Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту
інформації
_____ Іван ПАРХОМЕНКО
«___» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: «Методи захисту вебдодатків на Java»

Виконавець: студент IV курсу, групи КБ-41
_____ **Олег МІШНЬОВ**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я, прізвище**
Керівник		Олександр ЛАПТЄВ
Нормоконтроль		Сергій ДАКОВ

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-41** (група) **Мішньову Олегу Володимировичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Методи захисту вебдодатків на Java

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Засоби та методи забезпечення безпеки вебдодатків

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно провести аналіз методів та методик розробки вебдодатків на Java, дослідити сучасні підходи до їх захисту від кіберзлочинів, вивчити вразливості та механізми протидії (такі як SQL-ін'єкції, XSS, CSRF),

а також розробити практичні рекомендації щодо підвищення рівня безпеки Java-додатків, включаючи використання фреймворків (наприклад, Spring Security) та оптимізацію програмного коду.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність створення демонстраційного вебдодатка, який ілюструє як вразливі, так і захищені сценарії обробки запитів. Також розроблено рекомендації щодо впровадження дієвих механізмів безпеки в Java-додатках з урахуванням актуальних вимог до захисту даних.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав _____ (підпис) Олександр ЛАПТЄВ (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Олег МІШНЬОВ (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 22.01.2025	*виконано*
2	Аналіз літератури	29.01.2025 – 11.02.2025	*виконано*
3	Обґрунтування вибору рішення	12.02.2025 – 15.02.2025	*виконано*
4	Концепція хмарних обчислень	16.02.2025 – 04.03.2025	*виконано*
5	Аналіз сучасного стану захисту вебодатків	05.03.2025 – 21.03.2025	*виконано*
6	Дослідження вразливостей та загроз	22.03.2025 – 08.04.2025	*виконано*
7	Вироблення рекомендацій щодо рекомендацій щодо підвищення вебдодатків на Java	09.04.2025 – 10.05.2025	*виконано*
8	Оформлення пояснювальної записки	11.05.2025 – 27.05.2025	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	3.05.2025 – 13.06.2025	*виконано*

Завдання видав _____ (підпис) Олександр ЛАПТЄВ (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Олег МІШНЬОВ (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

УДК 004.056.55:004.738.5

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел і додатків. Робота містить 56 сторінки основного тексту, 2 таблиці та 30 рисунків. Список використаних джерел налічує 46 найменувань і займає 5 сторінок.

Об'єктом дослідження є процес захисту вебдодатків на Java.

Предмет дослідження - методи захисту вебдодатків на Java.

Мета роботи - дослідження методів захисту вебдодатків та видача рекомендації щодо їх використання.

Для досягнення зазначеної мети кваліфікаційної роботи поставлено наступні завдання:

- Провести системний аналіз літературних джерел та відкритих стандартів щодо захисту вебдодатків Java, включаючи найпоширеніші загрози та відповідні захисні практики.
- Дослідити сучасні підходи до реалізації захисту у вебдодатках на Java з використанням таких фреймворків як Spring Boot, Spring Security, а також засобів перевірки вхідних даних і управління доступом.
- Розробити тестовий вебдодаток, що демонструє реалізацію як вразливих, так і захищених сценаріїв, з метою експериментального аналізу ефективності обраних механізмів безпеки.

На основі проведених експериментів сформувати практичні рекомендації щодо впровадження найбільш ефективних методів захисту вебдодатків на Java для зниження ризиків компрометації.

Методи дослідження кваліфікаційної роботи:

- Аналіз, синтез;
- Експеримент, моделювання;

- компаративний аналіз.

Практична цінність роботи полягає у створенні демонстраційного вебдодатка, який ілюструє як вразливі, так і захищені сценарії обробки запитів. Також розроблено рекомендації щодо впровадження дієвих механізмів безпеки в Java-додатках з урахуванням актуальних вимог до захисту даних.

Ключові слова: вебдодатки, Java, автентифікація, SQL-ін’єкція, XSS, безпека даних, OWASP, Spring Security, кіберзагрози, кібербезпека.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

IDE	–	Integrated Development Environment
UI	–	User Interface
CLI	–	Command-Line Interface
SQL	–	Structured Query Language
XSS	–	Cross-Site Scripting
MD5	–	Message Digest 5
ПЗ	–	Програмне забезпечення
ШПЗ	–	Шкідливе програмне забезпечення
БД	–	База даних
NCSI	–	National Cyber Security Index
NSA	–	National Security Agency
OWASP	–	Open Web Application Security Project
VPN	–	Virtual Private Network
XSS	–	Cross-Site Scripting
IP	–	Internet Protocol
IT	–	Information Technology
JS	–	JavaScript
MFA	–	Multi-Factor Authentication
RFI	–	Remote File Inclusion
SQLi	–	SQL Injection

ВСТУП

З початку XXI століття стрімкий розвиток інформаційних технологій призвів до зростання залежності суспільства від цифрових сервісів, зокрема вебдодатків, що забезпечують доступ до широкого спектру послуг — від фінансових операцій до комунікації. У зв’язку з цим актуальність проблеми забезпечення безпеки вебзастосунків набуває особливого значення, оскільки кібератаки стають дедалі складнішими, а кількість виявлених уразливостей — постійно зростає. За даними звітів OWASP, найбільш поширеними загрозами залишаються ін’єкції, міжсайтовий скриптинг, слабка автентифікація та обробка небезпечного контенту.

Сучасні вебдодатки побудовані за принципами сервісно-орієнтованої архітектури, що передбачає високу інтегрованість різноманітних компонентів — від баз даних до механізмів автентифікації. Проте саме ця інтеграція відкриває численні вектори для атак, які можуть призвести до несанкціонованого доступу, викрадення конфіденційної інформації або повного порушення роботи системи. Таким чином, виникає потреба не лише в теоретичному дослідженні вразливостей, але й у практичному моделюванні сценаріїв атак із подальшою розробкою ефективних методів захисту.

Актуальність цієї роботи полягає у створенні навчального тестового вебдодатку, який реалізує приклади як уразливого, так і захищеного функціоналу для чотирьох типових кіберзагроз: Cross-Site Scripting (XSS), SQL-ін’єкцій, обробки небезпечних файлів (з інтеграцією сигнатурного аналізу YARA) та слабкої автентифікації. Такий підхід дозволяє порівняти ефективність захисних механізмів у реальних умовах та сформувати практичні рекомендації з безпечної розробки.

Мета роботи - дослідження методів захисту вебдодатків та видача рекомендації щодо їх використання.

Для досягнення зазначеної мети кваліфікаційної роботи поставлено наступні завдання:

- Провести системний аналіз літературних джерел та відкритих стандартів щодо захисту вебдодатків Java, включаючи найпоширеніші загрози та відповідні захисні практики.
- Дослідити сучасні підходи до реалізації захисту у вебдодатках на Java з використанням таких фреймворків як Spring Boot, Spring Security, а також засобів перевірки вхідних даних і управління доступом.
- Розробити тестовий вебдодаток, що демонструє реалізацію як вразливих, так і захищених сценаріїв, з метою експериментального аналізу ефективності обраних механізмів безпеки.
- На основі проведених експериментів сформувати практичні рекомендації щодо впровадження найбільш ефективних методів захисту вебдодатків на Java для зниження ризиків компрометації.

Об’єктом дослідження є процес захисту вебдодатків на Java.

Предмет дослідження - методи захисту вебдодатків на Java.

Методи дослідження кваліфікаційної роботи:

- Аналіз, синтез;
- Експеримент, моделювання;
- компаративний аналіз.

Практична цінність роботи полягає у створенні демонстраційного вебдодатка, який ілюструє як вразливі, так і захищені сценарії обробки запитів. Також розроблено рекомендації щодо впровадження дієвих механізмів безпеки в Java-додатках з урахуванням актуальних вимог до захисту даних.

РОЗДІЛ 1
АНАЛІЗ АРХІТЕКТУРИ ВЕБДОДАТКІВ У JAVA

1.1 Загальні відомості про структуру вебдодатків Java

Вебдодатки на основі Java становлять один із найпоширеніших типів програмного забезпечення, що використовуються для створення інтерактивних і масштабованих рішень. Структура таких додатків базується на парадигмі багатошарової додатки на основі Java становлять один із найпоширеніших типів програмного забезпечення, що використовуються для створення інтерактивних і масштабованих архітектури, що забезпечує чітке розділення логіки програми, презентаційного рівня та даних. Однією з ключових переваг такої структури є її адаптивність до різних середовищ розробки та високий рівень модульності [1].

Структура вебдодатків Java є багатокомпонентною системою, що поєднує технології роботи з даними, бізнес-логікою, а також управління представленням і масштабуванням. Її адаптивність і гнучкість роблять цю платформу провідним вибором для створення сучасних вебрішень. Вебдодатки, створені на основі Java, посідають чільне місце серед сучасних програмних рішень, які активно використовуються для реалізації інтерактивних, масштабованих та розподілених сервісів. Популярність цієї платформи зумовлена її стабільністю, широкою екосистемою бібліотек і фреймворків, а також здатністю адаптуватися до різноманітних вимог ринку. Архітектура таких додатків зазвичай ґрунтується на багаторівневій моделі, яка передбачає чітке розмежування відповідальностей між різними компонентами — логікою бізнес-процесів, рівнем представлення та роботою з даними. Такий підхід сприяє досягненню високої модульності, що істотно полегшує підтримку та масштабування додатків у довгостроковій перспективі [1].

Основу класичної Java-архітектури вебдодатків становлять технології Java Servlet API та JavaServer Pages (JSP). Вони забезпечують обробку HTTP-запитів

і динамічну генерацію вмісту відповідно до потреб користувача. Зокрема, Servlet-інтерфейс є фундаментальним механізмом для обробки запитів, де розробник реалізує відповідні класи, що наслідують HttpServlet, забезпечуючи таким чином необхідну бізнес-логіку [2]. У свою чергу, JSP дозволяє гнучко поєднувати HTML-розмітку з вбудованими елементами на Java, що значно спрощує створення інтерфейсів користувача на ранніх етапах розробки.

Для підвищення ефективності управління логікою взаємодії між користувачем та системою, в архітектурі сучасних вебдодатків на Java зазвичай реалізується модель Model-View-Controller (MVC). Цей підхід передбачає логічне розділення системи на три незалежні рівні: модель (Model) відповідає за доступ до даних і бізнес-об’єктів, контролер (Controller) реалізує керування потоками запитів, а уявлення (View) забезпечує візуальне представлення інформації [3]. MVC-модель дає змогу досягти високого рівня абстракції та спрощує тестування й підтримку системи.

Робота з даними у Java-додатках, як правило, реалізується за допомогою реляційних баз даних, з якими взаємодія забезпечується через інтерфейс JDBC (Java Database Connectivity). Завдяки йому розробник може виконувати SQL-запити, керувати транзакціями та обробляти результати в єдиному стилі, незалежно від конкретної реалізації бази даних. Крім того, широке поширення мають ORM-рішення, зокрема Hibernate, які дозволяють здійснювати об’єктно-реляційне відображення. Це значно знижує складність при розробці та супроводі програм, оскільки більшість рутинних операцій із базою даних виконується автоматично [4].

З огляду на сучасні вимоги до масштабованості та гнучкості, Java-додатки все частіше проєктуються у вигляді сукупності мікросервісів. Такий підхід дозволяє реалізовувати кожен функціональний блок як окремий сервіс, який можна незалежно тестувати, розгортати та масштабувати. У реалізації подібних архітектур значну роль відіграє фреймворк Spring Boot, що забезпечує швидкий старт, конфігурацію за замовчуванням і зручну інтеграцію з Docker-контейнерами [5].

