összehasonlító programot Pythonban. Az eredmények összehasonlítása mellett manuálisan kiértékeltük az eszközök által a 26 SunSpider benchmark programon talált 348 hívási élt. Ami a Node.js modulokat illeti, az élek mennyisége miatt lehetetlen volt mindet kézzel ellenőrizni. Így statisztikailag szignifikáns, reprezentatív véletlenszerű mintát választottunk az élekből, 95%-os megbízhatósággal, 5%-os hibahatárral.

Az eszközök által talált 348 hívásélből 257 valós él volt. Összesen 93 közös élt talált az öt eszköz, melyek mindegyike valós hívási él volt. Négy eszköz azonban olyan éleket is talált, melyeket a többi nem vett észre. Mivel kiértékeltük mind a 348 talált hívási élt, így ki tudjuk számítani az egyes eszközök és tetszőleges kombinációik pontossági és fedési értékeit is.

2. táblázat. Az eszközök pontossága és fedése

Eszköz(ök)	**TP**	**Összes**	**TP***	**Prec.**	**Rec.***	**F**
npm-cg	174	192	257	91%	68%	77%
ACG	233	235	257	99%	91%	95%
WALA	127	146	257	87%	49%	63%
Closure	230	284	257	81%	89%	85%
TAJS	182	186	257	98%	71%	82%

A 2. táblázat az egyes eszközök statisztikáit tartalmazza, míg a 3. táblázatban az eszközök legjobb kombinációit láthatjuk, F-mérték szerint csökkenő rendezésben. Az első oszlopban találhatjuk az eszköz vagy eszközkombinációk nevét. A második oszlop (TP) az eszköz(ök) által talált valós élek számát mutatja. A harmadik oszlopban (Összes) az eszköz(ök) által talált élek számát láthatjuk. A negyedik oszlop (TP*) a valós élek számát mutatja, melyet a kézi kiértékelés során ellenőriztünk. Az ötödik (Prec.), hatodik (Rec.*) és hetedik (F) oszlopok tartalmazzák a pontosság (TP / All), a fedés (TP* / TP) és az F-mérték értékét.

Az eszközöket egyesével megfigyelve az ACG kiemelkedő: szinte tökéletes pontossággal és meglehetősen magas fedéssel rendelkezik. Bár a TAJS és az npm-cg hasonlóan nagy pontossággal dolgozik, a fedésük messze elmarad az ACG fedésétől. A Closure fedés értéke nagyon közel van az ACG értékéhez, azonban a pontossága lényegesen alacsonyabb. A WALA elfogadható pontossággal, de a legrosszabb fedéssel rendelkezik a benchmark tesztünk alapján. Az eszközök kombinációját az F-mérték alapján vizsgálva az ACG+TAJS kombinációja kiemelkedik; együttesen szinte tökéletesen teljesítenek (98% pontosság és 99% fedés). Nincs olyan három, négy vagy öt eszközből álló kombináció, amely akár csak megközelítené ezt az F-értéket. Az összes eszközt figyelembe véve a kombinált pontosságuk 74%-ra csökken, tökéletes fedés mellett.

3. táblázat. A legjobb pontosság és fedés értékek az eszközök kombinációjával

Eszköz(ök)	**TP**	**Összes**	**TP***	**Prec.**	**Rec.***	**F**
ACG+TAJS	254	260	257	98%	99%	98%
npm-cg+ACG+TAJS	255	279	257	91%	99%	95%
ACG+WALA+TAJS	254	279	257	91%	99%	95%
ACG+WALA	241	262	257	92%	94%	93%
npm-cg+ACG	239	259	257	92%	93%	93%
npm-cg+WALA+TAJS	238	258	257	92%	93%	92%
npm-cg+ACG+WALA+TAJS	255	298	257	86%	99%	92%
npm-cg+TAJS	233	255	257	91%	91%	91%
ACG+Closure	255	309	257	83%	99%	90%
npm-cg+ACG+WALA	242	281	257	86%	94%	90%
ACG+Closure+TAJS	257	311	257	83%	100%	90%

Ahogy már említettük, csak az ACG és Closure volt képes elemezni a modern, Node.js alapú modulokat. A két eszköz által a hat modulban összesen megtalált 2281 élből 1304 él közös, ami majdnem 60%. Ez egy elég magas arány, figyelembe véve a Node.js modulok összetettségét, amely az olyan struktúrákból ered, mint az esemény callbackek, modul exportok, require utasítások, stb. 336 élt (14,7%) csak az ACG talált meg, míg 641 élt (28,1%) csak a Closure.

Mindegyik eszköznek vannak erősségei és gyengeségei is. A célunk nem egy győztes hirdetése volt, hanem az, hogy empirikus betekintést nyerjünk a manapság elterjedt statikus hívásigráfépítő algoritmusok képességeibe és hatékonyságába.

A szerző hozzájárulása

A szerző kutatta fel a lehetséges hívási gráf készítő eszközöket. A szerző aktívan részt vett a kutatás módszertanának megtervezésében. Az eszközök módosítását is a szerző végezte el a hívási gráfok kinyerésének érdekében. Ő készítette el az eszközök kimenetét egységes formátumra konvertáló eszközt is. A Node.js modulok kiválasztása és az eszközök stressztesztéléséhez szükséges mesterségesen létrehozott nagy példák elkészítése is a szerző munkája volt. A disszertáció szerzője részt vett az eredmények kiértékelésében is, valamint azok manuális validációjában is. A szerző dolgozta ki a performanciamérés módszertanát, valamint annak elvégzése is a szerző saját munkája.

- **Gábor Antal**, Péter Hegedűs, Zoltán Tóth, Rudolf Ferenc, and Tibor Gyimóthy Static JavaScript Call Graphs: A Comparative Study. In Proceedings of the 2018 IEEE 18th

International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177–186, Sep. 2018

– **Distinguished Research Paper Award**

III. Statikus és dinamikus kódelemzés ötvözése gépi tanulással a JavaScript programok hibáinak felderítésére

A hibaelőrejelzés célja, hogy megtalálja a szoftverrendszer azon forráskód-részleteit, amelyek valószínűleg hibákat tartalmaznak. A program leginkább hibaérzékeny részeinek ismeretében hatékonyan lehet elosztani a korlátozott tesztelési és kódellenőrzési (review) erőforrásokat. Ezért mind a sérülékenység-, mind a hibaelőrejelzés nagymértékben támogathatja a szoftverek karbantartását és fejlesztését.

