

Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
Informatikai Intézet

**Sérülékenység detekció statikus és dinamikus elemzés,
valamint nagy nyelvi modellek segítségével**

Szakdolgozat

Készítette:
Felegyi Mihály Patrik
üzemtechnikus-informatikus
szakos hallgató

Témavezető:
Dr. Hegedűs Péter
egyetemi adjunktus

Szeged
2025

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

FELADATKIÍRÁS

A szakdolgozat célja, hogy összehasonlítsa a sérülékenységek felderítésére szolgáló különböző technikákat, és bemutassa, hogyan lehet ezek kombinációjával hatékonyan azonosítani és megelőzni a szoftverekben található biztonsági réseket. Megvizsgáljuk, hogyan használhatók a nagy nyelvi modellek a statikus és dinamikus elemzések kiegészítésére.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

TARTALMI ÖSSZEFOGLALÓ

- ***A téma megnevezése:***

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

- ***A megadott feladat megfogalmazása:***

Egy olyan alkalmazás megvalósítása, amely különböző kód- és viselkedéselemzési technikákat integrál, valamint egy mesterséges intelligencia alapú megoldás kifejlesztése, amely képes a biztonsági hibák automatikus azonosítására. Vizsgálni fogjuk továbbá a nagy nyelvi modellek lehetőségeit a kód prediktív elemzésére, beleértve a lehetséges biztonsági réseket és a sérülékenységek kockázatainak felmérését.

- ***A megoldási mód:***

A dolgozat célja sérülékenységek detektálása statikus és dinamikus elemzéssel, nagy nyelvi modellek kiegészítésével. Python/Flask alapú webalkalmazás készült, amely eszközöket integrál weboldalak vizsgálatára. Az eredmények JSON/ZIP-ben, a jelentések DOCX formátumban érhetőek el.

- ***Alkalmazott eszközök, módszerek:***

Python programozási nyelv

Flask keretrendszer

HTML leírónyelv

CSS leírónyelv

JavaScript programozási nyelv

venv eszköz

shutil, requests, json, subprocess, logging, urllib.parse, docx, os, datetime, zipfile, io, collections, semgrep függvénykönyvtárak

Nikto, Arachni, Wapiti, DeepSeek-V3 eszközök

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

- ***Elért eredmények:***

Az alkalmazás sikeresen integrálja több biztonsági eszköz funkcionalitását egyetlen felületen, lehetővé téve weboldalak sebezhetőségének vizsgálatát különböző módokban (statikus, dinamikus, LLM). JSON formátumban szolgáltatja az eredményeket, amelyeket HTML/CSS felületen jelenít meg, továbbá DOCX jelentést és nyers kimeneteket generál, amelyek JSON formátumúak, és egyetlen ZIP fájlként letölthetőek. A ZIP fájlban a nyers kimenetek eszközönként külön fájlokban találhatók, például wapiti.json és arachni.json, amelyek grep-pel kereshetőek. A debug log letöltése támogatja a hibakeresést, így rugalmas és felhasználóbarát megoldást kínál biztonsági elemzésekhez.

- ***Kulcsszavak:***

Biztonsági szkennelés, Webalkalmazás, Sérülékenység vizsgálat, Statikus elemzés, Dinamikus elemzés, LLM analízis

TARTALOMJEGYZÉK

FELADATKIÍRÁS.....	2
TARTALMI ÖSSZEFOGLALÓ.....	3
TARTALOMJEGYZÉK.....	5
BEVEZETÉS.....	8
1. TECHNOLÓGIAI HÁTTÉR.....	10
1.1. Programnyelvek és technológiák.....	10
1.1.1. Python.....	10
1.1.2. HTML.....	10
1.1.3. CSS.....	10
1.1.4. JavaScript.....	10
1.1.5. Flask.....	11
1.1.6. Semgrep.....	11
1.1.7. Nikto.....	11
1.1.8. Arachni.....	11
1.1.9. Wapiti.....	11
1.1.10. DeepSeek-V3.....	12
1.2. Fejlesztői környezetek.....	12
2. SPECIFIKÁCIÓ.....	14
2.1. Felhasználói felület.....	14
2.2. Backend funkcionalitás.....	15
2.3. Technikai követelmények.....	15

2.4. Biztonsági megfontolások.....	15
3. TERVEZÉS.....	16
3.1. Architektúra.....	16
3.2. Működés.....	17
3.3. Felhasználói élmény.....	19
4. MEGVALÓSÍTÁS.....	20
4.1. Az alkalmazás alapja: app.py.....	20
4.2. Frontend: index.html és stíluslap.css.....	21
4.3. Biztonsági vizsgálati eszközök integrálása.....	24
4.4. Jelentés generálás.....	24
4.5. Nyers kimenet és napló letöltése.....	25
4.6. Hibakezelés és naplózás.....	25
4.7. Biztonsági szempontok.....	26
4.8. Optimalizálás.....	27
5. TESZTELÉS.....	28
5.1. A tesztelés folyamata.....	28
5.2. Tesztesetek.....	29
6. BŐVÍTÉSI ÖTLETEK.....	35
ÖSSZEFOGLALÁS.....	36

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

IRODALOMJEGYZÉK.....	37
NYILATKOZAT.....	38
KÖSZÖNETNYILVÁNÍTÁS.....	39
MELLÉKLETEK.....	40

BEVEZETÉS

A digitális világ térhódításával párhuzamosan a számítógépes rendszerek és webalkalmazások biztonsága egyre nagyobb hangsúlyt kapott. Míg korábban elsősorban hagyományos infrastruktúrákat érintettek a támadások, mára a webalkalmazások váltak az egyik leggyakoribb célponttá – részben azok nyitottsága, komplexitása és a gyors fejlesztési ciklusok miatt. A modern üzleti folyamatok jelentős része webes szolgáltatásokon keresztül zajlik, így ezek védelme kritikus jelentőségűvé vált. A kibertámadások száma folyamatosan nő, a támadók pedig egyre kifinomultabb módszereket alkalmaznak a rendszerek feltérképezésére és kihasználására.

Ennek következtében a sérülékenységvizsgálat, valamint a manuális és automatikus detekciós módszerek fejlesztése napjainkra nem csupán technológiai kihívássá, hanem stratégiai fontosságú feladattá vált. Ugyanakkor az ilyen rendszerek fejlesztése számos kihívással jár: a fals pozitív riasztások magas aránya, a különböző technológiákhoz való illeszkedés nehézsége, valamint az, hogy a támadások formája gyorsan változik és alkalmazkodik a védekezési technikákhoz.

Az utóbbi években különösen nagy figyelmet kaptak a webes sérülékenységek, amelyek között olyan technikák is megjelentek, mint a nagy nyelvi modelleket célzó prompt injection [1], a 2005-ben először dokumentált, majd 2019-ben újra előtérbe került HTTP-kérés csempészs (HTTP request smuggling) [2], vagy a 2024-ben hírhedtté vált webes gyorsítótár megtévesztés (web cache deception) [3]. Ezek a példák jól mutatják, hogy az alkalmazások támadási felülete, különösképpen a webalkalmazások támadási felülete egyre komplexebb, így a kiberbiztonsági szakembereknek is lépést kell tartaniuk a technológiai fejlődéssel.

Cél egy olyan eszköz megvalósítása, amely képes webalkalmazások sebezhetőségeinek azonosítására három különböző megközelítés kombinálásával: statikus elemzés, dinamikus elemzés és nagy nyelvi modelleken (LLM) alapuló elemzés segítségével. Az alkalmazott módszerek egymást kiegészítve biztosítanak átfogó képet a vizsgált rendszer biztonsági állapotáról.