Окрему увагу в архітектурі Java-додатків приділено керуванню сесіями та безпекою. Для збереження стану користувача між HTTP-запитами застосовуються сесійні ідентифікатори, cookie-файли або токени, які можуть кешуватися на стороні сервера. Сучасні підходи до авторизації базуються на відкритих стандартах, таких як OAuth 2.0, що дозволяють інтегрувати механізми доступу до ресурсів з урахуванням сучасних вимог до конфіденційності й безпеки. Одним з провідних рішень у цій сфері є Spring Security — потужний інструмент для контролю автентифікації, авторизації та захисту вебдодатків на всіх рівнях.

З метою розширення функціональності вебдодатки Java інтегрують зовнішні інтерфейси прикладного програмування (API). Найбільш поширеними є RESTful API, які дозволяють організовувати простий обмін даними через HTTP-запити у форматі JSON або XML. Крім того, дедалі частіше використовується GraphQL — мова запитів, що дає змогу клієнту точно визначати, які саме дані йому потрібні. Це мінімізує обсяг даних, що передаються, а також знижує навантаження на мережу, що особливо важливо в мобільних або високонавантажених системах.

Масштабування вебдодатків Java зазвичай досягається шляхом розгортання додатку на кількох серверах, організованих у кластер, з використанням балансування навантаження. Сервери застосунків, як-от Apache Tomcat, Jetty або WildFly, забезпечують стабільне виконання Java-компонентів у розподіленому середовищі. У поєднанні з хмарними сервісами, такими як Amazon Web Services або Google Cloud Platform, це дозволяє гнучко масштабувати ресурси відповідно до реального навантаження [6].

1.2 Огляд найпоширеніших структур вебдодатків

Вебдодатки мають різноманітні архітектурні підходи, що впливають на їх продуктивність, гнучкість, масштабованість і інтерактивність. Основними сучасними підходами є односторінкові додатки (SPA), мікросервіси та

серверniless-архітектури. Ці підходи пропонують різні переваги та підходять для різних бізнес-вимог.

Односторінкові додатки (SPA) забезпечують динамічну взаємодію з користувачем завдяки завантаженню лише необхідного контенту, що значно зменшує час завантаження та покращує загальний досвід використання. SPA працюють таким чином, що замість перезавантаження всієї сторінки контент оновлюється у межах однієї сторінки за допомогою AJAX (Asynchronous JavaScript and XML). Це перетворює поведінку вебдодатків, роблячи їх схожими на десктопні програми [7].

Основними перевагами SPA є:

- Менше споживання трафіку завдяки завантаженню лише необхідного контенту.
- Покращений досвід користувача завдяки плавному переходу між функціональними блоками.
- Легкість інтеграції з backend через API, такі як RESTful або GraphQL.

Недоліками є підвищені вимоги до клієнтського обладнання та ускладнення індексації сторінок пошуковими системами.

Переваги Serverless:

- Мінімізація витрат на інфраструктуру.
- Висока масштабованість завдяки автоматичному управлінню ресурсами.
- Простота розгортання додатків.

Недоліки:

- Обмеження в конфігурації через залежність від провайдера.
- Потенційні затримки через "холодний старт" функцій.

Кожна з описаних архітектур має свої сильні сторони і є оптимальною для певних умов. Односторінкові додатки підходять для систем із високими вимогами до інтерактивності, мікросервіси ідеальні для великих проєктів із багатьма незалежними функціями, а Serverless дозволяє знизити витрати на інфраструктуру. Наочна порівняльна характеристика представлена у таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика популярних архітектур

Характеристика	SPA (Single Page Applications)	Мікросервіси (Microservices)	Serverless (Серверless-архітектури)
Призначення	Інтерактивні, динамічні додатки з акцентом на UX.	Масштабовані системи з незалежними компонентами.	Логіка додатків без управління інфраструктурою.
Основні технології	AJAX, RESTful, GraphQL, JavaScript.	Docker, Kubernetes, API Gateway.	AWS Lambda, Google Functions, Azure Functions.
Продуктивність	Залежить від клієнтського обладнання.	Висока за рахунок розподілу навантаження.	Залежить від хмарного провайдера.
Масштабованість	Менш ефективна у випадку складних систем.	Висока: можливість масштабування окремих сервісів.	Автоматична, залежить від хмарного провайдера.
Масштабованість	Менш ефективна у випадку складних систем.	Висока: можливість масштабування окремих сервісів.	Автоматична, залежить від хмарного провайдера.
Вартість впровадження	Середня: вимагає значної фронтенд-розробки.	Висока через потребу управління мікросервісами.	Низька завдяки мінімальним витратам на сервери.
Гнучкість у виборі технологій	Обмежена JavaScript і клієнтськими фреймворками.	Висока: кожен сервіс може використовувати свою технологію.	Залежить від можливостей хмарного провайдера.
Безпека	Складнощі через велике навантаження на клієнтську сторону.	Вимагає додаткового управління доступом до сервісів.	Управляється хмарним провайдером.
Переваги	Плавний UX, зменшення трафіку, інтерактивність.	Масштабованість, незалежність компонентів.	Зниження витрат на інфраструктуру, простота.

продовження таблиці 1.1

Недоліки	SEO-проблеми, висока залежність від клієнтської потужності.	Ускладнення управління системою.	Затримки виконання (холодний старт).

Мікросервіси представляють собою підхід, за якого програма складається з незалежних сервісів, кожен з яких виконує окрему функцію. Ці сервіси не залежать один від одного і можуть бути розроблені з використанням різних мов програмування або технологій. Такий підхід дозволяє масштабувати та оновлювати окремі компоненти без впливу на всю систему [5].

Переваги мікросервісів:

- Можливість розробки компонентів незалежними командами.
- Легке масштабування окремих сервісів залежно від навантаження.
- Гнучкість у виборі технологій.

Недоліки:

- Зростання складності управління системою.
- Висока залежність від ефективності комунікації між сервісами.

Cервepless-архітектури ґрунтуються на використанні хмарних сервісів для виконання логіки програми без необхідності управління серверною інфраструктурою. Розробники передають обробку запитів хмарному провайдеру, що спрощує підтримку та знижує витрати [6].

1.3 Відомості про технології та фреймворки вебдодатків на Java

Технології та фреймворки для розробки вебдодатків на Java надають розробникам широкий інструментарій для створення масштабованих, продуктивних і захищених програмних рішень. Постійна еволюція цієї екосистеми супроводжується розвитком самої мови Java, удосконаленням підходів до інтеграції компонентів та оптимізацією способів взаємодії з даними.

Саме завдяки цим перевагам Java й надалі зберігає позицію одного з найпопулярніших інструментів у сфері корпоративної розробки.

Основою для побудови складних і розподілених вебдодатків є платформа Java Enterprise Edition (Java EE), що нині відома під назвою Jakarta EE. Вона містить набір узгоджених специфікацій, які охоплюють критично важливі аспекти розробки, зокрема обробку HTTP-запитів (Servlet API), генерацію динамічного контенту (JSP), реалізацію бізнес-логіки (EJB), а також побудову RESTful-сервісів через JAX-RS [8]. Завдяки цій модульності Jakarta EE дозволяє створювати комплексні системи з високим ступенем повторного використання коду та незалежності компонентів.

У контексті цієї платформи важливу роль відіграє Java Servlet API, який забезпечує низькорівневу взаємодію з клієнтськими запитами через протокол HTTP. Це дозволяє точно контролювати маршрутизацію запитів і керувати життєвим циклом вебдодатка. Паралельно із сервлетами використовується JavaServer Pages (JSP), яка значно спрощує розробку динамічних вебсторінок, об’єднуючи у собі можливості HTML і Java. Такий підхід полегшує інтеграцію даних на етапі рендерингу сторінки [5].

Популярною альтернативою Jakarta EE є Spring Framework — гнучкий і широко підтримуваний фреймворк, який пропонує модульну структуру, підтримку інверсії управління (IoC) та зручну інтеграцію з іншими технологіями. Одним із його ключових компонентів є Spring MVC, що реалізує архітектурну модель Model-View-Controller. Це дозволяє логічно розділити бізнес-логіку, управління даними та інтерфейс користувача, що вкрай важливо для великих систем з великою кількістю функціональних компонентів [9].

Spring також вирізняється підтримкою декларативного стилю програмування. За допомогою анотацій, таких як @Controller, @Service, @Repository, розробник може значно спростити конфігурацію додатку, роблячи його більш прозорим та легким у підтримці. У роботі з базами даних Spring інтегрується з модулем Spring Data, що автоматизує рутинні операції доступу до даних. А для асинхронної обробки запитів у реальному часі може

використовуватися Spring WebFlux — реактивний модуль для побудови високопродуктивних систем.

Коли мова заходить про роботу з базами даних, одними з найважливіших технологій у Java-середовищі залишаються Hibernate та Java Persistence API (JPA). Hibernate, як реалізація JPA, забезпечує ефективне об’єктно-реляційне відображення (ORM), що дозволяє розробникам працювати з базами даних через Java-об’єкти, мінімізуючи необхідність написання сирого SQL-коду. Це знижує ризик помилок і пришвидшує розробку завдяки абстрагуванню низькорівневої взаємодії з базою даних [4].

Ще одним прикладом сучасного підходу до розробки вебінтерфейсів є фреймворк Vaadin. Він дозволяє створювати інтерфейси користувача повністю на Java, як на клієнтській, так і на серверній стороні. Це особливо цінно для команд, які вже мають експертизу в Java і не хочуть залучати окремі фронтенд-технології. Такий підхід не лише спрощує командну роботу, а й зменшує потребу у взаємодії з JavaScript, забезпечуючи при цьому сучасний рівень інтерактивності вебдодатків [10].

Загалом, розробка вебдодатків на Java спирається на широкий набір технологій і фреймворків, кожен із яких виконує важливу роль у забезпеченні продуктивності, масштабованості та зручності розробки. Використання Jakarta EE або Spring дозволяє адаптувати архітектуру до потреб корпоративного середовища, тоді як Hibernate та Vaadin сприяють ефективному управлінню даними та побудові інтуїтивно зрозумілих інтерфейсів. Вибір конкретного набору технологій, як правило, визначається вимогами проєкту, доступними ресурсами та цілями команди.

Висновки за розділом 1

Таким чином, у розділі було проведено аналіз основних підходів до розробки вебдодатків у середовищі Java, а також розглянуто ключові технології та фреймворки. У якості підсумку варто зазначити, що архітектура вебдодатків

на Java ґрунтується на використанні багатошарової моделі, яка забезпечує чітке розділення рівнів даних, бізнес-логіки та представлення.

Односторінкові додатки (SPA) виділяються інтерактивністю, оптимізацією трафіку та підвищенням зручності користувачів завдяки використанню технологій, таких як AJAX. Мікросервісна архітектура забезпечує модульність та масштабованість систем за рахунок незалежності сервісів, тоді як Serverless-архітектура дозволяє спростити управління інфраструктурою завдяки інтеграції з хмарними платформами.

Крім цього, у розділі було розглянуто такі важливі технології, як Java EE (нині Jakarta EE) та Spring Framework, що забезпечують базу для створення корпоративних додатків. Інструменти Hibernate і JPA демонструють ефективність роботи з базами даних, а сучасні фреймворки, як-от Vaadin та Micronaut, надають додаткові можливості для побудови інтерфейсів та оптимізації мікросервісів.

Також було розглянуто аспекти безпеки вебдодатків, такі як автентифікація та авторизація за допомогою Spring Security, і автоматизоване тестування, яке забезпечує стабільність і якість продукту.

