**Eötvös Loránd Tudományegyetem**

Informatikai Kar

*Programozási nyelvek és Fordítóprogramok Tanszék*

Nem biztonságos kód biztonságossá transzformálása a RefactorErl

segítségével

**Témavezetők:** **Szerző:**

Dr. Tóth Melinda, Dr. Bozó István Juhász Alexandra Nelli

Egyetemi docens, Tudományos munkatárs Programtervező informatikus MSc.

**Budapest, 2025**

Absztrakt

A dolgozatom célja olyan automatikus, illetve részben automatikus transzformációk definiálása a RefactorErl statikus elemző eszközben, amelyek az Erlang nyelvben előforduló sérülékeny kódrészleteket biztonságossá alakítják. Az Erlang funkcionális, magas szintű programozási nyelv, amelyet gyakran használnak kritikus rendszerek – például banki, telekommunikációs és közlekedés szoftverek – fejlesztésére, így különösen fontos, hogy a forráskódokban ne maradjanak kiaknázható sérülékenységek. Statikus elemzési módszerekkel az ilyen pontok felderíthetők. A RefactorErl egy nyílt forráskódú statikus elemző eszköz, amely támogatja az Erlang fejlesztők munkáját különféle kódelemzési és keresési funkciókkal, valamint lehetőséget biztosít refaktorálások és transzformációk megvalósítására is. A kutatásom fő része azoknak a biztonságossá tevő mintáknak a meghatározása és implementálása, amelyek a program eredeti funkcionalitását megőrizve kizárják a potenciális sérülékenységeket. Ilyen például az inputok megfelelő ellenőrzése beszúrásos támadások megelőzésére. Célom olyan transzformációk létrehozása, amellyel növelhető a fejlesztési folyamat biztonsága és csökkenthető az emberi hibákból fakadó kockázat, ezzel hozzájárulva a megbízhatóbb Erlang alapú rendszerek megvalósításához.

Tartalomjegyzék

[1. Bevezetés 4](#_Toc210076629)

[1.1. A kutatás jelentősége 5](#_Toc210076630)

[2. Irodalmi áttekintés (10-12 oldal) 6](#_Toc210076631)

[2.1. Forráskód biztonsága 6](#_Toc210076632)

[2.2. Gyakori sérülékenységek a forráskódban 8](#_Toc210076633)

[2.2.1. Nem megfelelő input ellenőrzés 9](#_Toc210076634)

[2.2.2. Rossz hitelesítés és jelszókezelés 9](#_Toc210076635)

[2.2.3. Rossz fájl vagy memóriakezelés 10](#_Toc210076636)

[2.3. Statikus analízis szerepe a kódbiztonságban 10](#_Toc210076637)

[2.4. Erlang biztonsági szempontból 10](#_Toc210076638)

[2.5. RefactorErl további használati területei 10](#_Toc210076639)

[3. Sérülékeny kódrészletek és biztonsági minták (8-10 oldal) 11](#_Toc210076640)

[3.1. Gyakori biztonsági problémák az Erlangban 11](#_Toc210076641)

[3.2. Jellemző sérülékeny kódrészletek 11](#_Toc210076642)

[3.3. Biztonságossá tevő minták 11](#_Toc210076643)

[4. Sérülékeny Erlang kódok átalakításának módszertana RefactorErl segítségével (10-12 oldal) 12](#_Toc210076644)

[4.1. RefactorErl működése és komponensei 12](#_Toc210076645)

[4.2. A transzformációk definiálásának lehetőségei 12](#_Toc210076646)

[4.3. Általam alkalmazott módszerek 12](#_Toc210076647)

[5. Megvalósítás és eredmények (10-12 oldal) 13](#_Toc210076648)

[5.1. Implementált transzformációk bemutatása 13](#_Toc210076649)

[5.2. A transzformációk eredménye és tesztelése 13](#_Toc210076650)

[6. Összegzés és továbbfejlesztési lehetőségek (3-4 oldal) 14](#_Toc210076651)

[Irodalomjegyzék 15](#_Toc210076652)