A dolgozatban statikus elemzésként úgynevezett statikus webes válaszelemzés történik, tehát azt vizsgáljuk, amit a szerver egy-egy HTTP-kérésre válaszként küld. Ez lehetővé teszi az

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

alkalmazás biztonsági jellemzőinek értékelését HTTP-fejlécek, státuszkódok vagy kiszolgált tartalmak alapján.

A dinamikus elemzés ezzel szemben aktív interakciót jelent a célalkalmazással. Ebben az esetben futás közbeni megfigyelés és különféle tesztelési technikák – például bemeneti manipulációk és célzott lekérdezések – segítségével igyekszünk feltárni a potenciális sebezhetőségeket.

A dolgozat egyik innovatív eleme egy nagy nyelvi modell (a DeepSeek V3) alkalmazása a begyűjtött válasz automatikus értelmezésére. Az ilyen modellek képesek emberi nyelven megfogalmazott logikát követni, veszélyes mintákat felismerni, és akár javaslatokat is tenni a lehetséges biztonsági javításokra.

A dolgozat és a hozzá tartozó kód elérhető a következő linken:

<https://github.com/FMisi/Szakdolgozat-CC0LRB>

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

1. TECHNOLÓGIAI HÁTTÉR

1.1. Programnyelvek és technológiák

1.1.1. Python

A Python egy magas szintű, általános célú programozási nyelv, amelyet Guido van Rossum alkotott meg, és 1991-ben jelent meg először. A Python fejlesztése során a kód olvashatóságát és a programozói munka megkönnyítését helyezte előtérbe. Jellegzetes tulajdonsága a behúzáson alapuló blokkszerkezet (indentation), amely egyértelműen tükrözi a program szerkezetét. A nyelv platformfüggetlen, dinamikusan típusos, és rendkívül széles körben alkalmazható: például grafikus felhatalnáló felületek megvalósítása, adatelemzés, gépi tanulás, automatizálás és tudományos számítások terén egyaránt. [4]

1.1.2. HTML

A HTML (HyperText Markup Language) egy leíró nyelv, amely weboldalak szerkezetének meghatározására szolgál. A W3C által támogatott szabvány legújabb változata a HTML5. Alapvető szerepet tölt be az internetes tartalmak megjelenítésében. [5]

1.1.3. CSS

A CSS (Cascading Style Sheets) a HTML és XHTML dokumentumok megjelenésének formázására szolgáló nyelv. Ezenkívül használható bármilyen XML alapú dokumentum stílusának leírására is, mint például az SVG. Segítségével szétválasztható a tartalom és a megjelenés, így a weboldalak dizájnja könnyen módosítható. A „kaszád” elv alapján több stílus definíció esetén prioritásrend alapján kerül alkalmazásra. [6, 7, 8]

1.1.4. JavaScript

A JavaScript egy dinamikus, objektumorientált, prototípus-alapú programozási nyelv, amelyet elsősorban weboldalak interaktív elemeinek megvalósítására használnak. Az ECMAScript szabvány szerint fejlődik, és a Node.js révén szerveroldali fejlesztésre is alkalmassá vált. Népszerűsége és alkalmazhatósága jelentős, amit olyan keretrendszerek is erősítenek, mint a React, Angular vagy Vue.js. A modern webfejlesztés alapvető eleme. [9, 10]

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

1.1.5. Flask

A Flask egy Python-alapú, könnyű mikro webes keretrendszer, amely a WSGI specifikációra épül. Nem tartalmaz beépített megoldásokat komplexebb funkciókhoz (pl. adatbázis-kezelés), azonban könnyen bővíthető. A Jinja templatemotorra és a Werkzeug eszköztárra támaszkodik. Egyszerűsége és rugalmassága miatt kiváló választás gyors webalkalmazás-fejlesztésre. [11, 12]

1.1.6. Semgrep

A Semgrep egy nyílt forráskódú statikus kódelemző eszköz, amely gyors és rugalmas szabályalapú vizsgálatot tesz lehetővé különböző nyelveken. Deklaratív szabálynyelvre és folyamatosan bővülő közösségi szabálykészlete révén hatékonyan integrálható CI/CD folyamatokba. Főként biztonsági auditokra és kódminőség ellenőrzésére használatos. [13, 14]

1.1.7. Nikto

A Nikto egy nyílt forráskódú webkiszolgáló-szkenner, amely ismert sebezhetőségek és hibás konfigurációk felismerésére képes. Több mint 1250 féle elavult szerververziót, és több mint 7000-féle potenciálisan veszélyes fájlt, ezen felül hibás beállításokat ellenőriz.

Működése során nagy mennyiségű HTTP lekérdezést hajt végre, ami hasznos lehet behatolásészlelési rendszerek tesztelésére is. [15]

1.1.8. Arachni

Az Arachni egy Ruby-alapú, nyílt forrású, moduláris webalkalmazás-szkenner, amely automatizáltan képes feltérképezni dinamikus webes sebezhetőségeket. Támogatja a modern technológiákat, mint a JavaScript, DOM-manipuláció, AJAX. Webes kezelőfelülettel és jelentéskészítő eszközökkel is rendelkezik. Bár fejlesztése leállt, még mindig hasznos eszköz lehet dinamikus elemzésre. [16]

1.1.9. Wapiti

A Wapiti egy „black-box” típusú, Python-alapú biztonsági szkenner, amely a webalkalmazások futó felületét vizsgálja meg, a forráskód ismerete nélkül. Az oldal

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

struktúrájának feltérképezése után különféle payloadokat használva teszteli a bemenetek sebezhetőségét. Támogatja a GET és POST metódusokat, valamint a fájlalapú injektálást is. Rugalmas, könnyen kezelhető eszköz. [17]

1.1.10. DeepSeek-V3

A DeepSeek-V3 egy open-source nagy nyelvi modell (LLM), amely természetes nyelvi feldolgozási (NLP) feladatokra készült. Több milliárd paraméterrel rendelkezik, modern transformer-architektúrára épül, és gyors, skálázható működést biztosít. Alkalmazható chatbotokban, szöveggenerálásban és adatfeldolgozó rendszerekben is. [18, 19, 20]

1.2. Fejlesztői környezetek

Fejlesztői környezet:

- Visual Studio Code 1.98.2

A Python, HTML, CSS, és JavaScript kódok fejlesztésére a Visual Studio Code (VS Code) nevű ingyenes, modern fejlesztői környezetet használtam. A VS Code, bár kis méretű, rendkívül erőteljes eszköz, amely számos hasznos funkcióval rendelkezik. A VS Code kiemelkedő előnye, hogy automatikusan felismeri az adott nyelv szintaxisát, és különböző bővítmények felajánlásával (például szintaxis kiemelés, automatikus kiegészítés és hibajelzés) segíti a kód gyors és pontos írását.

Teszteléshez:

- Firefox 136.0.1
- Google Chrome 134.0.6998
- Microsoft Edge 134.0.3124

Nem elég egyetlen böngészőre tesztelni az adott programot, érdemes a három legnagyobb böngésző legújabb verzióján elvégezni a tesztelést.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

Egyéb:

- draw.io v26.1.1, az ábrákhoz
- Flameshot v12.1.0, a képernyőképekhez

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

2. SPECIFIKÁCIÓ

A biztonság kiemelten fontos a növekvő kibertámadások miatt. A Biztonsági Vizsgálati Eszköz célja, hogy különböző tesztekkel átfogó képet adjon a webalkalmazások sérülékenységeiről, könnyen értelmezhető jelentésekkel, különböző absztrakciós szinteken tesztelve (pl. válaszelemzés, támadásszimuláció).