Таким чином, встановлено, що Java пропонує широкий набір технологій і фреймворків, які відповідають сучасним вимогам розробки вебдодатків. Використання цих інструментів дозволяє створювати продуктивні, масштабовані та безпечні рішення, що задовольняють потреби різних типів користувачів і бізнесів.

РОЗДІЛ 2

АНАЛІЗ КЛАСИФІКАЦІЙ ВРАЗЛИВОСТЕЙ

2.1 Загальні відомості про популярні вразливості

Безпека вебдодатків є одним із найважливіших аспектів розробки, оскільки такі додатки активно взаємодіють із конфіденційними даними користувачів і стають частими цілями кіберзагроз. Вразливості вебдодатків охоплюють широкий спектр атак, які експлуатують недоліки програмного забезпечення, серверів або мережевої інфраструктури.

Вебдодатки є основою сучасної інформаційної екосистеми, проте їхня популярність робить їх привабливою ціллю для кіберзлочинців. Основні вразливості вебдодатків зазвичай пов'язані з недоліками в архітектурі, конфігурації та процесах верифікації даних. У цьому розділі розглядаються найпоширеніші вразливості, такі як XSS, SQL-ін’єкції, CSRF, а також атаки на автентифікацію та управління сесіями.

У контексті сучасних викликів кібербезпеки питання захисту вебдодатків набуває особливої актуальності, оскільки вебтехнології є одним із ключових векторів реалізації цифрових сервісів і водночас об’єктом численних атак з боку зловмисників. Автентифікація та управління сесіями є критично важливими складовими безпеки вебдодатків, оскільки вразливості в цих механізмах можуть стати основою для низки атак.

Однією з найпоширеніших загроз є Cross-Site Scripting (XSS), коли зловмисник вставляє шкідливий скрипт у вебдодаток, і цей код виконується у браузері жертви. Такі скрипти здатні викрадати файли cookie, сесійні токени або інші конфіденційні дані, зазвичай через відсутність або недостатність перевірки введених користувачем даних [11]. Іншою серйозною вразливістю є SQL-ін’єкція, яка передбачає впровадження зловмисного SQL-коду у запити до бази даних. Це дозволяє несанкціоновано отримати доступ до критичних даних, таких

як облікові записи користувачів.

Найефективнішими методами протидії є застосування параметризованих запитів і використання ORM-інструментів, що виключають можливість прямого впливу користувацького вводу на SQL-логіку [12]. Вразливості автентифікації також проявляються у вигляді Cross-Site Request Forgery (CSRF), де зловмисник змушує браузер жертви виконувати небажані запити від імені автентифікованого користувача, користуючись довірою вебдодатка до вже авторизованої сесії [13]. Ще однією небезпечною загрозою є Remote File Inclusion (RFI), яка полягає у завантаженні зовнішніх шкідливих файлів на сервер, що може призвести до виконання довільного коду та повного контролю над системою. Така атака можлива при неналежному використанні функцій включення файлів без достатньої валідації [14].

Окрему категорію становлять атаки типу Denial of Service (DoS), зокрема SYN flood, які призводять до перевантаження сервера шляхом надмірної кількості запитів. Це робить ресурс недоступним для легітимних користувачів і є серйозною загрозою для стабільності сервісу [15]. Належний захист конфіденційних даних також залишається ключовим аспектом інформаційної безпеки. Викрадення паролів, платіжних даних або токенів доступу можливе в разі відсутності шифрування та використання незахищених протоколів. Використання алгоритмів шифрування та протоколу HTTPS суттєво знижує ризики витоку інформації [16]. Ще однією критичною вразливістю є Buffer Overflow — ситуація, коли обсяг введених даних перевищує межі доступної пам’яті, що може спричинити збій програми або виконання довільного шкідливого коду [17].

Значну загрозу несе також можливість Unrestricted File Upload (UFU), коли система приймає завантаження файлів без перевірки MIME-типів, розширень або вмісту. Зловмисник може передати виконуваний файл, зберегти його на сервері та використати як точку входу для атаки. Запобігання цій вразливості включає ретельну валідацію, обмеження прав доступу до директорій та відокремлення зони збереження [18]. Нарешті, Security Misconfiguration, тобто неправильна

конфігурація компонентів, таких як сервери, бази даних або мережеві пристрої, створює умови для експлуатації системи через типові або незахищені налаштування. Часто це результат використання конфігурацій за замовчуванням, які легко вгадати або обійти [19].

Таблиця 2.1

Порівняльна характеристика популярних вразливостей

Тип вразливості	Причина	Ризики	Методи захисту
XSS	Відсутність перевірки вводу користувача.	Викрадення сесій, виконання шкідливого коду.	Очистка вводу, використання CSP.
SQL Injection	Неперевірене введення SQL-запитів.	Викрадення даних, зміна бази даних.	Підготовлені запити, ORM.
DoS	Перенавантаження запитами.	Відмова у доступі до сервісу.	Використання систем обмеження ресурсів.
RFI	Відсутність перевірки джерела файлів.	Виконання шкідливого коду.	Перевірка джерел, whitelist.
Buffer Overflow	Перевищення ліміту вхідних даних.	Збої у роботі, виконання коду.	Лімітування вводу, валідація розміру.
CSRF	Використання підроблених запитів.	Несанкціоновані дії від імені користувача.	CSRF-токени, перевірка реферера.
UFU	Завантаження шкідливих файлів.	Виконання коду, викрадення даних.	Перевірка типів файлів, валідація.
Security Misconfiguration	Невірна конфігурація компонентів.	Несанкціонований доступ, витік даних.	Регулярні перевірки налаштувань.
Sensitive Data Exposure	Некоректне шифрування даних.	Розголошення конфіденційної інформації.	SSL, шифрування, мінімізація доступу.

продовження таблиці 2.1

Broken Authentication	Некоректне управління автентифікацією.	Несанкціонований доступ до облікових записів.	Використання багатофакторної автентифікації.

Таким чином, аналіз популярних вразливостей вебдодатків демонструє, що ключовими загрозами є XSS, SQL-ін’єкції, DoS-атаки та інші проблеми, що виникають через недостатню перевірку вводу або неправильну конфігурацію системи. Дотримання рекомендацій OWASP та впровадження сучасних практик безпеки дозволяє значно зменшити ризики для вебдодатків. У таблиці 2.1 можна ознайомитись з порівняльною характеристикою популярних вразливостей.

2.2 CVE-огляд поширених вразливостей Java вебдодатків

Вебдодатки на Java широко застосовуються в корпоративному середовищі завдяки своїй надійності та масштабованості. Однак, як і будь-яке програмне забезпечення, вони не застраховані від появи вразливостей, які можуть ставити під загрозу безпеку даних і стабільність роботи системи. Саме тому ретельний аналіз і своєчасне виявлення потенційних ризиків є невід’ємною частиною процесу розробки і супроводу таких додатків.

Загальновідомі вразливості, які систематизуються у базі CVE (Common Vulnerabilities and Exposures), дають змогу відстежувати найбільш поширені проблеми безпеки. Це дозволяє розробникам і адміністраторам своєчасно впроваджувати необхідні заходи захисту, знижуючи ймовірність успішних атак. В Java-вебдодатках уразливості часто пов’язані з неправильним використанням API, сторонніх бібліотек або некоректними конфігураціями серверів, що може призводити до порушення цілісності і конфіденційності даних. Вивчення цих вразливостей і розуміння їх природи допомагає створювати більш безпечні і стійкі до атак додатки.

SQL-ін’єкції, які входять до переліку OWASP Top 10, залишаються одними з найпоширеніших уразливостей, що становлять загрозу для додатків, створених на Java. Прикладом такої уразливості є CVE-2019-14900, яка демонструє

ситуацію, коли Java-додаток із недостатньо захищеними SQL-запитами стає об’єктом атак. Основна причина полягає у динамічному формуванні запитів без використання параметризованих операторів PreparedStatement, що відкриває можливість для маніпуляцій із запитами до бази даних [20].

Не менш поширеними є вразливості міжсайтового скриптингу (XSS), які часто виникають у Java-додатках через недоліки в обробці користувацького вводу. Наприклад, CVE-2013-6430 була виявлена у популярному Java-фреймворку й виникала через недостатнє екранування або очищення введених користувачем даних, які могли містити JavaScript-код. Це дозволяло зловмиснику вставити шкідливі скрипти у контекст вебдодатка, отримати несанкціонований доступ до сесійних токенів або іншої конфіденційної інформації, а також виконати дії від імені користувача [21].

Вразливості типу CSRF також залишаються критичними для Java-додатків. CVE-2024-4328 деталізує уразливість, коли захист від CSRF-атак не був реалізований належним чином. У результаті цього зловмисники отримували можливість змусити автентифікованих користувачів здійснювати небажані дії, зокрема фінансові операції або зміну налаштувань акаунтів без їхньої згоди [22].

Проблеми з автентифікацією користувачів залишаються актуальними, особливо у випадках застосування слабких паролів або відсутності сучасних механізмів їхнього збереження. Уразливість CVE-2024-2365 ілюструє подібну ситуацію, коли вразливий Java-додаток не забезпечував належного захисту паролів, що дозволяло зловмисникам отримати доступ до системи. Одним із ефективних рішень є впровадження алгоритмів хешування на зразок bcrypt, які значно підвищують рівень захисту [23].

Java-додатки часто використовують сторонні бібліотеки, і вразливості в них також створюють потенційні загрози. Так, CVE-2025-30065 стосується застарілої версії бібліотеки Apache Struts, яка дозволяла виконання довільного коду. Це підкреслює необхідність регулярного моніторингу оновлень і усунення застарілих залежностей у програмному коді [24].

Конфігураційні вразливості на рівні серверів можуть призводити до витоку

даних або несанкціонованого доступу до критичних ресурсів. Наприклад, CVE-2025-24813 демонструє випадок неправильного налаштування сервера Tomcat, що дозволяло отримати доступ до конфіденційних файлів. Для мінімізації ризиків важливо дотримуватися практик безпечного налаштування, включаючи використання TLS, обмеження прав доступу та ізоляцію конфіденційних директорій [25].

REST API, який є ключовим компонентом багатьох сучасних Java-додатків, також піддається атакам у разі неналежної валідації запитів. Уразливість CVE-2024-20536 ілюструє ситуацію, коли неправильна обробка даних у API дозволяла змінювати стан системи через маніпуляції з параметрами. Запровадження перевірки автентичності, зокрема за допомогою токенів доступу, є критично важливим для забезпечення безпеки API [26].

Із розширенням застосування serverless-архітектур виникають нові типи уразливостей, зокрема пов’язані із неналежною ізоляцією середовища виконання. CVE-2020-8594 демонструє, як невірно налаштований серверless-функціонал у Java-додатку призвів до витоку даних через відсутність контейнерної ізоляції. Це вимагає комплексного підходу до конфігурації середовищ виконання в хмарних архітектурах [27].

Значну загрозу становить можливість завантаження небезпечних файлів, якщо обробка вхідних даних здійснюється без належної перевірки. Уразливість CVE-2025-4258 показує, як Java-додаток дозволяв завантаження файлів без перевірки типу або вмісту, що могло призвести до запуску шкідливого коду. Для запобігання таким інцидентам важливо обмежувати типи файлів, використовувати перевірку MIME-типів і вбудовувати антивірусне сканування у механізм обробки [28].

2.3 Дослідження та приклади реальних атак на вебдодатки Java

Вебдодатки на основі Java є важливою складовою корпоративного та індивідуального програмного забезпечення, однак їх поширеність робить їх

привабливою ціллю для кіберзлочинців. У цьому розділі розглядаються реальні випадки атак на Java вебдодатки, які демонструють найпоширеніші вразливості та підходи до їх експлуатації.