[Forráskód jegyzék 17](#_Toc210076653)

# 1. Bevezetés

Az informatika és annak folyamatos fejlődése rengeteg iparágra nagy hatással van. A közlekedésben – például a forgalomirányítási rendszerek vagy az önvezető autók estében – elengedhetetlen a szoftverek biztonsága és védelme a balesetek elkerülése érdekében. Az egészségügyben létfontosságú a betegek adatainak tárolása és azoknak a kezelése, hiszen súlyos következményekkel járhatna, ha a rendszer egy támadás áldozata lenne. A pénzügyi szektorban például digitális banki szolgáltatások és az online fizetési rendszerek biztonságától függ a tranzakciók megbízhatósága. A telekommunikációs iparágban a hálózatok védelme rendkívül fontos a szolgáltatások akadálymentes működéséhez. Jól látható, hogy egyre több folyamat kerül szoftverek irányítása alá, ezzel jelentősen megkönnyítve az emberek mindennapi életét.

Ahogy nő az informatika és a digitalizáció szerepe a világon, úgy vele egyenesen arányosan nő a kibertámadások számossága (gyakorisága / száma), melyek egyre komplexebbek lesznek az idő előre haladtával. Fontos tisztázni, hogy mit is értünk kibertámadás alatt: olyan szándékos tevékenység, amelynek célja a számítógépes rendszerek vagy hálózatok működésének a megzavarása, hozzáférés bizalmas adatokhoz, azok módosítása, ellopása vagy végleges törlése. Ilyen tevékenységek mögött állhatnak egyéni hackerek vagy szervezett bűnözői csoportok. Éppen ezért, a forráskódok biztonsága is egyre nagyobb hangsúlyt kap napjainkban, hiszen a támadások mértékétől függően komoly társadalmi és gazdasági károkat tudnak okozni. Már egyetlen sérülékenység is elegendő lehet ahhoz, hogy a támadók hozzáférhessenek érzékeny információkhoz, befolyásolhassák a vállalatok működését vagy akár megbéníthassák az egész rendszert. Tehát a fejlesztés során a működőképesség mellett nem elég csak a teljesítményre összpontosítani, a programkód biztonságossá tételével is foglalkozni kell.

A biztonságos szoftverfejlesztésre már létrejöttek szabályok és irányelvek, amelyek meghatározzák, hogy mely biztonsági szempontokat érdemes figyelembe venni. Az ISO / IEC 27001 [2] a legelterjedtebb ilyen nemzetközi szabvány, melynek fő célja, hogy egy minimum követelményt állítson fel a megbízható működéshez. Egy keretrendszert biztosít a szervezeteknek az érzékeny adatok kezeléséhez, továbbá segít azonosítani és kezelni a különféle felmerülő kockázatokat. Említésre méltó még az OWASP (Open Web Application Security Project) [1] alapítvány kezdeményezése, amely útmutatókat és különféle irányelveket nyújt a biztonságos alkalmazások fejlesztéséhez.

Jól látható, hogy a szoftverbiztonság meghatározó tényező az élet minden pontján, éppen ezért különösen fontos, hogy olyan funkcionális nyelvekben is, mint az Erlang, a potenciális támadható rések időben felismerhetők és kezelhetők legyenek.

## 1.1. A kutatás jelentősége

Fejlesztés során a programozók nem tudnak minden lehetséges sérülékenységet előre látni és lekezelni, így azokat utólag manuálisan javítják. Legtöbb esetben a kézi javítás rendkívül időigényes és sokkal nagyobb a hibázási lehetőség, ebből kifolyólag elengedhetetlen ezeknek a folyamatoknak az automatizálása. A procedúra képes azonosan kezelni minden sebezhető kódrészletet, így egy megbízható és egységes megoldást kapunk a végén, mely minimalizálja a nem biztonságos részeket és csökkenti a támadások esélyét.