Jelen tézispontban két előrejelzési modellt mutatunk be, amelyek különböző adathalmazokat és különböző jellemzőket használnak a szoftverproblémák előrejelzésére. Megvizsgáltuk, hogy a függvények hibáinak előrejelzése megvalósítható-e különböző szoftvermetrikák alapján. Összehasonlítottuk a legelterjedtebb gépi tanulási algoritmusok teljesítményét hibaelőrejelzési feladatokon. Az algoritmusok között szerepel kétfajta mély neurális hálózat (DNN_s, DNN_c), a k legközelebbi szomszéd algoritmus (KNN), egy döntési fa osztályozó (Tree), a tartóvektor gépek C-tartóvektor osztályozó változatát (SVM), a véletlen erdő (Forest), a logisztikus regresszió (Logistic), a lineáris regresszió (Linear) és a Gauss Naive Bayes algoritmust (Bayes). Az adathalmaz kiegyensúlyozatlanságának kezelésére különböző újramintázási stratégiákat is alkalmaztunk.

Először is megvizsgáltuk, hogy a gépi tanulási technikák hogyan teljesítenek lehetséges biztonsági réseket tartalmazó JavaScript függvények előrejelzésében. Legjobb tudomásunk szerint nem létezett kifejezetten JavaScript nyelvű programokra vonatkozó sérülékenységi adathalmaz, ezért létrehoztunk egy függvény-részletességű, nyilvános JavaScript sérülékenységi adathalmazt több sérülékenységi adatbázisból származó adatokkal, amelyeket automatikusan összevontunk a GitHubon elérhető információkkal (pl.: a javítást tartalmazó commitok és patchek). Az új függvény-szintű sérülékenység adathalmaz 12 125 funkciót tartalmaz, melyek közül 1496 tartalmaz sérülékenységeket.

4. táblázat. A gépi tanulási algoritmusok által elért F-mértékek

Alg.	**Nincs**	**↑25%**	**↑50%**	**↑75%**	**↑100%**	**↓25%**	**↓50%**	**↓75%**	**↓100%**	**Rand**
DNN_s	0,71	**0,71***	0,71	0,65	0,68	0,70	0,71	0,69	0,59	0,05
DNN_c	0,71	0,70	0,71	0,68	0,65	**0,71***	0,71	0,68	**0,66**	0,01
Forest	0,71	**0,74***	**0,74**	**0,73**	**0,72**	0,72	0,72	0,72	0,65	0,05
KNN	**0,76***	**0,75**	0,72	0,6935	0,6817	**0,76**	**0,75**	**0,74**	0,64	0,14
Linear	0,26	0,48	**0,55***	0,49	0,45	0,30	0,37	0,51	0,44	0,02
Logistic	0,33	0,50	**0,57***	0,55	0,49	0,38	0,45	0,53	0,49	0,01
SVM	0,67	0,70	**0,72***	0,70	0,68	0,67	0,67	0,67	0,65	0,16
Tree	**0,72***	0,71	0,71	0,71	0,70	0,70	0,69	0,67	0,59	0,15
Bayes	0,15	0,16	0,16	**0,21***	0,20	0,16	0,16	0,18	0,17	0,07
Medián	0,71	0,70	**0,71***	0,68	0,68	0,70	0,69	0,67	0,59	0,05

Statikus forráskód metrikákat használtunk prediktorként, és egy kiterjedt rácskereső (grid search) algoritmust, hogy megtaláljuk a legjobban teljesítő modelleket. Az elért eredmények

meglepően jók, tekintve, hogy a JavaScript egy rendkívül dinamikus nyelv, mi pedig csak statikus forráskód metrikákat használtunk prediktorként. A 9 modellből 5 modell ért el 0,70 feletti F-mértéket és az SVM is nagyon közel volt hozzá, 0,67-es F-mértékkel. Érdekes észrevétel, hogy az összes algoritmus esetében a pontosság jelentősen magasabb volt, mint a fedés, kivéve a döntési fa osztályozót, melynek pontossága 0,74, fedése 0,7 így F-mértéke 0,72 volt. Összehasonlításképp létrehoztunk egy egyszerű algoritmust (ami minden beérkező példányt sebezhetőnek jósol) 0,12-es pontosságot és 1-es fedést ért el, ami 0,21-es F-mértéket jelent. A modellek eredményeit a 4. táblázat foglalja össze. A legjobban teljesítő algoritmus a KNN volt, $0,76$-os F-mértékkel. Emellett a mélytanulás, a fa- és erdőalapú osztályozók, valamint az SVM is versenyképes volt, $0,70$ feletti F-mértékkel.

Megvalósítottunk egy másik függvény-szintű JavaScript hibaelőrejelző modellt is, amely statikus forráskód metrikákon alapul, azonban kiegészítettük hibrid (statikus és dinamikus) kódelemzésen alapuló metrikákkal, amelyek a bejövő és kimenő függvényhívások (HNII és HNOI) számát tárolják.

A motivációnk az volt, hogy a JavaScript egy rendkívül dinamikus szkriptnyelv, amely esetében a statikus kódelemzési módszerek nagyon pontatlanok lehetnek (ahogy azt már láttuk a hívási gráfok esetében is), ezért a pusztán statikus forráskódjellemzők használata egy előrejelzési feladathoz nem biztos, hogy elegendő.

Az ESLint JavaScript projekt 824 hibás és 1943 nem hibás függvényét gyűjtöttük ki a nyilvánosan elérhető BugsJS adatbázisból [4]. Létrehoztunk egy hibrid hívásigráf-elemző keretrendszert, amely a projekt forráskódját várja bemenetként. Ezután a forráskódot különböző statikus és dinamikus eszközökkel elemeztük (amihez szükség lehet a forráskód futtatására). Az elemzéseket követően a keretrendszer az összes eszköz kimenetét egységes JSON formátumba konvertálja, majd egyesíti. Az egységesített JSON-t használtuk a hibrid hívási metrikák (azaz a HNII és a HNOI) kiszámításához. A hibrid metrikák kiszámítása mellett az OpenStaticAnalyzer [1] statikus forráskódelemző program által biztosított statikus metrikákat is használtuk.

Eredményeink alapján megerősíthetjük a hibrid kódmetrikák pozitív hatását a gépi tanulási modellek előrejelzési teljesítményére.