Одним із найбільш відомих прикладів є атака на сервери Apache Struts, зареєстрована у 2017 році. Зловмисники скористалися вразливістю CVE-2017-5638, що дозволяла виконувати шкідливий код через некоректну обробку заголовків Content-Type. Атака мала глобальний вплив і призвела до порушення безпеки багатьох організацій, включаючи витік конфіденційних даних [29].

Іншим відомим прикладом є атака на Equifax у 2017 році, яка стала результатом експлуатації тієї ж вразливості в Apache Struts. Зловмисники отримали доступ до понад 147 мільйонів записів користувачів, включаючи соціальні номери, фінансову інформацію та імена. Атака підкреслила важливість своєчасного оновлення серверних компонентів [30].

Атака на вебдодаток, заснований на Spring Framework, у 2018 році була здійснена через уразливість CVE-2018-1273. Ця вразливість дозволяла виконання довільного коду через обхід захисту в механізмі обробки запитів. Зловмисники використовували її для інфікування серверів програмами-шифрувальниками, що призводило до значних фінансових втрат у постраждалих організацій [31].

Вебдодатки на основі Java є важливою складовою корпоративного та індивідуального програмного забезпечення, однак їх поширеність робить їх привабливою ціллю для кіберзлочинців. У цьому розділі розглядаються реальні випадки атак на Java вебдодатки, які демонструють найпоширеніші вразливості та підходи до їх експлуатації, а також дозволяють краще зрозуміти природу цих загроз у контексті сучасного цифрового середовища.

Одним із найбільш відомих прикладів є атака на сервери Apache Struts, зареєстрована у 2017 році. Зловмисники скористалися вразливістю CVE-2017-5638, що дозволяла виконувати шкідливий код через некоректну обробку заголовків Content-Type. Атака мала глобальний вплив і призвела до порушення безпеки багатьох організацій, включаючи витік конфіденційних даних, про що

повідомлялося у численних офіційних звітах безпекових служб і дослідників [29].

Цей інцидент безпосередньо пов'язаний із одним із наймасштабніших витоків персональних даних в історії — атакою на компанію Equifax, яка відбулася в тому ж році. Зловмисники, скориставшись тією ж уразливістю в Apache Struts, отримали доступ до понад 147 мільйонів записів користувачів, серед яких були соціальні номери, фінансова інформація та особисті дані. Ця подія стала уроком для цілого ринку, продемонструвавши фатальні наслідки нехтування оновленням критично важливих компонентів у ланцюжку постачання програмного забезпечення [30].

Атака на вебдодаток, заснований на Spring Framework, у 2018 році, також привернула значну увагу. Уразливість CVE-2018-1273 дозволяла виконання довільного коду через механізм некоректної обробки параметрів запитів у Spring Data Commons. Деякі зловмисники використали цю вразливість для впровадження програм-вимагачів, що, своєю чергою, призвело до блокування критичних бізнес-процесів у постраждалих компаніях та потреби в дорогому відновленні інфраструктури [31].

Інший резонансний приклад — атака на серверні системи одного з великих постачальників послуг (CVE-2024-22320), яка стала можливою через уразливість небезпечної десеріалізації даних у Java-додатках. Напад був здійснений шляхом завантаження шкідливих об'єктів, які система інтерпретувала як коректні, що в підсумку дозволило виконати довільний код. Цей випадок яскраво ілюструє ризики, пов’язані з необачним використанням механізмів Java Serialization без відповідних обмежень або перевірок [33].

Ще один приклад масштабного інциденту — злам серверів Болгарської податкової служби у 2019 році, вказує на критичність дотримання принципів безпечного проєктування державних інформаційних систем. Недостатнє ізолювання середовищ виконання й незахищені API були використані як вхідні точки для витоку даних про мільйони громадян [32].

Сукупність розглянутих випадків атак на Java вебдодатки свідчить про

наявність системних вразливостей, пов’язаних із недоліками реалізації серверної логіки, обробки введених даних та механізмів серіалізації. Експлуатація таких вразливостей зловмисниками часто ґрунтується на передбачуваності поведінки застосунків, відсутності належного оновлення програмних компонентів або застосування застарілих конфігурацій безпеки. Також виявляється необхідність системного впровадження криптографічних протоколів, механізмів контролю цілісності даних та політик обмеження доступу як на рівні окремих компонентів, так і в межах архітектури додатка загалом

Висновки за розділом 2

Таким чином, у розділі було розглянуто основні класифікації вразливостей вебдодатків Java та наведено приклади реальних атак, які підтверджують значущість цієї проблеми. У якості підсумку варто зазначити, що вразливості поділяються на кілька ключових типів: SQL-ін’єкції, XSS-атаки, уразливості автентифікації, небезпечна десеріалізація, атаки на сесійну безпеку, а також помилки конфігурації.

Особливої уваги заслуговують практичні приклади успішної експлуатації типових вразливостей вебдодатків, зокрема ін’єкцій, міжсайтового скриптингу та виконання довільного коду. Такі інциденти засвідчують, що нехтування безпековими аспектами на будь-якому етапі життєвого циклу розробки може мати критичні наслідки для функціонування системи та захисту даних користувачів.

Розглянуті приклади також демонструють, що значна частина атак виникає через недоліки у валідації даних або несвоєчасне оновлення компонентів серверів. Це свідчить про критичність впровадження передових практик, таких як використання OWASP-рекомендацій, регулярне тестування систем на уразливості та застосування засобів моніторингу кіберзагроз.

На основі проведеного аналізу можна стверджувати, що для захисту Java вебдодатків необхідне комплексне поєднання засобів шифрування,

багатофакторної автентифікації, політик безпеки та автоматизованих інструментів виявлення вразливостей. Реалізація таких підходів забезпечить належний рівень захисту додатків від сучасних кіберзагроз.

РОЗДІЛ 3
МЕТОДИ ЗАХИСТУ ВЕБДОДАТКІВ JAVA

1.1 Можливості мови Java щодо захисту вебдодатків

Java є однією з найпоширеніших мов програмування для створення вебдодатків, і її можливості безпеки забезпечують надійний захист від багатьох типів кіберзагроз. Завдяки інтеграції технологічного та криптографічного захисту, Java надає інструменти, які дозволяють створювати захищені програми, що дозволяє мінімізувати ризики несанкціонованого доступу до даних чи серверних ресурсів.

До технологічного захисту у Java можна віднести платформу JVM (Java Virtual Machine), яка ізолює виконання коду, запобігаючи шкідливій діяльності. Важливою функцією JVM є перевірка байт-коду перед виконанням, що дозволяє уникати потенційно небезпечних операцій у додатках [34].

Також для забезпечення валідації введених даних розробники можуть використовувати бібліотеки, такі як Apache Commons Validator. Це допомагає виявляти та запобігати SQL-ін’єкціям, XSS-атакам та іншим загрозам. Такий підхід мінімізує ризики некоректного введення користувачами даних, що можуть реалізувати вразливості [35].

Фреймворки, як-от Spring Security, пропонують комплексний підхід до впровадження багатофакторної автентифікації, контролю доступу та захисту від атак CSRF. Ці інструменти інтегруються з іншими компонентами додатка, що дозволяє створювати багаторівневі механізми безпеки.

Java має потужні інструменти для забезпечення криптографічного захисту інформації, що обробляється у додатках. Однією з ключових бібліотек є Java Cryptography Architecture (JCA), яка надає інтерфейси для реалізації шифрування, цифрових підписів та гешування. Використання JCA дозволяє розробникам надійно шифрувати дані в додатках [35].

Java Cryptography Extension (JCE) розширює можливості JCA шляхом підтримки таких алгоритми як AES, RSA та інші. Це дозволяє створювати безпечні ключі, шифрувати дані та забезпечувати їх цілісність навіть у складних розподілених системах. Окрім того, JCE надає можливості для генерації криптографічно захищених випадкових чисел шляхом використання класу SecureRandom [35].

Для захисту мережевої взаємодії у Java пропонується Java Secure Socket Extension (JSSE). JSSE підтримує протоколи SSL/TLS для шифрування даних у процесі їх передачі. Це робить Java-додатки прийнятними для використання у середовищах, що вимагають високі стандарти конфіденційності даних, зокрема фінансові системи [35].

Клас KeyStore дозволяє зберігати криптографічні ключі та сертифікати у форматі Java KeyStore (JKS), що дозволяє забезпечити захист важливих даних. Також наявна підтримка інтеграції з апаратними модулями безпеки (HSM), що використовуються для зберігання ключів.

Для забезпечення таких характеристик даних як автентичність та цілісність даних у Java присутній клас Signature, який дозволяє створювати цифрові підписи та проводити процедуру їх перевірки. Таким чином наявні елементи дозволяють забезпечити високий рівень довіри до інформації, що передається, та мінімізує ризики фальсифікації таких даних.

Також варто поговорити про комплексний захист вебдодатків на Java, що забезпечує інтеграцію двох розглянутих двох типах захисту. Інтеграція технологічного та криптографічного захисту в Java дозволяє створювати багаторівневі механізми безпеки. Наприклад, автентифікація користувачів може базуватися на токенах JWT, які шифруються за допомогою RSA-ключів. Додатково, використання токенів CSRF у поєднанні зі Spring Security забезпечує захист від міжсайтових атак .

Не менш важливим є зазначення ролі інструментів тестування безпеки, таких як OWASP ZAP, що інтегруються з Java-додатками для автоматизованого виявлення вразливостей. Використання таких рішень дозволяє оцінити

ефективність впроваджених засобів безпеки та забезпечити належний рівень захисту [36].

1.2 Методика захисту та протидії

Аналіз наукових досліджень, присвячених виявленню вразливостей у вебдодатках, демонструє широкий спектр методів та технологій, спрямованих на запобігання кіберзагрозам. Одним із таких методів є XSS-Check, розроблений у 2017 році, який дозволяє виявляти атаки типу Cross-Site Scripting (XSS). Цей інструмент аналізує динамічні вебсторінки, перевіряючи введені користувачем дані та структуру вебінтерфейсу на предмет наявності шкідливих скриптів. XSS-Check також виконує верифікацію HTTP-заголовків та параметрів DOM, що дозволяє ідентифікувати потенційні вразливості ще на стадії розробки вебдодатків [37].

У 2019 році було запропоновано метод виявлення SQL-ін’єкцій на основі архітектури Elastic Pooling CNN (EP-CNN). Цей підхід дозволяє автоматично аналізувати вхідні SQL-запити та виявляти характерні ознаки шкідливого трафіку. Головною перевагою EP-CNN є здатність обробляти дані у вигляді статичної 2D-матриці без втрати інформації, що підвищує точність виявлення атак та дозволяє застосовувати його в реальному часі для захисту вебдодатків [38].

Важливим напрямком забезпечення безпеки вебдодатків є використання технологій оцінки вразливостей та тестування на проникнення. У 2020 році дослідники проаналізували різні підходи до виявлення слабких місць у вебдодатках, включаючи інтеграцію відкритих інструментів для автоматичного сканування та оцінки безпеки систем. Було встановлено, що використання комплексних методів оцінки вразливостей дозволяє своєчасно виявляти та усувати потенційні загрози [39].

Для захисту від DoS-атак у 2018 році було розроблено систему Rampart, яка застосовує математичні моделі для виявлення та блокування атак, спрямованих на виснаження процесорних ресурсів. Rampart інтегрується з PHP

Zend Engine та дозволяє мінімізувати вплив атак на продуктивність вебдодатків. Зокрема, було продемонстровано, що ця технологія ефективно захищає популярні CMS-платформи, такі як WordPress і Drupal, від реальних та синтетичних DoS-атак [40].