Az Erlang programozási nyelvhez kifejlesztett RefactorErl eszköz képes azonosítani a biztonsági kockázatot jelentő kódrészeket, azonban még csak a felismerésen van a hangsúly. A célom az, hogy kiegészítsem ezt a folyamatot olyan automatikus transzformációkkal, amelyek a sérülékeny kódrészeket biztonságosabbá alakítják. A transzformációk alatt olyan kódon belüli átalakításokat értek, amelyek módosítják a kód szerkezetét – például új függvény vagy elágazás beszúrása – viszont a program viselkedése és eredménye változatlan marad.

Ezáltal a kutatásom hozzájárulhat ahhoz, hogy az Erlang alapú rendszerek biztonságossá tétele ne csak hatékonyabbá és egységesebbé, hanem hosszú távon fenntarthatóbbá is váljon.

# 2. Irodalmi áttekintés (10-12 oldal)

Ahhoz, hogy kutatásom érthetőbb legyen, célszerűnek tartom áttekinteni a témához kapcsolódó legfontosabb háttérismereteket.

## 2.1. Forráskód biztonsága

Egyáltalán nem könnyű feladat olyan kódot írni, amely ellenálló különböző kibertámadásokkal szemben. Nem csak a kód olvashatóságára és érthetőségére kell figyelni, hanem a kód minőségére is. Ha az utóbbira kevesebb hangsúlyt fektetünk, nagymértékben megemelheti a kód fenntartási költségeit, és a hibák számát fejlesztés során. Ahhoz, hogy minél jobb minőségű legyen a kód, a következő szempontokat érdemes alapul venni a programmal kapcsolatban [4, 5]:

* megbízhatóság
* hibatűrés
* elérhetőség
* lehetséges sérülékenységek korai azonosítása

Annak érdekében, hogy a szoftverek megfeleljenek a fentebbi elvárásoknak, számos szoftverfejlesztést támogató elmélet és módszertan született. A programverifikáció egyik meghatározó alakja Hoare, aki kidolgozta a Hoare-logikát, különféle algoritmusokat (például quicksort) és számos programnyelv tervezésében is részt vett. Minél bonyolultabb és összetettebb egy szoftver, annál több és nehezebben azonosítható hibát rejt magában. Ezt a gondolatot Hoare a következőképpen fogalmazta meg [6]:

*„Arra a következtetésre jutottam, hogy kétféleképpen lehet felépíteni egy szoftvertervet: Az egyik, hogy annyira egyszerűvé tesszük, hogy nyilvánvalóan nincs benne hiba. A másik, hogy olyan bonyolulttá tesszük, hogy a hibák nem nyilvánvalók.”*

A forráskód biztonsága lassan már alap követelménnyé vált, mégis érdemes lenne tisztázni, hogy mit is értünk alatta pontosan. A fejlesztési folyamat azon részét jelenti, amely a kódban található sérülékenységek megelőzésére, felismerésére és azok kezelésére irányul. Legfőbb célja, hogy a forráskód ne tartalmazzon olyan elemeket vagy logikai hibákat, amelyek sebezhetők, így védelmet nyújt a támadásokkal szemben.

Különféle (például iparági) jelentések erőteljesen megerősítik a forráskód biztonságának fontosságát. A Verzion 2025-ös DBIR (Data Breach Investigations Report) [7] szerint az elkövetett incidensek jelentős része kódbiztonsági problémákhoz köthetők. Összesen 22 052 vizsgált biztonsági esemény közül 12 195 mutatkozott valódi adatvédelmi esetnek. A kódbeli sérülékenységek kihasználása 20%-ra emelkedett, ami tavalyhoz képest 34%-os növekedést jelent. A ransomware[[1]](#footnote-1) incidensek aránya is jelentősen megemelkedett, a 2025-ös évben az összes adatvédelmi esetek 44%-ában voltak jelen, szemben a tavalyi 32%-kal. Külön érdemes kiemelni, hogy az emberi tényezők továbbra is nagyban befolyásolják a támadások gyakoriságát, ugyanis becslések szerint az események 60%-ában játszottak szerepet [8]. Itt beszélhetünk akár rossz hitelesítésről és jelszókezelésről, nem megfelelő bemeneti adatok ellenőrzéséről vagy rossz rendszerkonfigurációról.