5. táblázat. A 9 gépi tanulási modell legjobb eredményei F-mértékük szerint csökkenő sorrendben

Algoritmus	**Jellemzők**	**Találati arány**	**Pontosság**	**Fedés**	**F-mérték**	**MCC**
Forest	S+H	0,816	0,753	0,569	0,648	0,54
KNN	S+H	0,788	0,646	0,635	0,641	0,49
DNN_c	S+H	0,784	0,649	0,601	0,624	0,47
Tree	S+H	0,781	0,649	0,58	0,612	0,46
DNN_s	H	0,774	0,634	0,569	0,6	0,44
Logistic	S+H	0,787	0,682	0,533	0,598	0,46
SVM	S+H	0,789	0,699	0,515	0,593	0,47
Linear	S+H	0,769	0,67	0,443	0,533	0,4
Bayes	S+H	0,772	0,713	0,394	0,508	0,4

Az 5. táblázat bemutatja a 9 gépi tanulási modell legjobb teljesítményét (minden modell a legjobban teljesítő hiperparaméter beállításokkal és jellemzőkészlettel szerepel). A gépi tanulási algoritmustól, az alkalmazott hiperparaméterektől és a figyelembe vett mérőszámtól függően a hibrid hívási metrikák 2-10%-os növekedést eredményeznek a modellek teljesítményében (pontosság, fedés, F-mérték tekintetében). Érdekes módon, a statikus NOI és NII metrikák helyettesítése

a hibrid elemzésből származó társaikkal (HNOI és HNII), önmagában javítja a modellek teljesítményét, azonban a legjobb eredményt az összes metrika együttes használata adja.

A létrehozott sérülékenység adathalmaz nyilvánosan elérhető: `http://www.inf.u-szeged.hu/~ferenc/papers/JSVulnerabilityDataSet/`, míg a megvalósított keretrendszer a GitHubon található: `https://github.com/sed-szeged/hcg-js-framework`.

A szerző hozzájárulása

A szerző részt vett a tanulmány módszertanának kialakításában. A szakterület irodalmának feldolgozása is a szerző munkája. Jelentős részt vállalt az adatgyűjtő és összevonó rendszer megtervezésében és kialakításában. A szerző részt vett a manuális kiértékelésben is. A hibrid hívásigráfelemző keretrendszer megtervezése és megvalósítása főként a szerző munkája volt. A különböző jellemzőhalmazok kialakítása a szerző munkája. A gépi tanulási modellek eredményeinek kiértékelésében a szerző aktívan részt vett.

- Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, **Gábor Antal**, Dénes Bán, and Tibor Gyimóthy Challenging machine learning algorithms in predicting vulnerable JavaScript functions. In Proceedings of the 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2019), pages. 8-14, IEEE, May 28, 2019

- **Gábor Antal**, Zoltán Tóth, Péter Hegedűs and Rudolf Ferenc. Enhanced Bug Prediction in JavaScript Programs with Hybrid Call-Graph Based Invocation Metrics. In Technologies 9, no. 1: 3, MDPI.

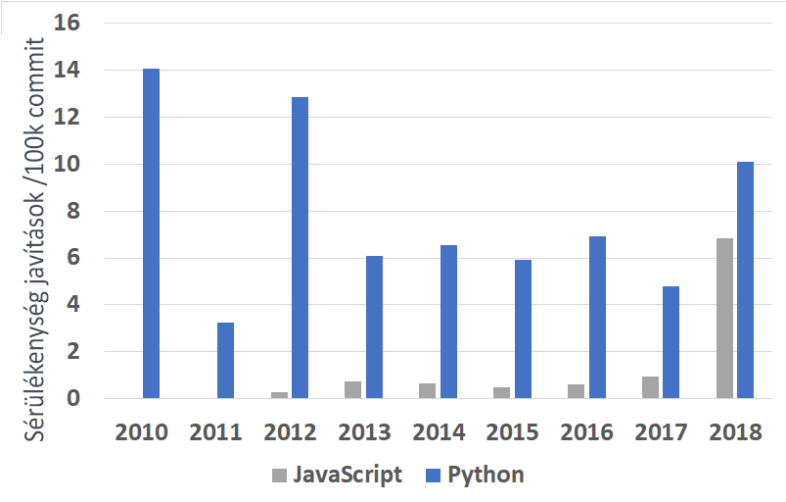
IV. Tipikus biztonsági hibák és javításuk vizsgálata nyílt forrású rendszerekben

Napjainkban a szoftverbiztonság kétségtelenül a szoftverfejlesztés egyik legnagyobb problémája. Habár a biztonsági problémákra gyakran tudatosan odafigyelnek a fejlesztők, a gyakorlati tapasztalatok azt mutatják, hogy a valóságban sem a megelőző intézkedések, sem a lehetséges problémákra való reakció nem mindig megfelelő.

A nyílt forráskódú fejlesztői közösségek nagy mennyiségű commitjának elemzésével kategorizálhatjuk a fejlesztők által javított sérülékenységeket, és megvizsgálhatjuk azok eloszlását, javítási idejét és egyéb jellemzőit, hogy megismerjük és továbbfejleszthessük a biztonságkezelési folyamatokat és gyakorlatokat. Ezenkívül a programozási nyelvek tipikus sérülékenységeinek megértése segíthet a kutatóknak a sebezhető szoftverkomponenseket előrejelző gépi tanulási modellek finomhangolásában. Ebben a tézispontban két különböző adatbázist használtunk sérülékenységi adatok bányászatához és azok elemzéséhez. A sérülékenységek azonosítására a CVE katalógust használtuk. A CVE-k (a Common Vulnerabilities and Exposures rövidítése) nyilvánosan közzétett biztonsági sérülékenységek, amelyeket online elérhetők és szabadon böngészhetők. A sérülékenységeket CWE (Common Weakness Enumeration) kategóriák szerint lehet csoportosítani, amely egy széles körben elfogadott kategorizálása a CVE azonosítóval rendelkező sérülékenységeknek.