Запобігання атакам типу Buffer Overflow вимагає використання захисних механізмів, таких як обмеження вхідних даних, застосування безпечних бібліотек та перевірка вихідного коду. Програмісти можуть використовувати засоби обмеження довжини введених рядків за допомогою JavaScript та HTML, однак для повноцінного захисту необхідно впроваджувати контроль вхідних даних на рівні сервера. Також ефективним методом є використання міжмережевих екранів, які дозволяють ідентифікувати підозрілу активність та запобігати атакам, що спричиняють переповнення буфера [55].

Одним із ефективних підходів до тестування безпеки вебдодатків є використання платформи FUSE, запропонованої у 2020 році для виявлення уразливостей у PHP-додатках. Система автоматично створює тестові запити, які дозволяють перевірити стійкість вебдодатків до атак через завантаження шкідливих файлів. Дослідження показали, що FUSE виявила 30 уразливостей у 33 протестованих програмах, включаючи 15 CVE, що підтверджує її високу ефективність у виявленні загроз [18].

Запобігання атакам TCP SYN Flood (DoS) може бути реалізовано за допомогою методів SPI (Stateful Packet Inspection) та налаштувань CSF у Linux. Запропонований підхід передбачає блокування надмірної кількості SYN-запитів за короткий проміжок часу, що дозволяє значно зменшити ризик перевантаження серверів. Крім того, ефективним рішенням є використання хмарних сервісів, таких як Cloudflare, які дозволяють фільтрувати шкідливий трафік та запобігати атакам на інфраструктуру вебдодатків [41].

Проблеми автентифікації та управління сеансами можуть бути вирішені шляхом впровадження технологій, що базуються на доказах з нульовим розголошенням (ZKP). У 2020 році було запропоновано використання хаотичних систем для генерації динамічних ключів, що дозволяє мінімізувати ризик

перехоплення паролів та забезпечити надійний механізм автентифікації. Дослідження показали, що запропонований підхід забезпечує високу стійкість до атак та дозволяє створювати безпечні вебдодатки [43].

Нарешті, використання систем оцінки вразливостей та автоматизованих сканерів дозволяє підвищити ефективність захисту вебдодатків. Наприклад, система SNEAKERZ забезпечує покращене виявлення загроз завдяки інтеграції кількох інструментів аналізу безпеки, що дозволяє підвищити точність виявлення вразливостей на понад 100% у порівнянні зі стандартними методами [42].

Загалом, сучасні методи захисту вебдодатків охоплюють широкий спектр технологій, включаючи машинне навчання, поведінковий аналіз, криптографічні алгоритми та автоматизовані засоби оцінки безпеки. Їхнє поєднання дозволяє забезпечити ефективний захист від сучасних кіберзагроз та гарантувати стабільну роботу вебдодатків.

Висновки за розділом 3

Таким чином, було розглянуто основні методи захисту вебдодатків Java. У підсумку варто зазначити, що безпека таких додатків базується на використанні багаторівневого підходу, який включає аутентифікацію та авторизацію, шифрування даних, захист від загроз типу SQL-ін'єкцій, XSS, CSRF, а також застосування безпечних практик програмування.

Було детально розглянуто механізми захисту, такі як використання Spring Security для управління доступом, шифрування з використанням TLS/SSL для безпечного з’єднання, а також принципи OWASP для захисту від вебвразливостей.

Також варто зазначити, що впровадження цих методів значно підвищує безпеку вебдодатків Java, однак забезпечення повного захисту вимагає регулярного оновлення програмного забезпечення, проведення тестування на вразливості та дотримання сучасних стандартів безпеки.

РОЗДІЛ 4
ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ ВЕБДОДАТКІВ, РОЗРОБЛЕНИХ З ВИКОРИСТАННЯМ JAVA

4.1 Обґрунтування вибору вразливостей

Для реалізації практичної частини дипломної роботи було вибрано чотири типів вразливостей, які мають найвищу поширеність, ризик впливу на систему та релевантність у контексті розробки вебдодатків на Java. Вибір кожної з них спрямовано на демонстрацію контрасту між уразливою реалізацією вебфункціоналу та безпечною альтернативою на основі сучасних технологій захисту, що підтверджуються галузевими стандартами та практиками безпеки, зокрема рекомендаціями OWASP Top 10.

Першою обраною вразливістю стала Cross-Site Scripting (XSS), оскільки вона є однією з найчастіших загроз, яка виникає внаслідок недостатньої перевірки вхідних даних. XSS дозволяє зловмиснику впровадити шкідливий JavaScript-код, що виконується у браузері жертви, з метою викрадення сесійних токенів або підміни вмісту сторінки. Актуальність цієї вразливості підтверджується її стабільною присутністю у звітах OWASP, а також значною кількістю інцидентів, зафіксованих у реальних системах. Демонстрація XSS у тестовому середовищі дозволяє візуалізувати механізм атаки та ефективність захисту через очищення вхідного запиту (input sanitization).

Другою вразливістю було обрано SQL-ін’єкцію, яка виникає внаслідок некоректної обробки параметрів запитів до бази даних. Атака SQL дозволяє зловмиснику змінити логіку виконання SQL-запиту для несанкціонованого доступу до інформації або її модифікації. Особливої уваги вона набуває у вебдодатках, побудованих із використанням JDBC або ORM-бібліотек без належного параметризування. У тестовому застосунку було реалізовано приклад форми з методом пошуку користувачів, де SQL-ін’єкція дозволяє отримати всі

дані. Це дає змогу чітко продемонструвати, як параметризовані запити, підготовлені вирази (PreparedStatements) або ORM-фреймворки типу Hibernate ефективно усувають цю загрозу.

Третій аспект безпеки стосується обробки небезпечних файлів, зокрема можливості завантаження шкідливого вмісту через форму файлів, що часто ігнорується в класичних тестах безпеки. Вибір цієї вразливості пов’язаний із поширеною практикою використання модулів завантаження файлів у корпоративних системах управління контентом (Content Management Systems), де відсутність перевірки типу, структури або вмісту файлу призводить до можливості виконання шкідливого коду на сервері. Для підвищення ефективності виявлення та блокування таких загроз було інтегровано механізм перевірки через сигнатурну систему YARA, яка дозволяє здійснювати контент-орієнтований аналіз файлів. Така інтеграція є прикладом розширеного підходу до превентивного захисту та демонструє потенціал використання механізмів безпеки нижчого рівня в Java-додатках.

Останнім, але не менш важливим компонентом практичної частини, стала демонстрація слабкої автентифікації, зосередженої на проблемах збереження паролів. На жаль, у багатьох реальних системах і досі зустрічається зберігання паролів у відкритому вигляді або з використанням слабких хеш-функцій, таких як MD5 (zdnet wiki) або SHA-1. Тому в рамках цієї роботи було вирішено показати уразливу реалізацію, яка зберігає паролі у відкритому вигляді, та захищену — із використанням алгоритму BCrypt, що дозволяє забезпечити адаптивне хешування з сіллю, захищаючи тим самим паролі від перебору навіть у разі витоку бази даних. Вибір BCrypt є обґрунтованим з огляду на його підтримку у фреймворках Spring Security та його стійкість до атак типу brute-force.

4.2 Архітектура та опис додатку

Застосунок організовано у вигляді Maven-проєкту (рисунок 4.1), основна конфігурація якого розміщена у файлі pom.xml. У цьому файлі перелічено необхідні залежності, серед яких spring-boot-starter-web, spring-boot-starter-security, spring-boot-starter-thymeleaf, spring-boot-starter-data-jpa, spring-boot-devtools, а також вбудована база даних H2, яка забезпечує зручність при розробці та тестуванні функціоналу без додаткового налаштування зовнішніх СУБД.

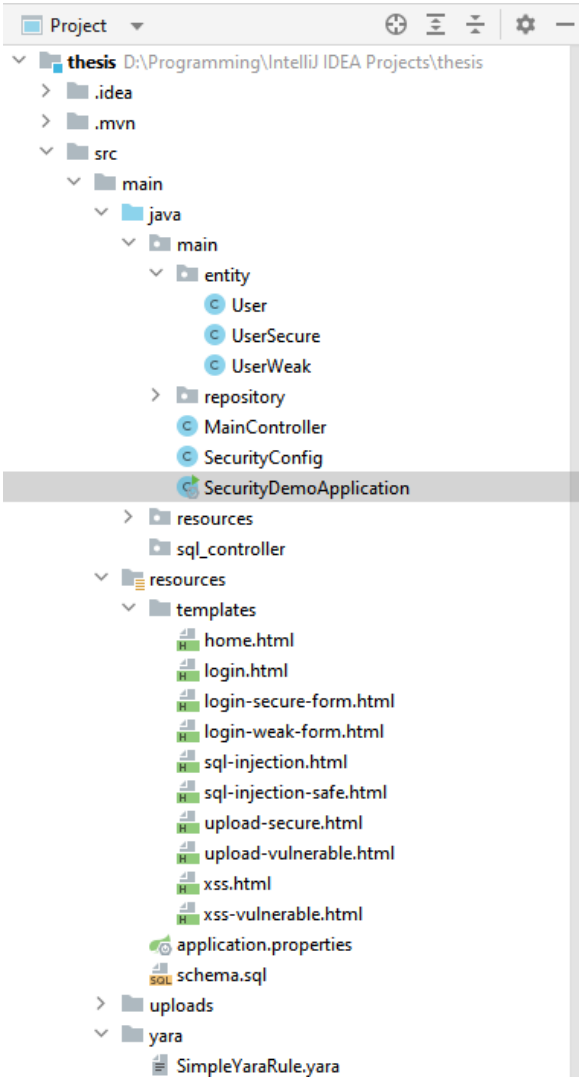

Рисунок 4.1 - Структура розробленого застосунку

У структурі проекту src/main/java розташовані такі основні пакети:

1. main.entity — містить сутності, зокрема класи User, UserSecure і UserWeak. Вони репрезентують таблиці бази даних із відповідними полями,

включно з іменем користувача, електронною адресою та паролем (у захищеному чи вразливому вигляді).

2. main.repository — забезпечує доступ до бази даних та включає класи-репозиторії для обробки користувацьких запитів.

3. main — містить клас SecurityConfig, що відповідає за конфігурацію Spring Security, зокрема правила авторизації, форми логіну та визначення тестових користувачів за допомогою InMemoryUserDetailsManager. Також у цьому пакеті знаходиться основний контролер MainController, що обробляє маршрутизацію до відповідних сторінок і обробку форм введення для демонстрації різних уразливостей.

Контролер MainController реалізує логіку переходу між захищеними та уразливими сторінками. Зокрема, методи контролера обробляють такі маршрути:

/login — реалізація класичної аутентифікації з підтримкою ролей;

/home – реалізація головного меню;

/xss та /xss-vulnerable — демонстрація захищеної (через th:text) та незахищеної (через th:utext) реалізацій обробки введених користувачем HTML-коментарів;

/sql-injection та /sql-injection-safe — реалізація форми логіну із класичною SQL-ін’єкцією через конкатенацію рядків у запиті та безпечного варіанту із використанням параметризованого SQL-запиту;

/login-weak-form та /secure-login — порівняння слабкої автентифікації, що використовує MD5 для перевірки пароля, із безпечною реалізацією, яка використовує BCryptPasswordEncoder;

/file-upload та /file-upload-secure — реалізація функції завантаження файлів без перевірок та з перевіркою за розширенням і сигнатурним аналізом через інтеграцію YARA CLI, яка блокує шкідливі файли перед остаточним збереженням.