Számos szabvány és irányelv jött létre annak érdekében, hogy a vállalatok biztonságossá tegyék a szoftvereiket, és minimalizálják a forráskód sérülékenységéből adódó támadásokat. Ide köthető az ISO / IEC 27001 [2], ami a legelterjedtebb nemzetközi információbiztonsági szabvány. A tanúsítvány megszerzése rengeteg iparágban (például pénzügy, egészségügy, telekommunikáció) versenyelőnyt jelent, de van ahol kötelező jelleggel kell. Olyan folyamatokat és követelményeket határoz meg, amelyek alapján a szervezetek kialakíthatják, működtethetik és fejleszthetik az ISMS-üket (Information Security Management System). A szabvány előírja a biztonsági kockázatok azonosítását, azok kezelésére a megfelelő protokollok bevezetését, továbbá az érzékeny adatok megfelelő védelmét. Nemcsak a kód biztonságára fókuszál, hanem a teljes fejlesztési folyamatra is. Fontos, hogy ott is érvényesüljenek a biztonsági szempontok, mint például a jogosultság kezelés, verziókövetés és a fejlesztési környezetek megfelelő védelme. Egyik fő alapelve, hogy a biztonsági szabályozásokat a kockázatelemzés alapján kell meghatározni, és nem egy előre megadott sablon alapján. Éppen ezért bármilyen méretű vállalatra könnyedén alkalmazható a szabvány.

Míg az ISO / IEC 27001 főleg szervezeti szinten szabályozza az információbiztonságot, addig az OWASP (Open Web Application Security Project) [1] fejlesztői irányelvekre és a kódot érintő különböző sérülékenységekre összpontosít. Az OWASP egy olyan nemzetközi nonprofit szervezet, amely legfőbb célja, hogy javítsa a szoftverek biztonságát különböző útmutatókkal és eszközökkel. Legismertebb kiadványa az OWASP Top Ten, amely a webalkalmazásokat érintő tíz legfontosabb biztonsági kockázatot sorolja fel, és konkrét gyakorlati tanácsokat ad azok megelőzésére és kezelésére. Egy másik ismertebb irányelve az OWASP ASVS (Application Security Verification Standard) [3], amely részletes követelményrendszert nyújt a biztonságos alkalmazások fejlesztéséhez és ellenőrzéséhez. Szintekre bontva (1-től 3-ig) fogalmazza meg a különböző biztonsági követelményeket, így a vállalatok a saját igényeikhez mérten választhatják ki az ellenőrzések mélységét.

## 2.2. Gyakori sérülékenységek a forráskódban

Az OWASP ASVS [3, 4] szerint a legismertebb sérülékenységek egy forráskódban a következők:

* Nem megfelelő input ellenőrzés
* Hitelesítés és jelszókezelés
* Fájl vagy memóriakezelés
* Kimenet kódolása: Nem megfelelő kimenet kódolásakor a rendszer XSS (Cross-Site Scripting) támadásnak lehet kitéve, amely során a támadó rosszindulatú kódot futtathat a felhasználó böngészőjében.
* Munkamenet és jogosultság kezelés: Hibás munkamenet vagy jogosultságkezelés esetén illetéktelenek férhetnek hozzá védett erőforrásokhoz, ahol kártékony műveleteket is végrehajthatnak.
* Adatvédelem: Ha az érzékeny adatok nincsenek megfelelően titkosítva vagy biztonságosan kezelve, akkor könnyedén nyilvánosságra kerülhetnek, melyek súlyos problémákat okozhatnak.
* Adatbázis biztonság: Nem megfelelő adatbázis kezelés SQL injection támadáshoz vagy adatvesztéshez vezethet.
* Rendszerkonfiguráció: Alapértelmezett jelszavak, rosszul beállított jogosultságok vagy túl engedékeny rendszerbeállítások esetén biztonsági rések nyílhatnak a rendszerben, melyen keresztül a támadó könnyen bejuthat és kárt okozhat.