A Software Heritage Graph Dataset segítségével megvizsgáltuk két népszerű szkriptnyelv, a Python és a JavaScript commitjait. Azonosítottuk azokat a commitokat, amelyek egy adott sérülékenységet enyhítenek a kódban (azaz a sérülékenységet kijavító commitokat). Megvizsgáltuk, hogy a JavaScript és a Python közösségek milyen gyorsan javítanak egy újonnan közzétett

biztonsági sérülékenységet. Megkülönböztettük a commit üzenetekben említett sérülékenységek típusait (melyekhez a CWE kategorizálást használtuk), és összehasonlítottuk ezek számát a két közösségen belül. Megállapítottuk, hogy a JavaScript projektek 87 különböző kategóriába tartozó biztonsági résre hivatkoznak, a Python projektek pedig 71-re, amelyek közül 55 kategória közös. Habár a sérülékenységi kategóriák között nagy volt az átfedés, a kategóriánként enyhített sérülékenységek száma jelentősen eltér a projektek nyelvétől függően. Például a Cross-Site Scripting (CWE-79), Path Traversal (CWE-22), Improper Input Validation (CWE-20) és az Uncontrolled Resource Consumption (CWE-400) típusú sérülékenységeket leginkább a JavaScript projektekben, míg az Resource Management Errors (CWE-399) és a Permissions, Privileges, and Access Controls (CWE-264) típusú sérülékenységeket leginkább a Python projektekben mitigálják. A sérülékenységet javító commitok számának növekedése mindkét nyelvben közös tendencia, de a növekedés arányos a commitok teljes számának növekedésével. Az összes commitra jutó sérülékenység javítás aránya csak lassan nő, azonban a 2018-as évben mind a JavaScript, mind a Python projektek esetében jelentős növekedés volt tapasztalható a javítások mennyiségében (ahogy látható a 2. ábrán).



2. ábra. A sérülékenységek javításának aránya évekre bontva

Míg a Python sérülékenységek javításának aránya meglehetősen stabil, a JavaScript projektek esetében ugyanez az arány 2015-től kezdve folyamatosan nő, 2018-ban mutatta a legjobb arányt, ami még mindig alacsonyabb, mint a Python projekteké. Az adott biztonsági rés közzététele és az annak mitigálását tartalmazó első commit időpontja között eltelt napok száma nagy szórást mutat. Jellemzően a Python commitok 100 napnál nem régebbi sérülékenységeket mitigálnak, míg egyes JavaScript commitok esetében akár egy évnél is régebbi sérülékenységeket javítanak.

Több eszközt is létrehoztunk, amelyek segítik az ehhez és hasonló tanulmányokhoz szükséges adatok bányászatát. A eszközök használatával bemutattuk az adatgyűjtési képességüket; több programozási nyelvhez bányásztunk (a GHTorrent szerint) népszerű GitHub repository-kat, és létrehoztuk saját adatbázisunkat, amely nyilvánosan elérhető. Célunk az volt, hogy kiderítsük, vannak-e egyező minták a legelterjedtebb programozási nyelveken belül a biztonsági problémák és javítások tekintetében.

Megállapításaink között szerepel, hogy ugyanazok a biztonsági problémák különböző nyelvekben eltérő módon jelenhetnek meg, és így a megoldások is eltérőek lehetnek. Azt is megállapítottuk, hogy a hasonló méretű projektek rendkívül eltérő eredményeket produkálhatnak, és különböző gyengeségekkel rendelkezhetnek, még akkor is, ha azonos feladat megoldására készítették őket. Ezek a statisztikák nem feltétlenül tükrözik teljes mértékben a projektek biztonsági színvonalát, de támpontot nyújthatnak ahhoz, hogy mire lehet számítani. Legtöbbször egy CVE-

bejegyzést akkor említenek, amikor úgy vélik, hogy javították, ami nem meglepő, hiszen az ember nem akarja felfedni a programjában lévő tényleges sérülékenységet, annak javítása előtt. Ebből a tényből kiindulva a legtöbb CVE-t csak egyszer kellene megemlíteni, amikor javítják őket. Azonban a legtöbb nagyméretű projekt esetében nem ez a helyzet. Feltételezzük, hogy ez azért történik, mert a későbbi változtatások újra bevezethetnek egy korábban javított sérülékenységet, ami azért valószínű, mert nagyobb rendszerekben sokkal nehezebb előre látni minden lehetséges eredményt, amit egy változtatás okozhat. A kézi vizsgálat során azt tapasztaltuk, hogy a hosszabb kódtörténettel rendelkező projektek általában több visszatérő problémával rendelkeznek, mint mások. A CVE-k súlyossága és a javításukhoz szükséges idő közötti összefüggés megmutatja, hogy a fejlesztők mennyire voltak felkészülve a sérülékenységek kijavítására. Mint tapasztaltuk, nincs erős összefüggés a súlyosság és a sérülékenység javítási ideje között, azonban apróbb összefüggések észrevehetők. Például a Python esetében a súlyosabb sérülékenységeket gyorsabban kijavították, mint a kevésbé súlyosakat. Ez arra utalhat, hogy a fejlesztők nagyobb hangsúlyt fektettek a súlyosabb problémák elhárítására.

A létrehozott eszközök és az adathalmaz elérhető a GitHubon: `https://cveminer.github.io`.

A szerző hozzájárulása

A disszertáció szerzője dolgozta ki a tanulmány alapvető koncepcióját. A Software Heritage Graph Dataset adatainak bányászata és az eredmények CVE/CWE adatokkal való összevonása is a szerző saját munkája. Ezenkívül ő fektette le a nyílt forrású adatbányászati eszközök megvalósításának alapjait. Az eszközök továbbfejlesztéséért is a szerző volt felelős. Az eredmények kiértékelése is főként a szerző munkája. Az eredmények kézi validálásában is aktívan részt vett a disszertáció szerzője.

- **Gábor Antal**, Márton Keleti, and Péter Hegedűs. Exploring the Security Awareness of the Python and JavaScript Open Source Communities. In Proceedings of the 17th International Conference on Mining Software Repositories (MSR '20). Association for Computing Machinery (ACM), New York, NY, USA, 16–20.
- **Gábor Antal**, Balázs Mosolygó, Norbert Vándor and Péter Hegedűs. A Data-Mining Based Study of Security Vulnerability Types and Their Mitigation in Different Languages. In Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2020), Published in Lecture Notes in Computer Science (LNCS), vol 12252. Springer, Cham, page 1019-1034, Cagliari, Italy, July 1-4, 2020.