Сторінки інтерфейсу реалізовані за допомогою Thymeleaf-шаблонів, які розташовані у директорії src/main/resources/templates. У проекті присутні шаблони для кожного демонстраційного кейсу:

Також у проекті наявні файли сторінок, як-от: xss.html, xss-vulnerable.html, sql-injection.html, sql-injection-safe.html, login-weak.html, secure-login.html, file-upload.html, file-upload-secure.html, login.html, home.html – усі вони слугують інтерактивним середовищем для введення даних, ініціювання атаки та спостереження за реакцією системи у вразливому та захищеному режимі.

Ініціалізація бази даних здійснюється через файл schema.sql (додаток Ф), у якому описано створення таблиць users, weak_users та login_users. Це забезпечує автоматичне підготування базової структури при запуску додатка, що включає в себе створення таблиць та їх наповняння.

Окрему увагу приділено підтримці системи виявлення шкідливих файлів. У папці src/main/yara зберігаються правила сигнатурного аналізу YARA, які застосовуються при завантаженні файлів у безпечному варіанті реалізації. Застосунок викликає зовнішній сканер yara64.exe з підключенням правил SimpleYaraRule.yara, що дозволяє реалізувати концепт розпізнання файлів за вмістом під час завантаження.

Результатом запуску Spring Boot-додатку є інтерфейс, доступний через веббраузер. Користувач після входу може обирати приклади вразливих та захищених реалізацій із навігаційного меню та експериментувати з введенням даних або завантаженням файлів, спостерігаючи відмінності у поведінці системи.

4.3 Реалізація захисту від конкретних вразливостей

Розгляньмо роботу розробленого вебзастосунку. На рисунку 4.2 представлено вікно входу у застосунок. Передбачено, що користувач може ввести йому відомі дані (пара ім’я-пароль: admin, admin) для доступу до ресурсів застосунку. За логіку цього вікна відповідає метод login (цей метод знаходиться у файлі «MainController.java», що можна переглянути у додатку А), а за відображення сторінки відповідає файл login.html (Додаток И). При неправильному вводі даних у вході буде відмовлено (рисунок 4.3).

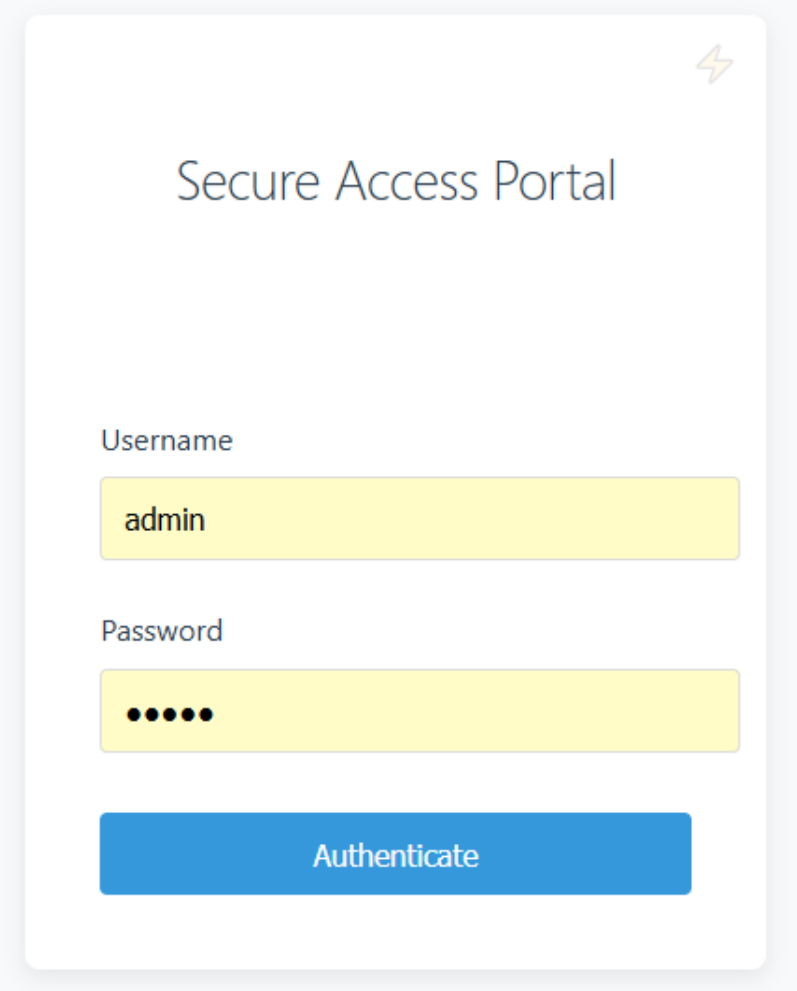

Рисунок 4.2 - Вікно входу

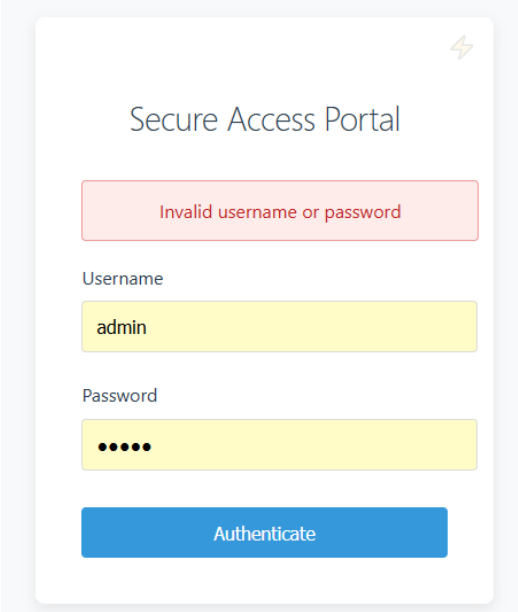

Рисунок 4.3 - Відмова у вході

Після успішного входу користувач потрапляє до головного меню вибору (рисунок 4.4). Тут у нього є можливість вибрати тип вразливості та переглянути її функціонал. Логіка реалізована методом home (цей метод знаходиться у файлі «MainController.java», що можна переглянути у додатку А), а за відображення сторінки відповідає файл home.html (Додаток З).

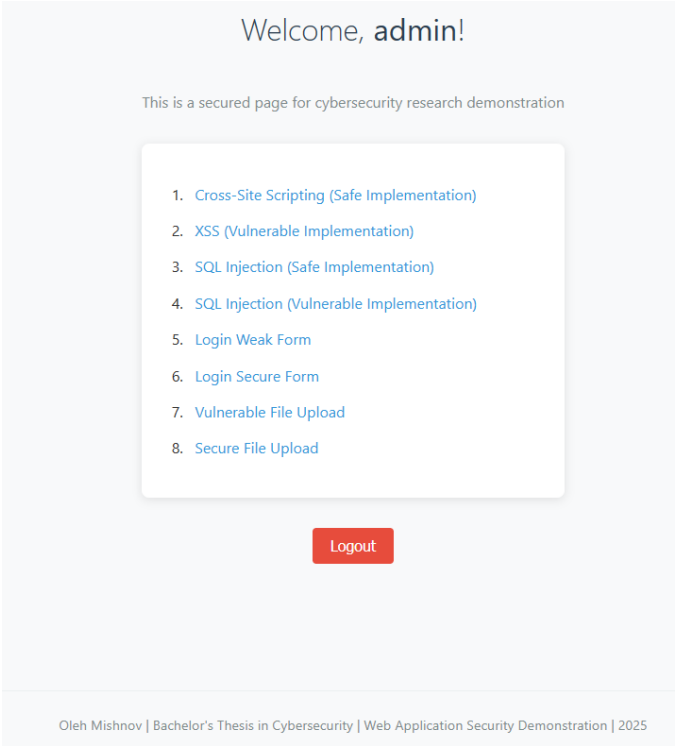

Рисунок 4.3 - Головне меню

Перейдімо тепер до безпосереднього огляду вразливостей та протидій їм, що були реалізовані у цьому застосунку.

Першою розглянемо XSS-атаку. Такі атаки виникають у результаті несанкціонованої ін’єкції JavaScript-коду через відсутність екранування вхідних даних користувача.

Обираємо небезпечну версію цієї вразливості. За логіку відповідає метод xss-vulnerable (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.4, відповідає файл xss-vulnerable.html (додаток К).

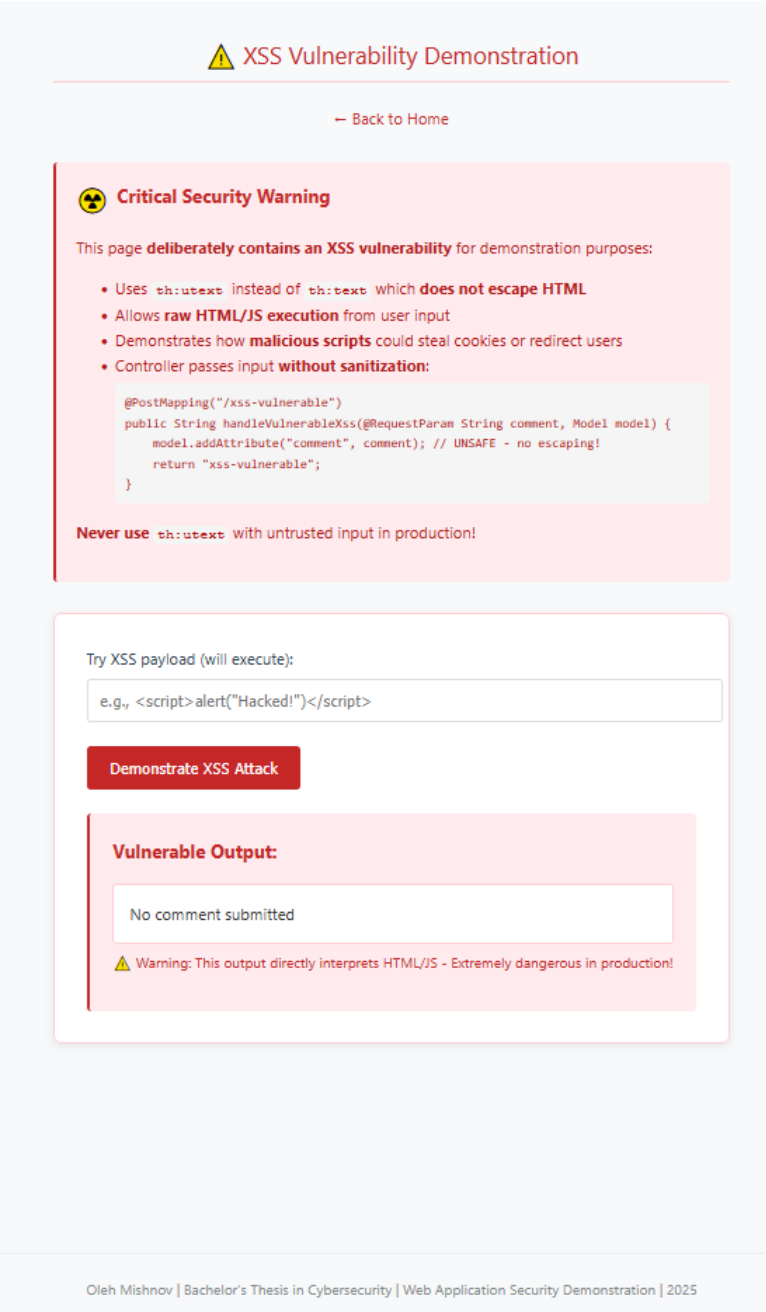

Рисунок 4.4 - Вразлива версія сторінки XSS

На сторінці також показано базову інформацію про цю вразливість. Спробуймо ввести базовий запит «<script\x20type="text/javascript">javascript:alert(1);</script>». Результат на рисунку 4.5. Як ми бачимо, у виводі ми отримали виключно «javascript:alert(1);». Це означає що при спробі вводу подібного рядка в аналогічну систему буде виконана скриптова частина – ввід не інтерпретується як простий текстовий рядок.