2.1. Felhasználói felület

Főoldal és vizsgálati felület: Az alkalmazás központi felülete lehetővé teszi az URL megadását és vizsgálati mód kiválasztását. A felület egyszerű és felhasználóbarát. A fő funkciók:

- URL mező a weboldal címének megadására.
- Vizsgálati mód választása: Statikus (Semgrep), Dinamikus (Nikto, Arachni, Wapiti), LLM (DeepSeek-V3), vagy Összes eszköz.
- Gombok: Vizsgálat indítása, Jelentés generálása, Log letöltése, Nyers output letöltése. Az eredmények strukturált formában jelennek meg.
- Jelentés generálás: Automatikusan generál egy .docx jelentést a vizsgálat eredményeiről. A jelentés tartalmazza a vizsgált URL-t, a sérülékenységek listáját, a sérülékenységek előfordulásának számát az adott sérülékenység mellett zárójelben, továbbá azt, hogy hány eszköz fedezte fel az adott sérülékenységet.
- Nyers output letöltése: ZIP fájlban elérhetőek az eszközök kimenetei JSON formátumban.
- Log letöltése: A debug log fájl (debug.log) letölthető.

Dizájn: Modern dizájn világosszürke háttérrel, kék gombokkal, reszponzivitással és egyedi faviconnal.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

2.2. Backend funkcionalitás

Az alkalmazás Flask alapú, és több módszert kínál:

- **Statikus mód:** Semgrep elemzi a kódot YAML szabályokkal.
- **Dinamikus mód:** Nikto, Arachni, Wapiti szkennelésekkel.
- **LLM mód:** DeepSeek-V3 modell HTTP fejlécek elemzésére.
- **Összes eszköz:** Kombinált eszközök futtatása.

Fájlkezelés: Az eszközök ideiglenes fájlokat hoznak létre, amelyek futás után törlődnek.

API végpontok:

- `/scan` (POST): Vizsgálat indítása.
- `/generate_report` (POST): Riport generálása.
- `/download_raw_zip` (POST): Nyers kimenet letöltése.
- `/download_log` (GET): Log letöltése.

Hibakezelés: Érvénytelen URL esetén 400 hiba, eszköz hibák esetén JSON hibaüzenet.

2.3. Technikai követelmények

A rendszer Python 3.x, venv, és Flask használatával működik. Szükséges könyvtárak: `shutil`, `requests`, `json`, `subprocess`, `logging`, `urllib.parse`, `docx`, `os`, `datetime`, `zipfile`, `io`, `collections`, `semgrep`. Külső eszközök: Nikto, Arachni, Wapiti, DeepSeek V3. Az LLM módhoz OpenRouter API kulcs szükséges.

2.4. Biztonsági megfontolások

A kérésekhez egyedi User-Agent, az API kulcs környezeti változóban tárolandó. Az Arachni riportok automatikusan törlésre kerülnek az elemzés után a felesleges tárhelyhasználat elkerülésére.

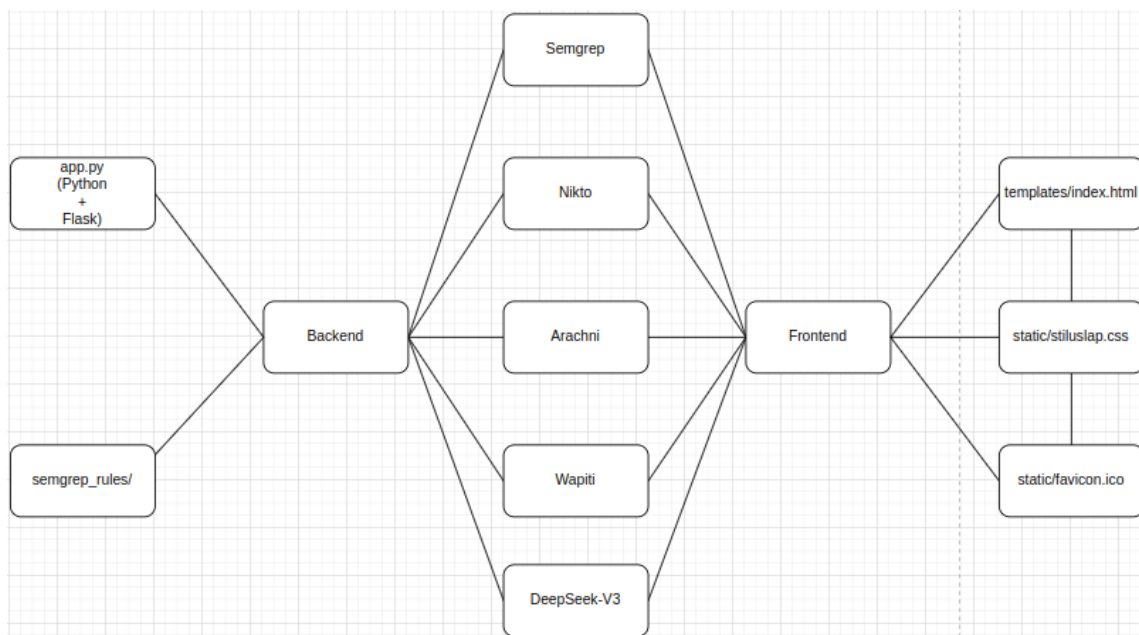
Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

3. TERVEZÉS

3.1. Architektúra

Az alkalmazás három rétegből épül fel:

- Frontend: letisztult webes felület, ahol a felhasználó megadja a vizsgálandó URL-t és elindítja a folyamatot.
- Backend: Python + Flask alapú szerveroldali logika, amely koordinálja a vizsgálatokat és összegzi az eredményeket.
- Vizsgálati réteg: külső eszközök (Semgrep, Nikto, Arachni, Wapiti, DeepSeek-V3), amelyeket a rendszer subprocess segítségével indít el, kivéve az DeepSeek-V3 eszközt, ugyanis az speciális API hívással indul.

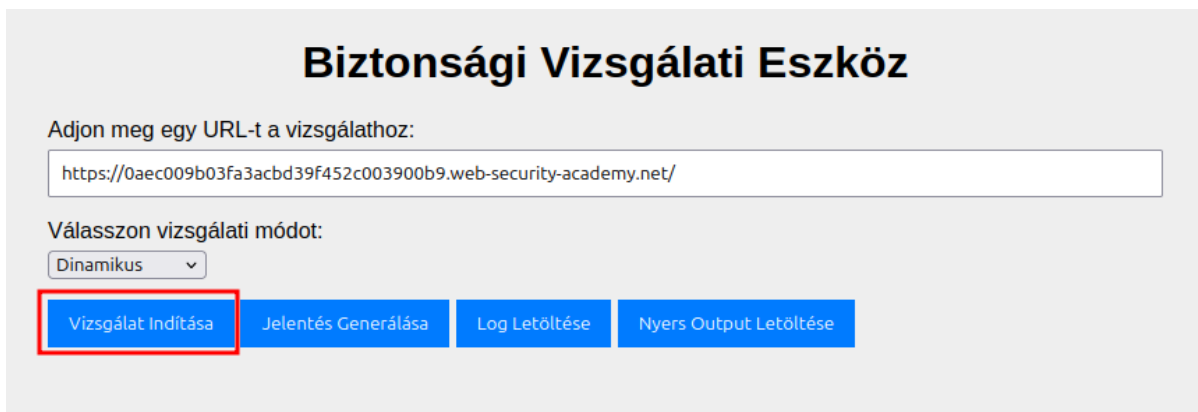


3.1. ábra – Az alkalmazás architektúrája

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

3.2. Működés

A felhasználó megadja a vizsgálandó oldalt és kiválaszthatja a vizsgálati módot, majd a backend automatikusan végrehajtja a kapcsolódó vizsgálatokat a „Vizsgálat Indítása” gomb megnyomása után. Az eredmény kimeneteket a rendszer összefoglalja, opcionálisan generál belőle letölthető jelentést is.



