**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 (3-5 oldal) 4](#_Toc209393924)

[1.1. A kutatás jelentősége 5](#_Toc209393925)

[2. Irodalmi áttekintés (10-12 oldal) 6](#_Toc209393926)

[2.1. Forráskód biztonsága 6](#_Toc209393927)

[2.2. Statikus analízis szerepe a kódbiztonságban 6](#_Toc209393928)

[2.3. Erlang biztonsági szempontból 6](#_Toc209393929)

[2.4. RefactorErl további használati területei 6](#_Toc209393930)

[3. Sérülékeny kódrészletek és biztonsági minták (8-10 oldal) 7](#_Toc209393931)

[3.1. Gyakori biztonsági problémák az Erlangban 7](#_Toc209393932)

[3.2. Jellemző sérülékeny kódrészletek 7](#_Toc209393933)

[3.3. Biztonságossá tevő minták 7](#_Toc209393934)

[4. Sérülékeny Erlang kódok átalakításának módszertana RefactorErl segítségével (10-12 oldal) 8](#_Toc209393935)

[4.1. RefactorErl működése és komponensei 8](#_Toc209393936)

[4.2. A transzformációk definiálásának lehetőségei 8](#_Toc209393937)

[4.3. Általam alkalmazott módszerek 8](#_Toc209393938)

[5. Megvalósítás és eredmények (10-12 oldal) 9](#_Toc209393939)

[5.1. Implementált transzformációk bemutatása 9](#_Toc209393940)

[5.2. A transzformációk eredménye és tesztelése 9](#_Toc209393941)

[6. Összegzés és továbbfejlesztési lehetőségek (3-4 oldal) 10](#_Toc209393942)

[Irodalomjegyzék 11](#_Toc209393943)

[Forráskód jegyzék 12](#_Toc209393944)

# 1. Bevezetés (3-5 oldal)

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, 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, szervezett bűnözői csoportok vagy akár nagyobb hatalommal rendelkező emberek. 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[[1]](#footnote-1) 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[[2]](#footnote-2) alapítvány kezdeményezése, melynek a legismertebb kiadványa az OWASP Top Ten. Ez a dokumentum a webalkalmazásokat érintő tíz legfontosabb biztonsági kockázatot sorolja fel, és konkrét tanácsokat ad azok megelőzésére. Emellett útmutatókat és különféle irányelveket is nyújt a biztonságos alkalmazások fejlesztéséhez.

Az Alkalmazásbiztonsági Ellenőrzési Szabvány (ASVS) [1] szerint a következők a legismertebb sérülékenységek egy forráskódban:

* Nem megfelelő bemenet ellenőrzés: Hiányos bemeneti adat vagy annak nem megfelelő ellenőrzése esetén könnyedén előfordulhat, hogy rosszindulatú bemeneti adat kerül a rendszerbe.
* Hitelesítés és jelszókezelés: Gyenge hitelesítés vagy jelszavak nem megfelelő tárolása esetén illetéktelenek könnyen hozzáférhetnek az erőforrásokhoz vagy bizalmas adatokhoz.
* Munkamenet és jogosultság kezelés: Ezek hibásan kezelésekor illetéktelenek férhetnek hozzá privát adatokhoz és rendszererőforrásokhoz.
* Kimenet kódolása: Nem megfelelő kimenet kódolásakor a rendszer Cross Site Scripting (XSS) támadásnak lehet kitéve, amikor a támadó rosszindulatú kódot juttathat a rendszerbe, amely klines oldalon (pl. böngészőben) fut le.
* Rendszerkonfiguráció: Alapértelmezett jelszavak vagy túl engedékeny rendszerbeállítások esetén a támadó könnyen bejuthat a rendszerbe.
* Adatvédelem: Titkosítás hiánya vagy adatok nem megfelelő tárolása növeli a támadások sikerességének az esélyét.
* Adatbázis biztonság: Lekérdezések, érzékeny adatok helytelen kezelése adatvesztéshez vagy SQL injekcióhoz vezethet.
* Fájl vagy memóriakezelés: Ellenőrizetlen fájlműveletek injekciós támadásokhoz vagy fájlmódosításokhoz vezethetnek, míg a memória hibák (pl. erőforrás kimerítése, túlindexelés) potenciális célponttá tehetik a rendszert DoS/DDoS[[3]](#footnote-3) (túlterheléses) támadások számára.

A felsorolt sérülékenységek és következményeik jól mutatják, hogy mennyire fontos a kód biztonsága, és nagy kockázattal jár, ha ezekkel nem törődünk, vagy bizonyos dolgokat nem jól kezelünk.

## 1.1. A kutatás jelentősége

Fejlesztés közben nem tudunk minden lehetséges sérülékeny kódrészletet lefedni és biztonságossá tenni, továbbá kézi javítás során sokkal nagyobb a hibázási lehetőség. Ebből kifolyólag elengedhetetlen ezeknek a folyamatoknak az automatizálása, ezáltal minimalizálva a sebezhető kódrészeket és csökkentve a támadások esélyét.

Az Erlang programozási nyelvhez kifejlesztett RefactorErl eszköz képes azonosítani a biztonsági kockázatot jelentő kódrészeket. A dolgozatom fő célja, hogy kiegészítsem ezt a folyamatot olyan automatikus transzformációkkal, amik a támadható kódrészeket biztonságossá alakítják, így egy egységes és megbízható megoldást kaphatunk. A transzformációk alatt olyan kódon belüli átalakításokat értek, amik megváltoztatják a kód mellékhatását, viszont megőrzik annak működését.

# 2. Irodalmi áttekintés (10-12 oldal)

## 2.1. Forráskód biztonsága

## 2.2. Statikus analízis szerepe a kódbiztonságban

## 2.3. Erlang biztonsági szempontból

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

[2] Brigitta Baranyai, István Bozó, Melinda Tóth: Supporting Secure Coding with RefactorErl, Eötvös Loránd Tudományegyetem, Informatikai Kar, Programnyelvek és Fordítóprogramok Tanszék, 2024

[3]

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. Information security, cybersecurity and privacy protection — Information security management systems — Requirements

   <https://www.iso.org/standard/27001> [↑](#footnote-ref-1)
2. Open Web Application Security Project: <https://owasp.org/> [↑](#footnote-ref-2)
3. Denial of Service / Distributed Denial of Service [↑](#footnote-ref-3)