A következőkben részletesebben bemutatok néhány sérülékenységet és azok következményeit, melyek jól szemléltetik, hogy mennyire fontos a kód biztonsága. Amennyiben ezek figyelmen kívül maradnak, úgy komoly kockázatot jelenthetnek a rendszerek biztonságára nézve.

### 2.2.1. Nem megfelelő input ellenőrzés

A bemeneti adatok hibás vagy hiányos ellenőrzése lehetővé teszi, hogy a támadó ártalmas kódot vagy parancsot juttasson a rendszerbe. Ez különösen sérülékennyé teszi az alkalmazást az injection (beszúrásos) támadásokkal szemben.

Egyik leggyakoribb ilyen példa az SQL injection, amikor a támadó a bemeneti adatok között kártékony SQL parancsokat helyez el, amelyeket az alkalmazás végrehajt az adatbázisban. Ez nem kívánt műveleteket eredményezhet, például adatok módosítását, törlését, ellopását vagy akár az autentikáció megkerülését [5].

A hibás inputkezelés akár XSS (Cross-Site Scripting) támadáshoz is vezethet. Ilyenkor a támadó rosszindulatú kódrészletet juttat be egy űrlapon vagy egy beviteli mezőn keresztül, amelyet az alkalmazás megkötés nélkül megjelenít a weboldalon. A kód ekkor a felhasználók böngészőjében fog lefutni, lehetővé téve a támadónak az adatlopást vagy jogosulatlan műveletek végrehajtását.

Ezen felül a nem megfelelően kezelt bemenetek memóriahibákat is okozhatnak, amelyek a DoS/DDoS[[2]](#footnote-2) (túlterheléses) támadások alapját képezhetik.

### 2.2.2. Rossz hitelesítés és jelszókezelés

A hitelesítési folyamatok hibás vagy nem biztonságos megvalósítása, továbbá a jelszavak helytelen kezelése lehetővé teszi, hogy illetéktelen személyek férjenek hozzá védett adatokhoz vagy rendszerekhez. Tipikus hibák például:

* Gyenge jelszókövetelmény (rövid, könnyen kitalálható jelszavak),
* Általános jelszavak használatának engedélyezése (például „admin”),
* Jelszavak nem biztonságos tárolása vagy titkosítása,
* Többfaktoros hitelesítés hiánya,
* A session id-k helytelen kezelése: sikeres bejelentkezés után nem változik meg, vagy kijelentkezés után nincs megfelelően érvénytelenítve [5].

A brute force típusú támadások gyakran előfordulnak, különösen akkor, ha a rendszer nem korlátozza a bejelentkezési kísérletek számát. Ilyenkor a támadó manuálisan, vagy jellemzően automatizált eszközökkel sorra kipróbálja a lehetséges kulcs és jelszó párosokat addig, amíg be nem tud jelentkezni. Ezt követően a támadó hozzáférhet érzékeny adatokhoz, manipulálhatja az információkat és a szolgáltatásokat, megzavarhatja az üzleti folyamatokat, illetve további rendszerek felé is terjeszkedhet.

### 2.2.3. Rossz fájl vagy memóriakezelés

Fájlok nem megfelelő kezeléséről például akkor beszélhetünk, amikor nem biztonságosan vannak létrehozva ideiglenes fájlok, nincsenek megfelelően kezelve az elérési útvonalak, vagy fájlfeltöltéskor nincsenek ellenőrizve a bemeneti fájlok.

A felsorolt hibák különféle következményekkel járhatnak. Az első kettő esetben a támadó könnyedén hozzáférhet különböző védett fájlokhoz (például konfigurációs), amelyekből információt szivárogtathat ki. Az ellenőrizetlen fájlfeltöltés injection típusú támadáshoz vezethet, például amikor egy rosszindulatú script fájl felkerül a szerverre és RCE-t (Remote Code Execution) eredményez, vagyis a támadó képes lesz távolról kódot futtatni.

Memóriakezelési hibák akkor jelentkeznek, ha a program helytelenül kezeli az erőforrásokat. Ilyen eset lehet például a hibás indexelés, a túlzott memóriafoglalás vagy az erőforrások kimerítése. Ezek a problémák elsősorban DoS/DDoS típusú támadásokhoz vezethetnek, de súlyosabb esetben akár RCE-t is lehetővé tehetnek.