3.2. ábra – A vizsgálat indításának működése

Miután a „Jelentés Generálása” gombra kattint a felhasználó, a program automatikusan generál egy .docx jelentést a vizsgálat eredményeiről.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével



Biztonsági Vizsgálati Jelentés

Vizsgált URL: <https://0a4600fe0426842e8048264c006c0009.web-security-academy.net/>

HTTP Secure Headers (3)

Detektálva 1 eszköz által.

Interesting Response (2)

Detektálva 1 eszköz által.

HTTP Strict Transport Security (HSTS) (2)

Detektálva 2 eszköz által.

Common Directory Found (1)

Detektálva 1 eszköz által.

Session Cookie Without the secure Flag Set (1)

Detektálva 1 eszköz által.

Session Cookie Without the httponly Flag Set (1)

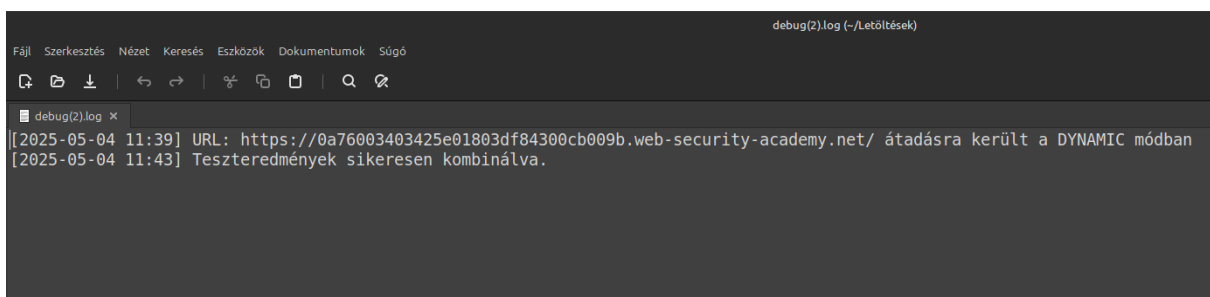
Detektálva 1 eszköz által.

Content Security Policy Configuration (1)

Detektálva 1 eszköz által.

3.3. ábra – Egy kigenerált jelentés tartalma

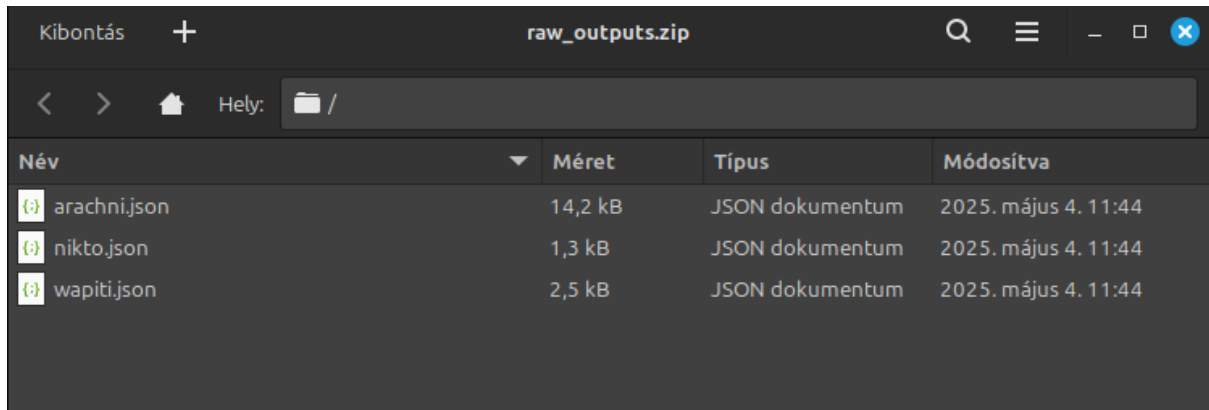
A „Log Letöltése” gombra kattinva a program letölti a debug naplót debug.txt néven.



3.4. ábra – Bejegyzések a debug.txt-ben

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

A „Nyers Output Letöltése” gombra kattinva letölthetőek az eszközök nyers kimenetei ZIP formátumban, raw_outputs.zip néven.



3.5. ábra – Egy vizsgálat utáni raw_outputs.zip tartalma

3.3. Felhasználói élmény

Az egész alkalmazás úgy lett kialakítva, hogy minimális lépésben elindítható legyen a teljes vizsgálat. A vizuális megjelenés letisztult, reszponzív, és csak a legszükségesebb elemeket tartalmazza.

4. MEGVALÓSÍTÁS

A szakdolgozat célja egy webalkalmazás fejlesztése volt, amely különböző biztonsági vizsgálati eszközöket integrál egy könnyen használható felületen keresztül. Az implementáció során törekedtem a modularitásra, az újrahasznosíthatóságra és a fenntarthatóságra, hogy a kód könnyen bővíthető és karbantartható legyen. Az alkalmazás fejlesztése során logikusan, lépésről lépésre haladtam, különös figyelmet fordítva a hibakezelésre és a naplózásra.

4.1. Az alkalmazás alapja: *app.py*

Az alkalmazás fő komponense az *app.py* fájl, amely egy Flask alapú webszervert valósít meg. Ez a fájl felelős az összes backend logika kezeléséért, beleértve a HTTP kérések fogadását, a biztonsági vizsgálatok futtatását és a jelentések generálását. A Flask keretrendszer választása lehetővé tette a gyors fejlesztést és a rugalmas routolást. Az alkalmazás modularitását a jól elkülönített függvényekkel biztosítottam, például a *download_html*, *run_semgrep*, *run_nikto*, *run_arachni*, *run_wapiti* és *run_llm* függvényekkel, amelyek külön-külön kezelik az egyes vizsgálati eszközök futtatását. A naplózási rendszer (*log_debug_message*) implementálásával biztosítottam, hogy minden fontos esemény és hiba rögzítésre kerüljön a *debug.log* fájlban, ami nagyban megkönnyítette a hibakeresést.

A kód újrafelhasználhatóságát az egységes hibakezelési mechanizmusokkal és a strukturált JSON válaszokkal növeltem. Például minden vizsgálati eszköz JSON formátumban adja vissza az eredményeit, amelyeket a kliens oldalon könnyen feldolgozhatunk. Az *app.py* fájlban található endpointok, mint a */scan*, */generate_report*, */download_raw_zip* és */download_log*, világosan elkülönítik az egyes funkciókat, így az alkalmazás könnyen bővíthető új eszközökkel vagy funkciókkal.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

```
@app.route('/scan', methods=['POST'])
def scan():
    data = request.get_json()
    url = data.get("url")
    mode = data.get("mode", "static")

    if not url:
        return jsonify({"error": "Nem adtál meg URL-t!"}), 400

    headers = {'User-Agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0'}
    headers_str = headers['User-Agent']

    # HTML letöltés
    html_result = download_html(url, headers)
    if not html_result.get("success"):
        return jsonify({"error": html_result.get("error", "Ismeretlen hiba")}), 500

    response_headers = html_result["headers"]
    response_body = html_result["body"]
    full_content = str(response_headers) + "\n\n" + response_body

    log_debug_message(f"[{datetime.now().strftime('%Y-%m-%d %H:%M')}] URL: {url} átadásra került a {mode.upper()} módban")

    # Eredmények
    scan_results = {}

    if mode == "static":
        scan_results["semgrep"] = run_semgrep(full_content)
    elif mode == "dynamic":
        scan_results["nikto"] = run_nikto(url)
        scan_results["arachni"] = run_arachni(url, headers_str)
        scan_results["wapiti"] = run_wapiti(url)
    elif mode == "llm":
        scan_results["llm"] = run_llm(response_headers)
    elif mode == "all":
        scan_results["semgrep"] = run_semgrep(full_content)
        scan_results["nikto"] = run_nikto(url)
        scan_results["arachni"] = run_arachni(url, headers_str)
        scan_results["wapiti"] = run_wapiti(url)
        scan_results["llm"] = run_llm(response_headers)
    else:
        return jsonify({"error": f"Ismeretlen mód: {mode}"}), 400

    combined_results = {
        "url": url,
        "scan_results": scan_results
    }

    log_debug_message(f"[{datetime.now().strftime('%Y-%m-%d %H:%M')}] Teszteredmények sikeresen kombinálva.")
    return jsonify(combined_results)
```