Összefoglalás

Jelen disszertációban négy témát és több, mint 6 évnyi kutatómunkát mutattunk be. Az érintett területek közé tartozik a C++ örökölt fordítási környezetek támogatása úgy, hogy lehetővé tegyük a fejlesztők számára az újabb nyelvi szabványok egy részének használatát, statikus JavaScript hívásigráf-építő algoritmusok közötti különbségek vizsgálata, predikciós modellek készítése JavaScript függvények szoftverproblémáinak előrejelzésére, és végül, de nem utolsósorban, a tipikus biztonsági problématípusok tanulmányozása számos programozási nyelvben.

Először is, bemutattunk egy teljes módszert a C++ projektekben az *örökölt fordítási környezetek támogatására*, oly módon, hogy az eszközünk a C++11-ben meghatározott új nyelvi

jellemzők egy részhalmazát tartalmazó kódot olyan funkcionálisan egyenértékű kóddá alakítja át, amely tetszőleges szabványos C++03 fordítóval lefordítható. Számos tesztet is készítettünk, valamint 6 valós alkalmazáson is folyamatosan teszteltük a megoldásunkat. Az eredményeink alapján kijelenthetjük, hogy a transzformációs keretrendszer képes több millió kódsort tartalmazó projektek átalakítására is.

A *statikus JavaScript hívásigráf-algoritmusok* területén elvégeztük az öt legmodernebb eszköz szisztematikus összehasonlítását, egy JavaScript benchmark és számos valós Node.js modul segítségével. Bemutattuk az eszközök közötti hasonlóságokat és különbségeket. Nem tudtunk megállapítani egy abszolút győztest, mivel minden eszköznek vannak erősségei és gyengeségei. Megmutattuk azt is, hogy a különböző eszközök kombinációja adja a legjobb eredményt.

A *szoftverproblémák előrejelzésének* témakörében 8 jól ismert gépi tanulási algoritmust hasonlítottunk össze sebezhető JavaScript funkciók előrejelzésében, amelyhez egy újonnan létrehozott adathalmazt használtunk, ami számos statikus forráskód-metrikát tartalmaz függvény szintre bontva. Mivel bíztató eredményeket kaptunk, így kiterjesztettük a kutatás hatókörét, és két hibrid hívási gráf alapú metrikával bővítettük a jellemzőhalmazt: hibrid kimenő hívások száma (HNOI), hibrid bejövő hívások száma (HNII), amelyek kiszámításához a statikus hívási élek mellett dinamikus (futásidejű) függvényhívási információkat is használtunk. Elkészítettünk egy hibrid hívási gráf keretrendszert is, hogy megkönnyítsük a hibrid elemzéssel kapcsolatos jövőbeli kutatásokat. Összehasonlítottunk 8 jól ismert gépi tanulási algoritmust a JavaScript funkciókban található szoftverhibák előrejelzésére. Kimutattuk, hogy a hibrid hívásalapú metrikák következetesen javítják a predikciós modellek performanciáját; a használt gépi tanulási algoritmustól függően 2-10% javulás mérhető a modellek teljesítményében (pontosság, fedés, F-mérték).

Végezetül pedig a *tipikus biztonsági problémák típusainak tanulmányozása* terén bemutattuk a módszerünket, hogyan lehet összegyűjteni a szükséges adatokat a Software Heritage Graph Datasetből, valamint, hogy hogyan lehet hatékonyan bányászni a szükséges adatokat bármely Git repository-ból. Számos eszközt készítettünk, amelyek segíthetik a terület kutatóit. Eredményeink azt mutatják, hogy a programozási nyelvekben léteznek tipikus sérülékenységtípusok, melyek mitigálása sokkal több időt vehet igénybe, mint gondolnánk. Szerencsére azonban az idő múlásával a sérülékenységek javítási folyamata egyre gyorsabbá válik, ami mindenki számára megnyugtató lehet.

Köszönetnyilvánítás

Habár a disszertáció a szerző hozzájárulását hangsúlyozza, a bemutatott kutatómunka eredményeit nem érhettem volna el mások segítsége nélkül. Mindenekelőtt szeretnék köszönetet mondani témavezetőmnek, Dr. Ferenc Rudolfnak az útmutatásért és hasznos tanácsaiért, amelyek a tanulmányaim során végig segítettek. A pozitív és nyugodt hozzáállása sokat segített nekem; nélküle valószínűleg sosem végeztem volna tudományos kutatómunkát. Külön köszönet illeti Dr. Hegedűs Pétert, akit második számú mentoromnak, témavezetőmnek tartok. Sok nélkülözhetetlen dolgot tanított nekem a kutatással kapcsolatban, és sok segítséget kaptam tőle az évek során. Őszinte köszönet illeti Dr. Gyimóthy Tibort, a Szoftverfejlesztés Tanszék korábbi vezetőjét, a kutatómunkám során nyújtott támogatásáért. Szeretnék köszönetet mondani Dr. Nagy Csabának és Dr. Szőke Gábornak, akik tudományos pályám kezdetén rengeteget segítettek. Hálás köszönetemet szeretném kifejezni kollégáimnak és társszerzőimnek, név szerint Dr. Tóth Zoltánnak, Dr. Siket Istvánnak, Dr. Beszédes Árpádnak, Dr. Mihalicza Józsefnek, Keleti Mártonnak, Mosolygó Balázsnak, Vándor Norbertnek, Gyimesi Péternek, és Bán Dénesnek. Szeretnék köszönetet mondani az NNG Kft-nek a C++ kódtranszformációval kapcsolatos érdekes témáért. Szeretném

megköszönni Szűcs Editnek, hogy stilisztikai és nyelvtani észrevételeivel segítette disszertációm létrejöttét.

Végül, de nem utolsósorban, szeretnék köszönetet mondani a családomnak, amiért támogató hátteret biztosítottak a tanulmányaimhoz, valamint amiért végig biztattak a kutatói pályám során.

A disszertáció eredményeinek nagy része a SETIT projekt (2018-1.2.1-NKP-2018-00004)[1] keretében készült.

A disszertációban szereplő kutatást az Innovációs és Technológiai Minisztérium és a Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal támogatta a Mesterséges Intelligencia Nemzeti Laboratórium keretében.

Antal Gábor, 2021

[1] A 2018-1.2.1-NKP-2018-00004 számú projekt a Nemzeti Kutatási és Innovációs Alapból biztosított támogatással, a "Nemzeti Kiválósági Program: 2018-1.2.1-NKP" pályázati program finanszírozásában valósult meg.

Hivatkozások

[1] OpenStaticAnalyzer - GitHub. `https://github.com/sed-inf-u-szeged/OpenStaticAnalyzer`. Accessed: 2021-04-29.