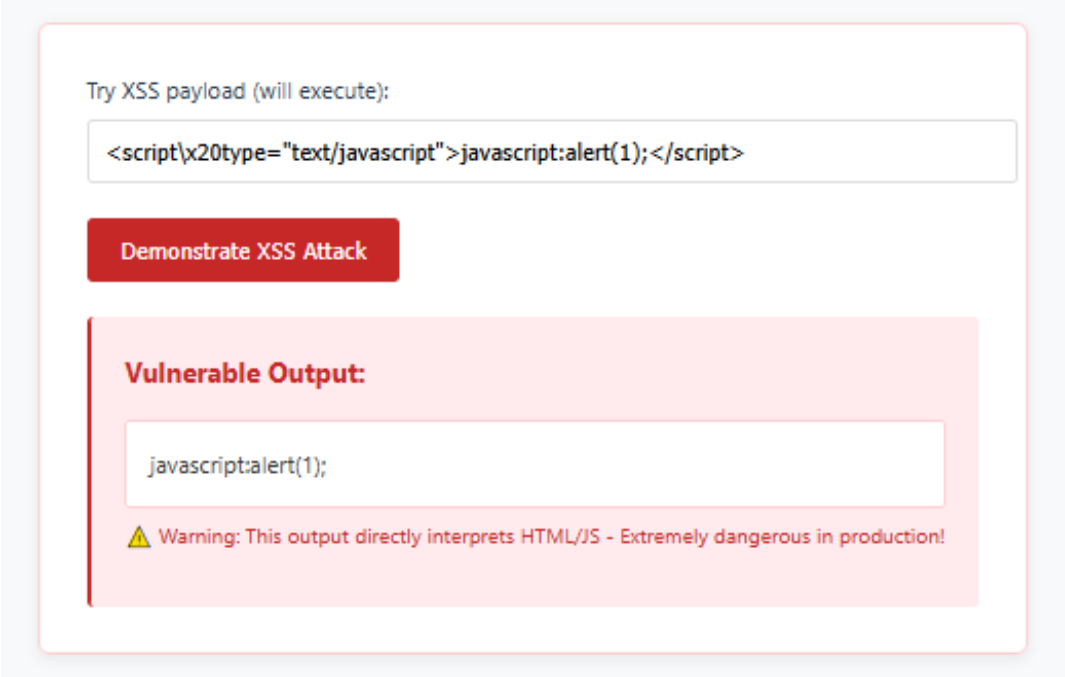

Рисунок 4.5 - Виконання скріпта

Вразливий код використовує шаблон th:utext, що виводить введений текст без попередньої обробки. Введення коду <script>alert(1)</script> призводить до виконання скрипта на боці клієнта. Це відбувається через відповідний запис у файлі сторінки, що вказано на рисунку 4.6.

```
<div class="output-panel">
    <h3>Vulnerable Output:</h3>
    <div class="output-content" th:utext="${comment} ?: 'No comment submitted'">
        No comment submitted
    </div>
```

Рисунок 4.6 - Вразливість у файлі

Тепер перейдемо до безпечної версії. За логіку відповідає метод xss (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.7, відповідає файл xss.html (додаток Т).

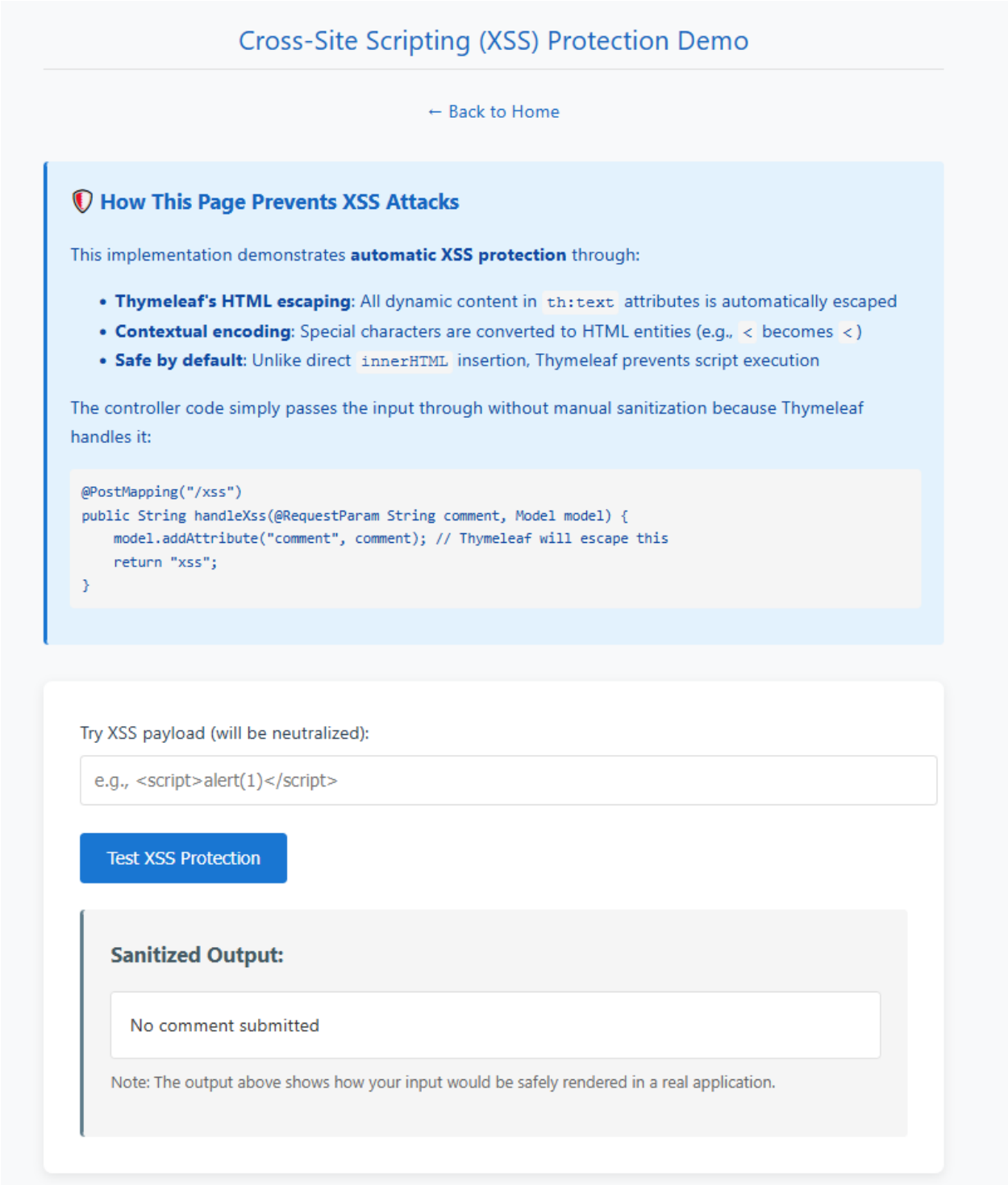

Рисунок 4.7 - iБезпечна версія сторінки XSS

На сторінці також показано базову інформацію про цю вразливість. Спробуймо ввести базовий запит «<script\x20type="text/javascript">javascript:alert(1);</script>». Результат на рисунку 4.8. Як ми бачимо, у виводі ми отримали дзеркальний вивід. Це означає що при спробі вводу подібного рядка в аналогічну систему скриптова система не

буде виконана, оскільки запит сприймається як текстовий рядок.

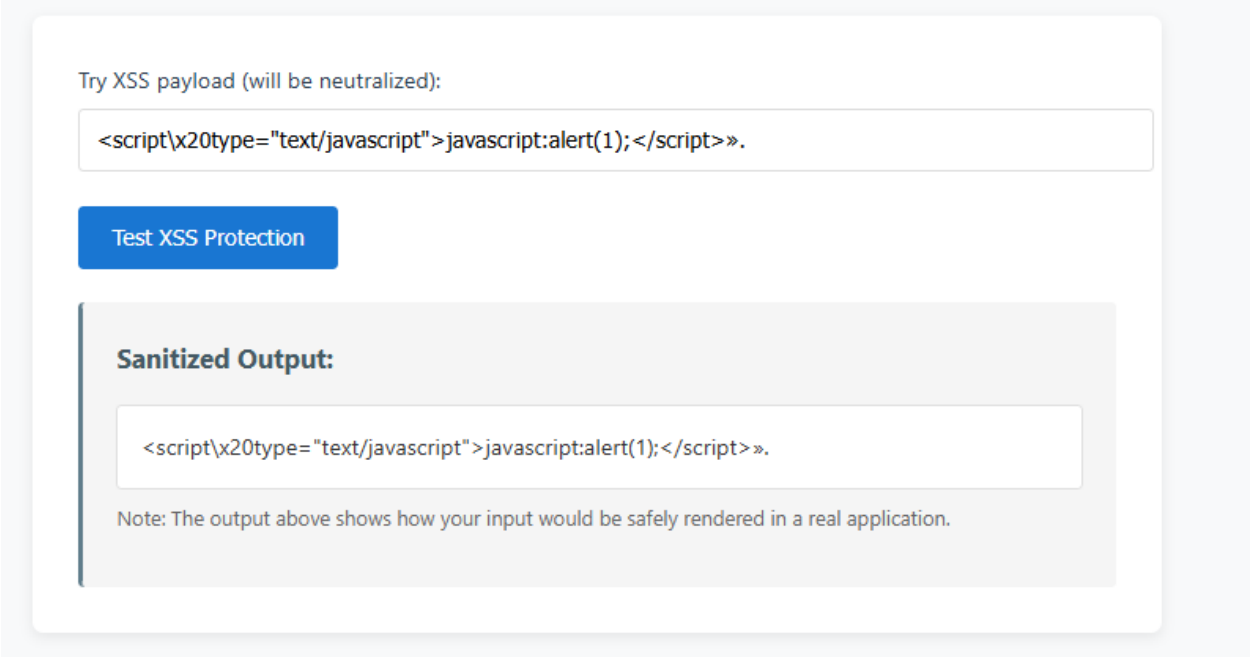

Рисунок 4.8 – Немає виконання скріпта

Захищений варіант застосовує директиву th:text, яка автоматично екранує символи <, > та інші потенційно небезпечні елементи. В результаті шкідливий скрипт не виконується, а відображається як звичайний текст. Це відбувається через відповідний запис у файлі сторінки, що вказано на рисунку 4.8.

```
<div class="output-panel">
    <h3>Sanitized Output:</h3>
    <div class="output-content" th:text="${comment} ?: 'No comment submitted'">
        No comment submitted
    </div>
    <p style="color: #666; font-size: 0.9rem; margin-top: 0.5rem;">
        Note: The output above shows how your input would be safely rendered in a real application.
    </p>
</div>
div>
```

Рисунок 4.9 - Захист у файлі

Тепер перейдімо до SQL-ін’єкції. Цей тип атаки базується на підміні структури SQL-запиту шляхом ін’єкції вхідних параметрів, що особливо небезпечно при формуванні запитів через конкатенацію рядків.

Обираємо небезпечну версію цієї вразливості. За логіку відповідає метод sql-injection (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.10, відповідає файл sql-injection.html (додаток М).