4.1. ábra – A scan függvény az app.py-on belül

4.2. Frontend: index.html és stíluslap.css

A frontend az index.html fájlban található, amely egy egyszerű, de funkcionális felhasználói felületet biztosít. A felület lehetővé teszi az URL megadását, a vizsgálati mód kiválasztását (statikus, dinamikus, LLM, vagy összes eszköz), valamint a vizsgálat indítását, jelentés generálását, nyers kimenet letöltését és a naplófájl letöltését. A JavaScript kód az aszinkron kérések kezelésére épül, a fetch API használatával kommunikál a backenddel. Az eredmények megjelenítése JSON formátumban történik, amelyet a böngészőben formázott szöveggént (<pre> taggel) jelenítünk meg, így könnyen olvasható.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

A kinézetet a stiluslap.css fájl definiálja, amely egy letisztult, reszponzív dizájnt biztosít. A CSS-ben egyszerű színpalettát használtam (#EEEEEE háttér, #007BFF gombok), és az Arial betűtípussal garantáltam a platformfüggetlen olvashatóságot. A reszponzív dizájn érdekében a container osztály maximális szélességét 800 pixelre korlátoztam, így a felület kisebb eszközökön is jól használható.

```
<!DOCTYPE html>
<html lang="hu">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Biztonsági Vizsgálati Eszköz</title>
  <link rel="shortcut icon" href="static/favicon.ico" />
  <link rel="stylesheet" type="text/css" href="static/stiluslap.css">
</head>
<body>
  <div class="container">
    <h1>Biztonsági Vizsgálati Eszköz</h1>

    <div class="input-group">
      <label for="urlInput">Adjon meg egy URL-t a vizsgálatához:</label>
      <input type="text" id="urlInput" placeholder="https://pelda.com">
    </div>

    <div class="input-group">
      <label for="modeSelect">Válasszon vizsgálati módot:</label>
      <select id="modeSelect">
        <option value="static">Statikus</option>
        <option value="dynamic">Dinamikus</option>
        <option value="llm">LLM</option>
        <option value="all">Összes Eszköz</option>
      </select>
    </div>

    <button onclick="scanWebsite()">Vizsgálat Indítása</button>
    <button onclick="generateReport()">Jelentés Generálása</button>
    <button onclick="downloadLog()">Log Letöltése</button>
    <button onclick="downloadRawOutput()">Nyers Output Letöltése</button>

    <div id="results" style="margin-top: 20px;"></div>
  </div>
```

4.2. ábra – Részlet az index.html tartalmából a JavaScript-tel megvalósított rész nélkül

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

```
async function scanWebsite() {
  const urlInput = document.getElementById('urlInput');
  const modeSelect = document.getElementById('modeSelect');
  const resultsDiv = document.getElementById('results');
  const url = urlInput.value;
  const mode = modeSelect.value;

  resultsDiv.innerHTML = "Vizsgálat...";

  try {
    const response = await fetch('/scan', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ url: url, mode: mode })
    });

    if (!response.ok) {
      const errorData = await response.json();
      resultsDiv.innerHTML = `<pre>Hiba: ${JSON.stringify(errorData, null, 2)}</pre>`;
      return;
    }

    const data = await response.json();

    let html = '<h3>Vizsgálati Eredmények</h3>';
    if (data.scan_results) {
      for (const [tool, result] of Object.entries(data.scan_results)) {
        html += `<h4>${tool.toUpperCase()}</h4>`;
        html += `<pre>${JSON.stringify(result, null, 2).replace(/\n/g, "\n")}</pre>`;
      }
    }

    resultsDiv.innerHTML = html;

    localStorage.setItem('scanResults', JSON.stringify(data));
  } catch (error) {
    resultsDiv.innerHTML = `<pre>Hiba: ${error.message}</pre>`;
  }
}
```

4.3. ábra – Példa JavaScript kód az aszinkron kérések kezelésére az index.html-ben

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

4.3. Biztonsági vizsgálati eszközök integrálása

Az alkalmazás lényege a különböző biztonsági vizsgálati eszközök integrálása. Az eszközök futtatása a megadott vizsgálati mód alapján történik:

- **Statikus:** A `run_semgrep` függvény a Semgrep eszközt használja a letöltött HTML válaszkód statikus elemzésére. A kódot először egy `response.html` fájlba mentjük, majd a Semgrep JSON formátumban adja vissza az eredményeket.
- **Dinamikus:** Három eszközt integráltam: Nikto, Arachni és Wapiti. Mindegyik eszköz parancssori interfészen keresztül fut, és a `subprocess.run` segítségével vezéreljük őket. A Nikto és Wapiti JSON kimenetet generál, míg az Arachni egyéni riportfájlt (`.afr`) hoz létre, amelyet később törlünk a tárhely felszabadítása érdekében.
- **LLM:** Az LLM mód a DeepSeek API-t használja a HTTP fejlécek elemzésére, például a Strict-Transport-Security (HSTS) vagy a web cache poisoning vektorok ellenőrzésére. Az API hívások hibakezelése alapos, minden hálózati hibát naplózunk.
- **Összes eszköz:** Ez a mód az összes fent említett eszközt lefuttatja, és kombinálja az eredményeket.

Az eszközök kimeneteit egységes JSON struktúrába rendeztem, amely lehetővé teszi a későbbi elemzést és a jelentés generálását. A naplózási rendszer minden mód futtatásakor rögzíti a kulcsfontosságú eseményeket, így könnyen nyomon követhető a vizsgálat folyamata.

4.4. Jelentés generálás

A jelentés generálása a `/generate_report` endpointon keresztül történik, amely egy Word dokumentumot (`.docx`) hoz létre a `python-docx` könyvtár segítségével. A jelentés a vizsgálati eredmények alapján összesíti a talált sebezhetőségeket, és azok előfordulási gyakoriságát a Counter és defaultdict osztályok használatával számolja. A sebezhetőségeket az előfordulások száma alapján csökkenő sorrendbe rendezem, és minden sebezhetőséghez megadom, hogy mely eszközök detektálták. A jelentés letölthető formátumban kerül a klienshez, így könnyen megosztható.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

4.5. Nyers kimenet és napló letöltése

A nyers kimenet letöltése a /download_raw_zip endpointon keresztül valósul meg, amely egy ZIP fájlba csomagolja az összes eszköz JSON kimenetét. A ZIP fájl generálása a zipfile modulval történik, és a fájlokat memóriában kezeljük (io.BytesIO), így nincs szükség fizikai fájlok létrehozására.