[2] Sunspider 1.0.2 benchmark. `https://github.com/WebKit/webkit/tree/master/PerformanceTests/SunSpider/tests/sunspider-1.0.2`. Accessed: 2018-10-16.

[3] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[4] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Árpád Beszédes, Rudolf Ferenc, and Ali Mesbah. BugsJS: a benchmark of javascript bugs. In *Proceedings of 12th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 90–101, 2019.

[5] Internet Crime Complaint Center (IC3) of US Federal Bureau of Investigation and United States of America. Internet Crime Report 2020. `https://www.ic3.gov`, 2020. Accessed: 2021-04-29.

[6] Ond Lhoták et al. Comparing Call Graphs. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 37–42. ACM, 2007.

[7] Jaechang Nam. Survey on software defect prediction. *Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep*, 2014.

[8] Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. The software heritage graph dataset: Public software development under one roof. In *MSR 2019: The 16th International Conference on Mining Software Repositories*, pages 138–142. IEEE, 2019.

[9] K Punitha and S Chitra. Software defect prediction using software metrics-a survey. In *2013 International Conference on Information Communication and Embedded Systems (ICICES)*, pages 555–558. IEEE, 2013.

[10] Danijel Radjenović, Marjan Heričko, Richard Torkar, and Aleš Živkovič. Software fault prediction metrics: A systematic literature review. *Information and software technology*, 55(8):1397–1418, 2013.

A szerző publikációi

[11] G. Antal, D. Havas, I. Siket, Á. Beszédes, R. Ferenc, and J. Mihalicza. Transforming c++11 code to c++03 to support legacy compilation environments. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 177–186, 2016.

[12] G. Antal, P. Hegedus, Z. Tóth, R. Ferenc, and T. Gyimóthy. Static javascript call graphs: A comparative study. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 177–186, Sep. 2018.

[13] Gábor Antal, Márton Keleti, and Péter Hegedűs. Exploring the security awareness of the python and javascript open source communities. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 16–20, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Gábor Antal, Balázs Mosolygó, Norbert Vándor, and Péter Hegedűs. A data-mining based study of security vulnerability types and their mitigation in different languages. In *International Conference on Computational Science and Its Applications*, pages 1019–1034. Springer, 2020.

[15] Gábor Antal, Zoltán Tóth, Péter Hegedűs, and Rudolf Ferenc. Enhanced bug prediction in javascript programs with hybrid call-graph based invocation metrics. *Technologies*, 9(1):3, 2021.

[16] R. Ferenc, P. Hegedűs, P. Gyimesi, G. Antal, D. Bán, and T. Gyimóthy. Challenging machine learning algorithms in predicting vulnerable javascript functions. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 8–14, 2019.

Témavezetői nyilatkozat

Alulírott, Dr. Ferenc Rudolf, mint *Antal Gábor* doktorjelölt témavezetője kijelentem, hogy a jelölt *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című értekezésében felhasznált eredmények a jelölt saját hozzájárulását tükrözik.

2021. április 28.

Dr. Ferenc Rudolf

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán and Tibor Gyimóthy. **Challenging machine learning algorithms in predicting vulnerable JavaScript functions.** In 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2019), pages. 8-14, IEEE, May 28, 2019

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt meghatározó:

- A szoftvermetrika-alapú sérülékenység-előrejelzéssel kapcsolatos szakirodalom feldolgozása [1]
- A nsp, Snyk, GitHub oldalakról adatokat gyűjtő, egységesítő alkalmazás tervezése, implementációja [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A gépi tanulási algoritmusokat futtató keretrendszer implementálása [1]

2021. április 28.

Dr. Bán Dénes László

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Balázs Mosolygó, Norbert Vándor and Péter Hegedűs. **A Data-Mining Based Study of Security Vulnerability Types and Their Mitigation in Different Languages.** In International Conference on Computational Science and Its Applications (ICCSA 2020), Springer, Cham, page 1019-1034, Cagliari, Italy, July 1-4, 2020

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- A GitHub bányászatához használt eszközök implementációja [1]
- Az eredmények előállítása, kiértékelése, statisztikák készítése [1]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- A sérülékenységvizsgálatokkal kapcsolatos szakirodalom feldolgozása [1]
- A kutatás elvi módszerének kidolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

2021. április 28.

Mosolygó Balázs József

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Péter Hegedűs, Zoltán Tóth, Rudolf Ferenc, and Tibor Gyimóthy. **Static JavaScript Call Graphs: A Comparative Study.** In 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177–186, Sep. 2018
2. Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán and Tibor Gyimóthy. **Challenging machine learning algorithms in predicting vulnerable JavaScript functions.** In 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2019), pages. 8-14, IEEE, May 28, 2019
3. Gábor Antal, Zoltán Tóth, Péter Hegedűs and Rudolf Ferenc. **Enhanced Bug Prediction in JavaScript Programs with Hybrid Call-Graph Based Invocation Metrics.** In Technologies. 2021. MDPI; 9(1):3
4. Gábor Antal, Márton Keleti, and Péter Hegedűs. **Exploring the Security Awareness of the Python and JavaScript Open Source Communities.** In Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020), page 16-20, June 29-30, 2020.
5. Gábor Antal, Balázs Mosolygó, Norbert Vándor and Péter Hegedűs. **A Data-Mining Based Study of Security Vulnerability Types and Their Mitigation in Different Languages.** In International Conference on Computational Science and Its Applications (ICCSA 2020), Springer, Cham, page 1019-1034, Cagliari, Italy, July 1-4, 2020

cikkekben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- Az algoritmusok összehasonlítási módszerének elvi kidolgozása [1]
- A hívási gráfok elkészítése a kiválasztott alkalmazások segítségével [1]
- A hívási gráfok összehasonlítása, az eredmények kiértékelése [1]
- A gépi tanulási algoritmusok eredményének kiértékelése, összehasonlítása [2, 3]
- A gépi tanuló algoritmusok összehasonlítási módszerének elvi kidolgozása [2, 3]
- Az eredmények előállítása, kiértékelése, statisztikák készítése [5]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- A hívási gráf készítő eszközök kiválasztása, módosítása, a hívási gráf kinyerésének céljából [1]