## 2.3. Statikus analízis szerepe a kódbiztonságban

## 2.4. Erlang biztonsági szempontból

## 2.5. RefactorErl további használati területei

# 3. Sérülékeny kódrészletek és biztonsági minták (8-10 oldal)

## 3.1. Gyakori biztonsági problémák az Erlangban

## 3.2. Jellemző sérülékeny kódrészletek

## 3.3. Biztonságossá tevő minták

# 4. Sérülékeny Erlang kódok átalakításának módszertana RefactorErl segítségével (10-12 oldal)

## 4.1. RefactorErl működése és komponensei

## 4.2. A transzformációk definiálásának lehetőségei

## 4.3. Általam alkalmazott módszerek

# 5. Megvalósítás és eredmények (10-12 oldal)

## 5.1. Implementált transzformációk bemutatása

## 5.2. A transzformációk eredménye és tesztelése

# 6. Összegzés és továbbfejlesztési lehetőségek (3-4 oldal)

# Irodalomjegyzék

[1] OWASP Foundation: Open Web Application Security Project, 2025. <https://owasp.org/>  
(Elérés dátuma: 2025. 09. 27.)

[2] ISO / IEC 27001: Information security, cybersecurity and privacy protection — Information security management systems — Requirements, 2022. <https://www.iso.org/standard/27001>  
(Elérés dátuma: 2025. 09. 27.)

[3] The OWASP Foundation. Application Security Verification Standard, 2025. <https://owasp.org/www-project-application-security-verification-standard/>  
(Elérés dátuma: 2025. 09. 22.)

[4] Brigitta Baranyai, István Bozó, Melinda Tóth: *„Supporting Secure Coding with RefactorErl”,* Eötvös Loránd Tudományegyetem, Informatikai Kar, Programozási Nyelvek és Fordítóprogramok Tanszék, 2024.

[5] Baranyai Brigitta: *„Funkcionális nyelvek és a statikus kódelemzéssel támogatott biztonságos szoftverfejlesztés*”, TDK dolgozat, Eötvös Loránd Tudományegyetem, Informatikai Kar, Programozási Nyelvek és Fordítóprogramok Tanszék, 2020.

[6] Charles Antony Richard Hoare: *„The emperor’s old clothes”*, Communications of the ACM, 24.2, 1981, 75 – 83. <https://dl.acm.org/doi/10.1145/358549.358561>  
(Elérés dátuma: 2025. 09. 27.)

[7] Verzion: Data Breach Investigations Report (DBIR) 2025, <https://www.verizon.com/business/resources/reports/2025-dbir-data-breach-investigations-report.pdf>  
(Elérés dátuma: 2025. 09. 28.)

[8] Mimecast Blog: Verzion: 60% of breaches involve human error, 2025, <https://www.mimecast.com/blog/verizon-60-of-breaches-involve-human-error/>  
(Elérés dátuma: 2025. 09. 28.)

A könyvek idézésének formája a következő:  
[<hivatkozási azonosítója>] <szerző neve>: <a könyv címe>, <a kiadó neve>, <a kiadás éve>, [<terjedelme>], <ISBN szám>. Például: [2] Jan Sommerville: Software Engineering, Addison Wesley, 1988, [742], ISBN-0201- 42765-6.  
Folyóiratcikk idézésének formája a következő:  
[<hivatkozási azonosítója>] <szerző neve>: <a cikk címe>, <a folyóirat címe>, <Vol. száma>, <évszám>, [terjedelme].  
Hálózaton elérhető forrásmunka esetén a „link” megadása után adjuk meg az **elérés dátumát**!

# Forráskód jegyzék

1. Olyan rosszindulatú program, mely titkosítja az áldozat adatait vagy rendszereit, és a támadó váltságdíjat követel a feloldásért vagy az ellopott adatok visszatartásáért. [↑](#footnote-ref-1)
2. Denial of Service / Distributed Denial of Service [↑](#footnote-ref-2)