A napló letöltése a /download_log endpointon keresztül érhető el, amely a debug.log fájlt küldi vissza. Mindkét funkció alapos hibakezeléssel rendelkezik, például ellenőrzi, hogy létezik-e a letöltendő fájl.

```
@app.route('/download_raw_zip', methods=['POST'])
def download_raw_zip():
    try:
        data = request.get_json()
        if not data:
            return jsonify({"hiba": "Nincs beérkező adat a ZIP generálásához"}), 400

        scan_results = data.get("scan_results", {})
        if not scan_results:
            return jsonify({"hiba": "Nem találhatóak vizsgálati eredmények"}), 400

        memory_file = io.BytesIO()
        with zipfile.ZipFile(memory_file, 'w', zipfile.ZIP_DEFLATED) as zf:
            for tool, result in scan_results.items():
                if not result:
                    continue # Üres eredményt nem mentünk el

                filename = f"{tool}.json"

                # JSON formázás grep-barát módon
                if isinstance(result, (dict, list)):
                    content = json.dumps(result, indent=4, ensure_ascii=False).replace("\\n", "\n")
                else:
                    content = str(result).replace("\\n", "\n") # Ha a kimenetben \n van, akkor az valódi új sor legyen

                zf.writestr(filename, content)

            if not zf.filelist: # Ha nincs mit menteni, hibaüzenetet adunk
                return jsonify({"hiba": "A nyers output fájl üres, nincs mit letölteni!"}), 400

            memory_file.seek(0)
            return send_file(memory_file, download_name="raw_outputs.zip", as_attachment=True)

    except Exception as e:
        return jsonify({"hiba": f"Hiba a ZIP generálása közben: {str(e)}"}), 500
```

4.6. ábra – A download_raw_zip() függvény

4.6. Hibakezelés és naplózás

A hibakezelés az alkalmazás egyik kulcsfontosságú eleme. Minden endpoint és függvény try-except blokkokkal van körülvéve, hogy a váratlan hibák ne okozzanak összeomlást. A hibák részletesen naplózásra kerülnek a debug.log fájlba, beleértve az időbélyeget, a hiba típusát és

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

a kontextust. Például a `get_deepseek_valasz` függvény minden HTTP kérést naplóz, és a válaszokat JSON formátumban menti el a hibakeresés megkönnyítése érdekében.

Ez a megközelítés lehetővé tette a fejlesztés során felmerülő problémák gyors azonosítását és megoldását.

```
DEBUG = True

def initialize_log():
    with open("debug.log", "w") as log_file:
        pass

def log_debug_message(message):
    if DEBUG:
        with open("debug.log", "a") as log_file:
            log_file.write(message + "\n")
> #region---
    log_debug_message(f"[{datetime.now().strftime('%Y-%m-%d %H:%M')}]] URL: {url} átadásra került a {mode.upper()} módban")
```

4.7. ábra – Példa a naplózási mechanizmusra

4.7. Biztonsági szempontok

Az alkalmazás fejlesztése során külön figyelmet fordítottam a biztonsági szempontokra. Az API kulcsok (például a DeepSeek API kulcs) tárolására éles környezetben környezetváltozókat használnék a hardcoded kulcsok helyett. A HTTP kérésekhez egyedi User-Agent fejléccet állítottam be, hogy az eszközök ne legyenek letiltva a célzott weboldalak által. A fejlesztés elején korlátozások nélkül is lefuttattam az eszközöket a teljes funkcionalitásuk megismerése érdekében, azonban a további futtatások során a vizsgálatok mélységét és időtartamát szándékosan korlátoztam, mivel a vizsgálatok célja csupán demonstrációs jellegű volt.

```
# Nikto parancs beállítása
nikto_command = [
    'nikto',
    '-h', url,
    '-maxtime', '90',
    '-Tuning', '1,2,3',
    '-nointeractive'
]
```

4.8. ábra – A `run_nikto()` függvény azon része, ahol a nikto-t kapcsolókkal korlátoztam

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

4.8. Optimalizálás

A fejlesztés során folyamatosan teszteltem az alkalmazást különböző URL-ekkel és vizsgálati módokkal. A tesztek során azonosítottam és javítottam több hibát, például a Semgrep JSON kimenetének dekódolási problémáit vagy az Arachni riportfájlok helytelen kezelését. Az optimalizálás részeként csökkentettem a redundáns fájlmentéseket (például az Arachni riportok törlése a vizsgálat után) és a HTTP kérések időtúllépési értékét 10 másodpercre korlátoztam a DeepSeek API hívásoknál.

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

5. TESZTELÉS

A teszteléssel célunk megtalálni és megelőzni a hibákat, valamint növelni a rendszer megbízhatóságát.

5.1. A tesztelés folyamata

A fejlesztési folyamat során követtem a szoftvertesztelés alapelveit, különös figyelmet fordítva a tesztelés „aranyszabályaira”. Minden funkció elkészülte után különféle tesztadatokkal ellenőriztem annak működését. A feltárt hibákat azonnal javítottam, ezáltal a későbbi tesztelési fázisokat is gördülékenyebbé tettem. A rendszert strukturált módon, szisztematikus teszteléssel vizsgáltam végig, amely biztosította az egyes funkciók megbízható működését. Külön hangsúlyt fektettem az extrém vagy nem várt bemeneti értékekkel történő tesztelésre is, hogy feltárjam a rendszer esetleges gyenge pontjait. A specifikációban meghatározott funkciókat mind érvényes („jó”) bemenetekkel, mind szándékosan hibás („rossz”) adatokkal teszteltem, hogy biztosítsam a helyes működés mellett a hibakezelés megfelelőségét is. [21]

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

5.2. Tesztesetek

1. Teszteset: Alapvető URL vizsgálat statikus módban

- **Teszt tárgya:** URL helyességének ellenőrzése és statikus vizsgálat futtatása
- **Leírás:** Különböző formátumú URL-ekkel történő tesztelés statikus vizsgálati módban
- **Feladat:**

1. lépés: Érvényes URL megadása HTTP protokollal

Tesztadatok: http://example.com

Elvárt eredmény: Sikeres vizsgálat, Semgrep eredmények megjelenítése

Kapott eredmény: {"url": "http://example.com", "scan_results": {"semgrep": {...}}} – Sikeres lépés

2. lépés: Érvényes URL megadása HTTPS protokollal

Tesztadatok: https://example.com

Elvárt eredmény: Sikeres vizsgálat, Semgrep eredmények megjelenítése

Kapott eredmény: {"url": "https://example.com", "scan_results": {"semgrep": {...}}} – Sikeres lépés

3. lépés: Hiányzó URL esetén

Tesztadatok: Üres mező

Elvárt eredmény: Hibaüzenet: "Nem adtál meg URL-t!"

Kapott eredmény: {"hiba": "Nem adtál meg URL-t!"} – Sikeres lépés

4. lépés: Érvénytelen URL formátum

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

Tesztadatok: example.com (hiányzó protokoll)

Elvárt eredmény: Hibaüzenet a letöltés során

Kapott eredmény: {"error": "Hiba történt a letöltés során: ..."} – Sikeres lépés

2. Teszteset: Dinamikus vizsgálat funkciók tesztelése

- **Teszt tárgya:** Dinamikus vizsgálati mód működésének ellenőrzése

- **Leírás:** Dinamikus vizsgálat futtatása és eszközök eredményeinek ellenőrzése

- **Feladat:**

1. lépés: Dinamikus vizsgálat indítása érvényes URL-lel

Tesztadatok: https://example.com, mód: Dinamikus

Elvárt eredmény: Nikto, Arachni és Wapiti eredmények megjelenítése

Kapott eredmény: A Nikto, Arachni és Wapiti nyers kimenetei megjelennek a weblapon - Sikeres lépés

2. lépés: Nem elérhető weboldal vizsgálata

Tesztadatok: https://nemletezooldal123456.com, mód: Dinamikus

Elvárt eredmény: Hibaüzenet a letöltés során

Kapott eredmény: {"error": "Hiba a letöltés során: HTTPSConnectionPool

(host='nemletezooldal123456.com', port=443): Max retries exceeded with url: / (Caused by NameResolutionError("\": Failed to resolve 'nemletezooldal123456.com' ([Errno -2] Name or service not known)\"))

"} - Sikeres lépés

3. lépés: Dinamikus vizsgálat helytelen URL-lel

Tesztadatok: ftp://example.com, mód: Dinamikus

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

Elvárt eredmény: Hibaüzenet az érvénytelen URL miatt

Kapott eredmény: {"hiba": "Hiba történt a letöltés során: No connection adapters were found for 'ftp://example.com'"} – Sikeres lépés