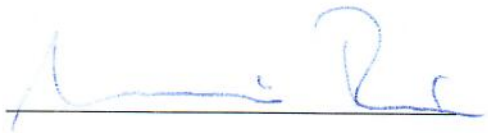
- Formátum átalakító eszköz megtervezése, kifejlesztése [1]
- Nyílt forrású rendszerek választása, mesterséges példák generálásának elvi kidolgozása, implementálása, validálása [1]
- A performanciamérés elvi kidolgozása, implementálása, a performanciamérés elvégzése, kiértékelése [1]
- A szoftvermetrika-alapú sérülékenység-előrejelzéssel kapcsolatos szakirodalom feldolgozása [2]
- A nsp, Snyk, GitHub oldalakról adatokat gyűjtő, egységesítő alkalmazás tervezése, implementációja [2]
- A metrika-alapú hibaelőrejelzéssel kapcsolatos szakirodalom feldolgozása [3]
- A hibrid metrikákat előállító keretrendszer megtervezése és implementációja [3]
- A különböző jellemzőhalmazok előállítása [3]
- Az eredmények előállítása, kiértékelése, statisztikák készítése [4]
- A sérülékenységvizsgálatokkal kapcsolatos szakirodalom feldolgozása [4, 5]
- A kutatás elvi módszerének kidolgozása [4, 5]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A hívási gráfokat merge-elő alkalmazás elkészítése [1]
- Az összegyűjtött adatok manuális validációja és tisztítása [2]
- A biztonsági sérülékénysegek vizsgálatának motivációja [4, 5]
- A projekt szakmai koordinációja [4, 5]

2021. április 28.



Dr. Hegedűs Péter

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Dávid Havas, István Siket, Árpád Beszédes, Rudolf Ferenc and József Mihalicza. **Transforming C++ 11 code to C++ 03 to support legacy compilation environments.** In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177-186. IEEE. Oct. 2016

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- Az ipari rendszereken való futás kiértékelése [1]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- Lehetséges transzformációs forgatókönyvek feltérképezése [1]
- Az inkrementális keretrendszer megtervezése és kidolgozása, az adattároláshoz szükséges séma kialakítása [1]
- Teszteléshez szükséges környezet kialakítása, fejlesztői folyamatokba való integráció kidolgozása [1]
- A kódtranszformációval kapcsolatos szakirodalom feldolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

2021. április 28.

Dr. Siket István

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Balázs Mosolygó, Norbert Vándor and Péter Hegedűs. **A Data-Mining Based Study of Security Vulnerability Types and Their Mitigation in Different Languages.** In International Conference on Computational Science and Its Applications (ICCSA 2020), Springer, Cham, page 1019-1034, Cagliari, Italy, July 1-4, 2020

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- A GitHub bányászatához használt eszközök implementációja [1]
- Az eredmények előállítása, kiértékelése, statisztikák készítése [1]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- A sérülékenységvizsgálatokkal kapcsolatos szakirodalom feldolgozása [1]
- A kutatás elvi módszerének kidolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

2021. április 28.

Vándor Norbert Rudolf

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Péter Hegedűs, Zoltán Tóth, Rudolf Ferenc, and Tibor Gyimóthy. **Static JavaScript Call Graphs: A Comparative Study.** In 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177–186, Sep. 2018
2. Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán and Tibor Gyimóthy. **Challenging machine learning algorithms in predicting vulnerable JavaScript functions.** In 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2019), pages. 8-14, IEEE, May 28, 2019

cikkekben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt meghatározó:

- A hívási gráf készítő eszközök kiválasztása, módosítása, a hívási gráf kinyerésének céljából [1]
- Formátum átalakító eszköz megtervezése, kifejlesztése [1]
- Nyílt forrású rendszerek választása, mesterséges példák generálásának elvi kidolgozása, implementálása, validálása [1]
- A performanciamérés elvi kidolgozása, implementálása, a performanciamérés elvégzése, kiértékelése [1]
- A szoftvermetrika-alapú sérülékenység-előrejelzéssel kapcsolatos szakirodalom feldolgozása [2]
- A nsp, Snyk, GitHub oldalakról adatokat gyűjtő, egységesítő alkalmazás tervezése, implementációja [2]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A dinamikus nyelvekkel kapcsolatos elemzések motivációja [1, 2]

2021. április 28.

Dr. Gyimóthy Tibor

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Dávid Havas, István Siket, Árpád Beszédes, Rudolf Ferenc and József Mihalicza. **Transforming C++ 11 code to C++ 03 to support legacy compilation environments.** In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177-186. IEEE. Oct. 2016

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt meghatározó:

- Lehetséges transzformációs forgatókönyvek feltérképezése [1]
- Az inkrementális keretrendszer megtervezése és kidolgozása, az adattároláshoz szükséges séma kialakítása [1]
- Teszteléshez szükséges környezet kialakítása, fejlesztői folyamatokba való integráció kidolgozása [1]
- A kódtranszformációval kapcsolatos szakirodalom feldolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A projekt szakmai vezetése [1]

2021. április 28.

Dr. Beszédes Árpád

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Dávid Havas, István Siket, Árpád Beszédes, Rudolf Ferenc and József Mihalicza. **Transforming C++ 11 code to C++ 03 to support legacy compilation environments.** In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177-186. IEEE. Oct. 2016
2. Gábor Antal, Péter Hegedűs, Zoltán Tóth, Rudolf Ferenc, and Tibor Gyimóthy. **Static JavaScript Call Graphs: A Comparative Study.** In 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177–186, Sep. 2018
3. Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán and Tibor Gyimóthy. **Challenging machine learning algorithms in predicting vulnerable JavaScript functions.** In 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2019), pages. 8-14, IEEE, May 28, 2019
4. Gábor Antal, Zoltán Tóth, Péter Hegedűs and Rudolf Ferenc. **Enhanced Bug Prediction in JavaScript Programs with Hybrid Call-Graph Based Invocation Metrics.** In Technologies. 2021. MDPI; 9(1):3

cikkekben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- Az algoritmusok összehasonlítási módszerének elvi kidolgozása [2]
- A gépi tanulási algoritmusok eredményének kiértékelése, összehasonlítása [3]
- A gépi tanuló algoritmusok összehasonlítási módszerének elvi kidolgozása [3]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- Lehetséges transzformációs forgatókönyvek feltérképezése [1]
- Az inkrementális keretrendszer megtervezése és kidolgozása, az adattároláshoz szükséges séma kialakítása [1]
- Teszteléshez szükséges környezet kialakítása, fejlesztői folyamatokba való integráció kidolgozása [1]
- A kódtranszformációval kapcsolatos szakirodalom feldolgozása [1]
- A hívási gráf készítő eszközök kiválasztása, módosítása, a hívási gráf kinyerésének céljából [2]