3. Teszteset: LLM vizsgálat funkció tesztelése

- **Teszt tárgya:** LLM mód működésének ellenőrzése

- **Leírás:** LLM vizsgálat futtatása és eredmények értelmezése

- **Feladat:**

1. lépés: LLM vizsgálat indítása érvényes URL-lel

Tesztadatok: https://example.com, mód: LLM

Elvárt eredmény: DeepSeek API válaszáinak megjelenítése

Kapott eredmény: {"headers_analyzed": {...}, "llm_response": "...",
"message": "LLM analysis completed", ...}} – Sikeres lépés

2. lépés: LLM vizsgálat hibás API kulccsal

Tesztadatok: https://example.com, mód: llm (API_KEY üres)

Elvárt eredmény: Hibaüzenet az API hívás során

Kapott eredmény: {"headers_analyzed": {...},
"llm_response": "Hiba: Nincs válasz, 401 Client Error: Unauthorized for url:
https://openrouter.ai/api/v1/chat/completions", "message": "LLM analysis completed"} -
Sikeres lépés

4. Teszteset: Jelentés generálás funkció tesztelése

- **Teszt tárgya:** Word dokumentum jelentés létrehozásának ellenőrzése

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

- **Leírás:** Különböző vizsgálati eredmények alapján jelentés generálása

- **Feladat:**

1. lépés: Jelentés generálása vizsgálat nélkül

Tesztadatok: Nincs előző vizsgálat

Elvárt eredmény: {"hiba": "Nincs adat megadva a jelentésgeneráláshoz"}

Kapott eredmény: {"hiba": "Nincs adat megadva a jelentésgeneráláshoz"} - Sikeres lépés

2. lépés: Jelentés generálása sikeres vizsgálat után

Tesztadatok: Előzőleg sikeres vizsgálat eredményei

Elvárt eredmény: scan_report.docx sikeresen letöltődik

Kapott eredmény: scan_report.docx sikeresen letöltődik - Sikeres lépés

3. lépés: Jelentés tartalmának ellenőrzése

Tesztadatok: Ismert sebezhetőségeket tartalmazó vizsgálat

Elvárt eredmény: A dokumentum tartalmazza az észlelt sebezhetőségeket

Kapott eredmény: A DOCX fájl tartalmazza a megfelelő fejezeteket - Sikeres lépés

5. Teszteset: Nyers output letöltés funkció tesztelése

- **Teszt tárgya:** Nyers vizsgálati eredmények ZIP formátumban történő letöltése

- **Leírás:** Különböző esetekben történő nyers adat letöltés

- **Feladat:**

1. lépés: Nyers output letöltése vizsgálat nélkül

Tesztadatok: Nincs előző vizsgálat

Elvárt eredmény: {"hiba": "Nem találhatóak vizsgálati eredmények"}

Kapott eredmény: {"hiba": "Nem találhatóak vizsgálati eredmények"} - Sikeres lépés

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

2. lépés: Nyers output letöltése sikeres vizsgálat után

Tesztadatok: Előzőleg sikeres vizsgálat eredményei

Elvárt eredmény: raw_outputs.zip sikeresen letöltődik

Kapott eredmény: raw_outputs.zip sikeresen letöltődik - Sikeres lépés

3. lépés: ZIP tartalmának ellenőrzése

Tesztadatok: Ismert eszközöket tartalmazó vizsgálat

Elvárt eredmény: A ZIP csak a JSON fájlokat tartalmazza az egyes eszközök kimeneteivel

Kapott eredmény: A ZIP csak a JSON fájlokat tartalmazza az egyes eszközök kimeneteivel
- Sikeres lépés

6. Teszteset: Log fájl letöltés funkció tesztelése

- **Teszt tárgya:** Debug log fájl letöltésének ellenőrzése

- **Leírás:** Log fájl létezésének és tartalmának ellenőrzése

- **Feladat:**

1. lépés: Log letöltése üres loggal

Tesztadatok: Üres debug.log

Elvárt eredmény: Üres debug.log letöltődik

Kapott eredmény: Üres debug.log letöltődik - Sikeres lépés

2. lépés: Log letöltése tartalommal

Tesztadatok: Több vizsgálat után

Elvárt eredmény: A log fájl tartalmazza a vizsgálati eseményeket

Kapott eredmény: A log fájl tartalmazza a vizsgálati eseményeket - Sikeres lépés

3. lépés: Log letöltése hiányzó fájl esetén

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

Tesztadatok: Törölt debug.log

Elvárt eredmény: Hibaüzenet: "Nincs elérhető log fájl!"

Kapott eredmény: {"hiba": "Nincs elérhető log fájl!"} - Sikeres lépés

7. Teszteset: "Összes Eszköz" mód tesztelése

- **Teszt tárgya:** Komplex vizsgálat minden eszközzel

- **Leírás:** Az összes vizsgálati eszköz egyszerre történő futtatása

- **Feladat:**

1. lépés: Összes eszköz futtatása érvényes URL-lel

Tesztadatok: https://example.com, mód: Összes Eszköz

Elvárt eredmény: Minden eszköz nyers kimenete megjelenik a weboldalon

Kapott eredmény: Minden eszköz nyers kimenete megjelenik a weboldalon – Sikeres lépés

2. lépés: Időtúllépés vizsgálata hosszú futású oldalon

Tesztadatok: Nagy forgalmú weboldal, mód: Összes Eszköz

Elvárt eredmény: Egyes eszközök hibát jelezhetnek időtúllépés miatt

Kapott eredmény: Egyes eszközöknél {"Timeout was reached"} – Sikeres lépés

3. lépés: Nem támogatott protokoll vizsgálata

Tesztadatok: ftp://example.com, mód: Összes Eszköz

Elvárt eredmény: A dinamikus eszközök hibát jeleznek

Kapott eredmény: A dinamikus eszközök hibát jeleznek – Sikeres lépés

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

6. BŐVÍTÉSI ÖTLETEK

- **Új eszközök integrálása:** Lehetőség van arra, hogy a rendszerbe új eszközök vagy szkriptek legyenek hozzáadva (programozva), akár saját fejlesztésűek is.
- **Vizsgálati profilok:** A vizsgálatok hatékonyságának optimalizálása érdekében bevezetésre kerülhetnének a vizsgálati profilok funkció, amely lehetőséget biztosítana arra, hogy a felhasználók az egyes eszközök erősségeit és beállításait testre szabják. Ennek segítségével a felhasználók saját profilokat hozhatnának létre, amelyeket később újra alkalmazhatnak a vizsgálatok során. Ezáltal minden egyes eszköz konfigurálható lenne a felhasználó igényei szerint, biztosítva a vizsgálatok hatékonyságát és testreszabhatóságát.
- **Saját LLM:** Saját LLM megvalósítása bővített képességekkel, amely alkalmazható lenne az LLM-alapú vizsgálatokhoz. Az interneten saját LLM megvalósítására fellelhető útmutató. [22]

ÖSSZEFOGLALÁS

A tervezés és implementáció során figyelembe vettem a kiberbiztonsági szakértői közösség tapasztalatait, hogy az alkalmazás a valós igényekhez igazodjon, és a lehető legpraktikusabb legyen a sérülékenységek detektálásában.

Tapasztalataim szerint a biztonsági szakemberek olyan eszközt részesítenek előnyben, amely átfogó, mégis egyszerűen használható felületet nyújt, és különböző elemzési módokat egyesít. Ennek megfelelően az alkalmazás Python/Flask alapú, moduláris felépítésű. Kiemelt figyelmet fordítottam a tesztelésre, hogy az alkalmazás robusztus legyen. A tesztek során szisztematikusan ellenőriztem az egyes komponenseket, majd valós weboldalak vizsgálatával validáltam az eredményeket, figyelembe véve a közösség tapasztalatait is.

Az implementáció során nagy hangsúlyt fektettem a kód újrafelhasználhatóságára és fenntarthatóságára. A modularitás érdekében külön függvények kezelik az egyes eszközök futtatását, és egységes JSON struktúrába rendeztem a kimeneteket úgy, hogy azok tartalmi könnyedén kereshetőek legyenek grep-pel a sérülékenységtesztelők számára további analízishez. A naplózási rendszer bevezetésével biztosítottam a hibakeresés hatékonyságát. A felhasználói felület letisztult, reszponzív és intuitív, lehetővé téve az eredmények gyors áttekintését és a jelentések generálását DOCX formátumban, vagy az eszközök nyers eredményeinek letöltését JSON formátumban egy ZIP fájlban.

A jövőben igény szerint tervezem az alkalmazás továbbfejlesztését, például új biztonsági eszközök integrálásával, vizsgálati profilokkal, és az LLM-alapú elemzések tovább fejlesztésével, hogy még pontosabb és átfogóbb megoldást nyújtsak a webalkalmazások sebezhetőségvizsgálatához, ezzel támogatva a kiberbiztonsági szakembereket a sérülékenységek felfedezésében.

IRODALOMJEGYZÉK

- [1] <https://portswigger.net/web-security/llm-attacks>
- [2] <https://portswigger.net/research/request-smuggling>
- [3] <https://portswigger.net/research/gotta-cache-em-all>
- [4] https://hu.wikipedia.org/wiki/Python_%28programoz%C3%A1si_nyelv%29#
- [5] <https://hu.wikipedia.org/wiki/HTML>
- [6] <https://hu.wikipedia.org/wiki/CSS>
- [7] <https://prog.hu/cikkek/100430/bevezetes-a-css-alapjaiba>
- [8] <https://web.dev/learn/css/the-cascade/>
- [9] https://webprogramozas.inf.elte.hu/tananyag/wf2/lecke3_lap1.html
- [10] <https://en.wikipedia.org/wiki/JavaScript>
- [11] <https://github.com/pallets/flask/blob/main/README.md>
- [12] <https://flask.palletsprojects.com/en/stable/>
- [13] <https://github.com/semgrep/semgrep/blob/develop/README.md>
- [14] <https://appsec.guide/docs/static-analysis/semgrep/>
- [15] <https://www.cirt.net/Nikto2>
- [16] <https://github.com/Arachni/arachni/blob/master/README.md>
- [17] <https://github.com/wapiti-scanner/wapiti/blob/master/README.rst>
- [18] <https://github.com/deepseek-ai/DeepSeek-V3/blob/main/README.md>
- [19] <https://adasci.org/deepseek-v3-explained-optimizing-efficiency-and-scale/>
- [20] <https://zilliz.com/blog/why-deepseek-v3-is-taking-the-ai-world-by-storm>
- [21] Dr. Beszédes Árpád – Szoftvertesztelés alapjai előadás, 2019-2020
- [22] https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/chatbot_tutorial.ipynb

NYILATKOZAT

Alulírott Felegyi Mihály Patrik üzemmérnök-informatikus BProf szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, üzemmérnök-informatikus BProf diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

Szeged, 2025. május 20.

Aláírás

Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

KÖSZÖNETNYILVÁNÍTÁS

Ezúton szeretném köszönetemet és hálámat kifejezni mindenkinek, aki bármilyen módon hozzájárult szakdolgozatom elkészítéséhez.

Külön köszönet illeti témavezetőmet, Dr. Hegedűs Pétert.

Hálás vagyok továbbá szakképzett barátaimnak is, akik meglátásaikkal, tanácsaikkal és építő jellegű visszajelzéseikkel értékes segítséget nyújtottak a munkámhoz.

Köszönetemet fejezem ki továbbá a kiberbiztonsági közösségnek.

MELLÉKLETEK

1. A stiluslap.css tartalma

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #EEEEEE;
}
.container {
  max-width: 800px;
  margin: 0 auto;
}
h1 {
  text-align: center;
}
.input-group {
  margin-bottom: 15px;
}
label {
  display: block;
  margin-bottom: 5px;
}
input[type="text"] {
  width: 100%;
  padding: 8px;
  box-sizing: border-box;
}
button {
  padding: 10px 15px;
  background-color: #007BFF;
  color: white;
  border: none;
  cursor: pointer;
}
button:hover {
  background-color: #0056b3;
}
pre {
  background-color: #DDDDDD;
  padding: 15px;
  overflow-x: auto;
}
```


Sérülékenység detekció statikus és dinamikus elemzés, valamint nagy nyelvi modellek segítségével

2. A generate_report() függvény

```
@app.route('/generate_report', methods=['POST'])
def generate_report():
    try:
        data = request.get_json()
        if not data:
            return jsonify({"hiba": "Nincs adat megadva a jelentésgeneráláshoz"}), 400

        url = data.get("url", "N/A")
        scan_results = data.get("scan_results", {})

        # Sérülékenységek számlálása és eszközök követése
        vuln_counter = Counter() # Előfordulások száma
        vuln_tools = defaultdict(set) # Eszközök halmaza sérülékenységenként

        for tool, results in scan_results.items():
            #region semgrep--
            #region nikto
            elif tool == "nikto" and "részletek" in results:
                details = results["részletek"]
                #region nikto occurrences--
                if hsts_occurrences > 0:
                    vuln_counter["HTTP Strict Transport Security (HSTS)"] += hsts_occurrences
                    vuln_tools["HTTP Strict Transport Security (HSTS)"].add(tool)
                if mimetype_occurrences > 0:
                    vuln_counter["MIME Type Confusion"] += mimetype_occurrences
                    vuln_tools["MIME Type Confusion"].add(tool)
                if csp_occurrences > 0:
                    vuln_counter["Content Security Policy Misconfiguration"] += csp_occurrences
                    vuln_tools["Content Security Policy Misconfiguration"].add(tool)
                if "Server: No banner retrieved" not in details:
                    vuln_counter["Information Disclosure (Server Banner)"] += 1
                    vuln_tools["Information Disclosure (Server Banner)"].add(tool)
                if no_secure_flag_occurrences > 0:
                    vuln_counter["Session Cookie Without the secure Flag Set"] += no_secure_flag_occurrences
                    vuln_tools["Session Cookie Without the secure Flag Set"].add(tool)
                if no_httponly_flag_occurrences > 0:
                    vuln_counter["Session Cookie Without the httponly Flag Set"] += no_httponly_flag_occurrences
                    vuln_tools["Session Cookie Without the httponly Flag Set"].add(tool)
            #endregion
            #region arachni--
            #region wapiti--
            #region llm--

        # Sérülékenységek csökkenő sorrendbe rendezése az előfordulások száma alapján
        sorted_vulns = sorted(vuln_counter.items(), key=lambda x: x[1], reverse=True)

        # Riport generálása
        doc = Document()
        doc.add_heading('Biztonsági Vizsgálati Jelentés', level=1)
        doc.add_paragraph(f"Vizsgált URL: {url}")

        for vuln, count in sorted_vulns:
            if count > 0:
                tool_count = len(vuln_tools[vuln])
                doc.add_heading(f"{vuln} ({count})", level=2)
                doc.add_paragraph(f"Detektálva {tool_count} eszköz által.")

        # Jelentés mentése
        report_file = "vizsgalat_report.docx"
        doc.save(report_file)

        return send_file(report_file, as_attachment=True)

    except Exception as e:
        return jsonify({"hiba": f"Hiba a jelentés generálása során: {str(e)}"}), 500
```