- Formátum átalakító eszköz megtervezése, kifejlesztése [2]
- Nyílt forrású rendszerek választása, mesterséges példák generálásának elvi kidolgozása, implementálása, validálása [2]
- A performanciamérés elvi kidolgozása, implementálása, a performanciamérés elvégzése, kiértékelése [2]
- A szoftvermetrika-alapú sérülékenység-előrejelzéssel kapcsolatos szakirodalom feldolgozása [3]
- A nsp, Snyk, GitHub oldalakról adatokat gyűjtő, egységesítő alkalmazás tervezése, implementációja [3]
- A metrika-alapú hibaelőrejelzéssel kapcsolatos szakirodalom feldolgozása [4]
- A hibrid metrikákat előállító keretrendszer megtervezése és implementációja [4]
- A különböző jellemzőhalmazok előállítása [4]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A projekt hátterének biztosítása és koordinációja [1]
- A projekt szakmai koordinációja [2, 3, 4]
- A hibrid forráskódmetrika-alapú kutatás motivációja [4]

2021. április 28.

Dr. Ferenc Rudolf

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Péter Hegedűs, Zoltán Tóth, Rudolf Ferenc, and Tibor Gyimóthy. **Static JavaScript Call Graphs: A Comparative Study.** In 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177–186, Sep. 2018
2. Gábor Antal, Zoltán Tóth, Péter Hegedűs and Rudolf Ferenc. **Enhanced Bug Prediction in JavaScript Programs with Hybrid Call-Graph Based Invocation Metrics.** In Technologies. 2021. MDPI; 9(1):3

cikkekben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- A gépi tanulási algoritmusok eredményének kiértékelése, összehasonlítása [2]
- A gépi tanuló algoritmusok összehasonlítási módszerének elvi kidolgozása [2]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- A hívási gráf készítő eszközök kiválasztása, módosítása, a hívási gráf kinyerésének céljából [1]
- Formátum átalakító eszköz megtervezése, kifejlesztése [1]
- Nyílt forrású rendszerek választása, mesterséges példák generálásának elvi kidolgozása, implementálása, validálása [1]
- A performanciamérés elvi kidolgozása, implementálása, a performanciamérés elvégzése, kiértékelése [1]
- A metrika-alapú hibaelőrejelzéssel kapcsolatos szakirodalom feldolgozása [2]
- A hibrid metrikákat előállító keretrendszer megtervezése és implementációja [2]
- A különböző jellemzőhalmazok előállítása [2]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A hívási gráfokkal kapcsolatos szakirodalom feldolgozása [1]

2021. április 28.

Dr. Tóth Zoltán Gábor

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán and Tibor Gyimóthy. **Challenging machine learning algorithms in predicting vulnerable JavaScript functions.** In 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2019), pages. 8-14, IEEE, May 28, 2019

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt meghatározó:

- A szoftvermetrika-alapú sérülékenység-előrejelzéssel kapcsolatos szakirodalom feldolgozása [1]
- A nsp, Snyk, GitHub oldalakról adatokat gyűjtő, egységesítő alkalmazás tervezése, implementációja [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A sérülékenységek javítását metódusokra leképező alkalmazás készítése [1]

2021. április 28.

Gyimesi Péter

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Márton Keleti, and Péter Hegedűs. **Exploring the Security Awareness of the Python and JavaScript Open Source Communities.** In Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020), page 16-20, June 29-30, 2020.

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- Az SWHGD adatbázissal kapcsolatos performanciajavítások [1]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- A kutatás elvi módszerének kidolgozása [1]
- Az eredmények előállítása, kiértékelése, statisztikák készítése [1]
- A sérülékenységvizsgálatokkal kapcsolatos szakirodalom feldolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- Az SWHGD adatbázis importálása lokális környezetben [1]

2021. április 28.

Keleti Márton

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Dávid Havas, István Siket, Árpád Beszédes, Rudolf Ferenc and József Mihalicza. **Transforming C++ 11 code to C++ 03 to support legacy compilation environments.** In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177-186. IEEE. Oct. 2016

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekhez való hozzájárulásunk oszthatatlan:

- Transzformációs algoritmusok elvi kidolgozása, implementálása [1]
- Az ipari rendszereken való futás kiértékelése [1]

A következő eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- Lehetséges transzformációs forgatókönyvek feltérképezése [1]
- Az inkrementális keretrendszer megtervezése és kidolgozása, az adattároláshoz szükséges séma kialakítása [1]
- Teszteléshez szükséges környezet kialakítása, fejlesztői folyamatokba való integráció kidolgozása [1]
- A kódtranszformációval kapcsolatos szakirodalom feldolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- Az eszköz futtatása ipari és nyílt forrású rendszereken [1]
- Az eszköz párhuzamosításának megvalósítása [1]

2021. április 28.

Havas Dávid

Társszerzői nyilatkozat

Kijelentem, hogy ismerem *Antal Gábor* PhD fokozatra pályázó *„Applying Code Analysis and Machine Learning Techniques to Improve Compatibility and Security of Programs"* című disszertációját. A disszertációban szereplő és a

1. Gábor Antal, Dávid Havas, István Siket, Árpád Beszédes, Rudolf Ferenc and József Mihalicza. **Transforming C++ 11 code to C++ 03 to support legacy compilation environments.** In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 177-186. IEEE. Oct. 2016

cikkben publikált közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt meghatározó:

- Lehetséges transzformációs forgatókönyvek feltérképezése [1]
- Az inkrementális keretrendszer megtervezése és kidolgozása, az adattároláshoz szükséges séma kialakítása [1]
- Teszteléshez szükséges környezet kialakítása, fejlesztői folyamatokba való integráció kidolgozása [1]
- A kódtranszformációval kapcsolatos szakirodalom feldolgozása [1]

és ezeket az eredményeket nem használtam fel és a jövőben sem használom fel tudományos fokozat megszerzéséhez.

A következő eredményekben az én hozzájárulásom volt a meghatározó:

- A transzformációs keretrendszer motivációja [1]
- A transzformációs keretrendszer tesztelése az NNG Kft. környezetében [1]

2021. április 28.

Dr. Mihalicza József