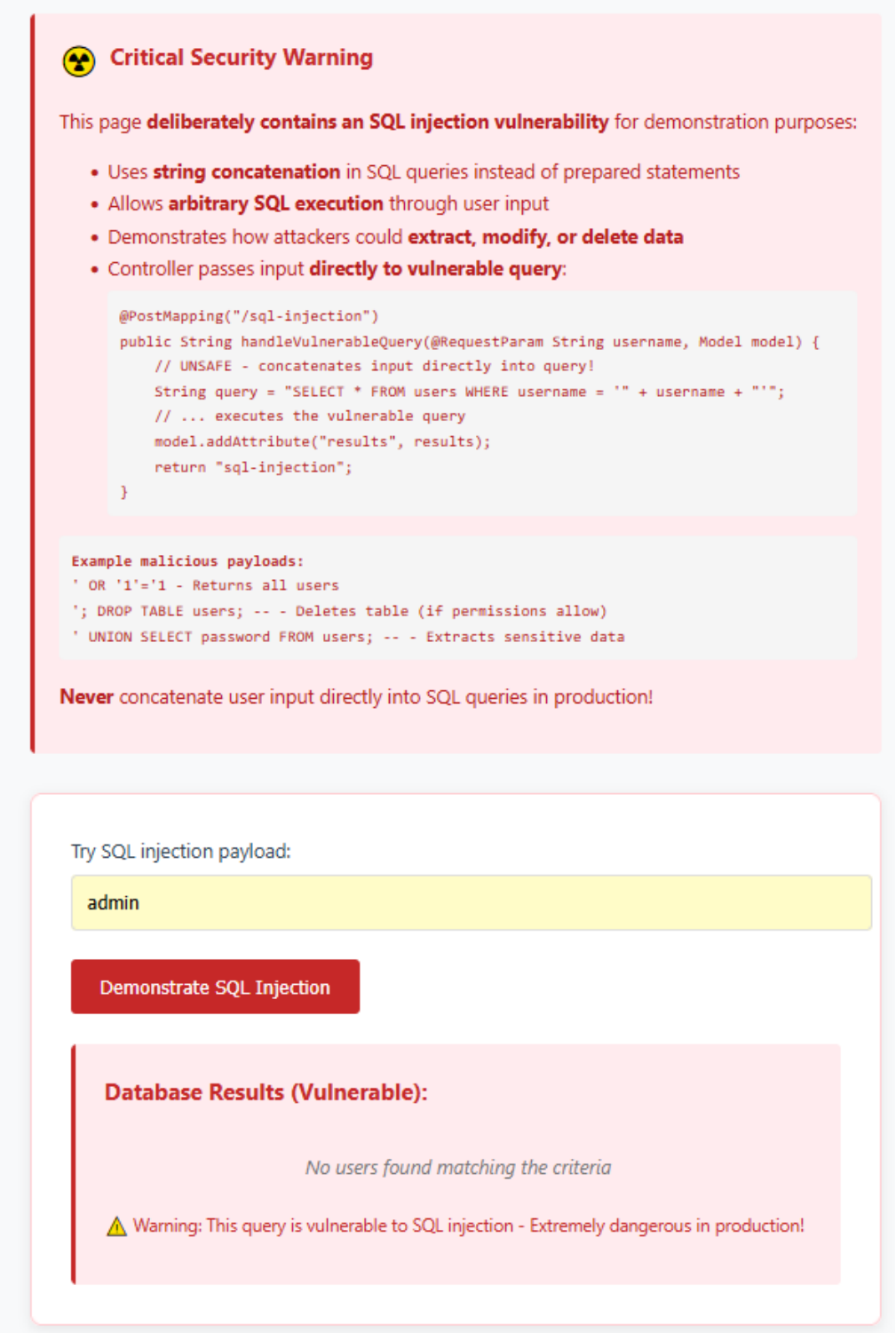

Рисунок 4.10 - SQL-ін’єкція Вразлива версія

На сторінці також міститься інформація про спосіб, за яким ця вразливість є можливою. Принцип полягає у наступному: формується запит типу « "SELECT * FROM users WHERE username = '" + username + "'"» (у методі injection), де username є просто підстановкою вводу користувача.

Спробуймо ввести ім’я користувача, що є в БД, і вираз «' OR '1'='1» (рисунки 4.11-4.12).

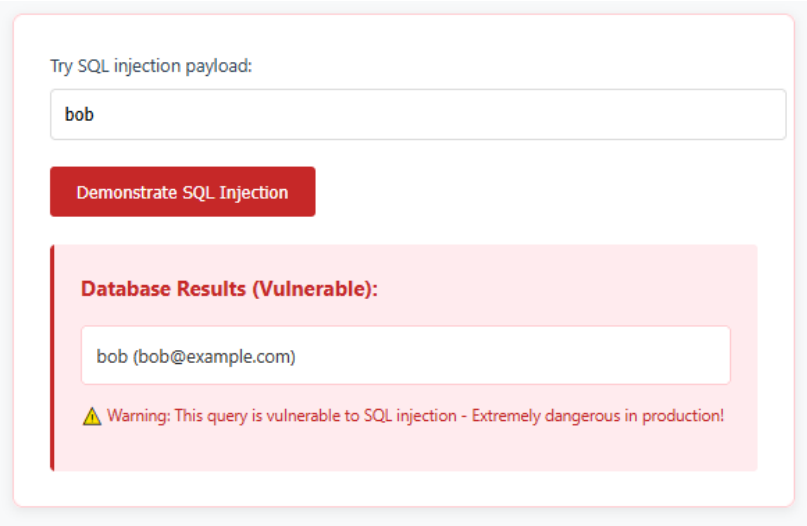

Рисунок 4.11 - Запит за звичайним іменем

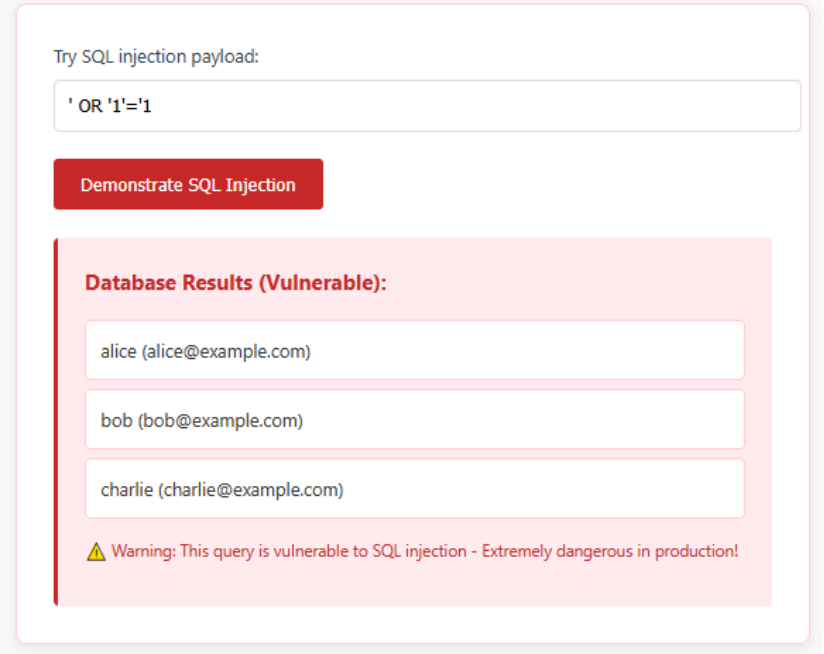

Рисунок 4.12 - Запит за виразом

Як ми бачимо, у першому випадку відбувається вивід за конкретним іменем користувача. У другому випадку теоретичний зловмисник зміг отримати доступ до всіх записів за допомогою спеціального виразу.

Обираємо захищену версію цієї вразливості. За логіку відповідає метод sql-injection-safe (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.13, відповідає файл sql-injection-safe.html (додаток Н).

На сторінці також міститься інформація про спосіб, за яким ця вразливість є нейтралізована. Принцип полягає у наступному: захищена реалізація застосовує параметризований запит із PreparedStatement «String query = "SELECT * FROM users WHERE username = ?";» і передає параметр окремо, що унеможливлює SQL-ін’єкцію.

Спробуймо ввести ім’я користувача, що є в БД, і вираз «' OR '1'='1» (рисунки 4.14-4.15).

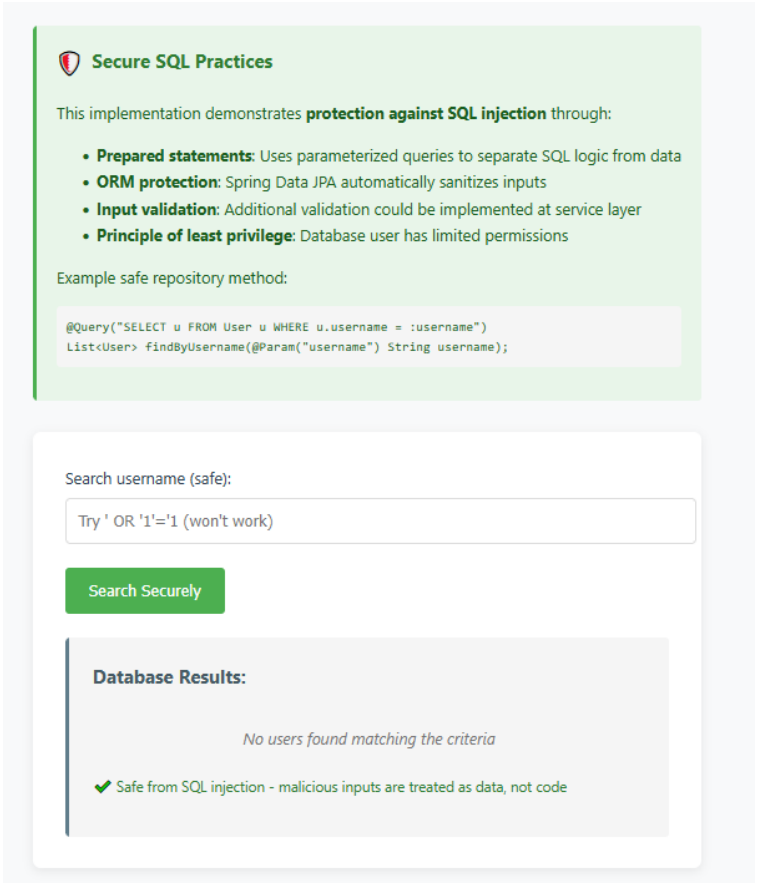

Рисунок 4.13 - SQL-ін’єкція захищена версія

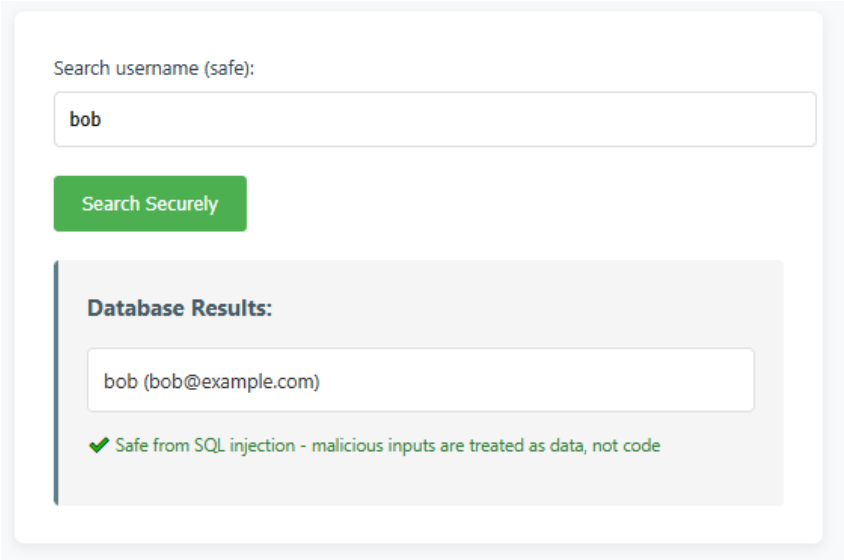

Рисунок 4.15 - Запит за звичайним іменем

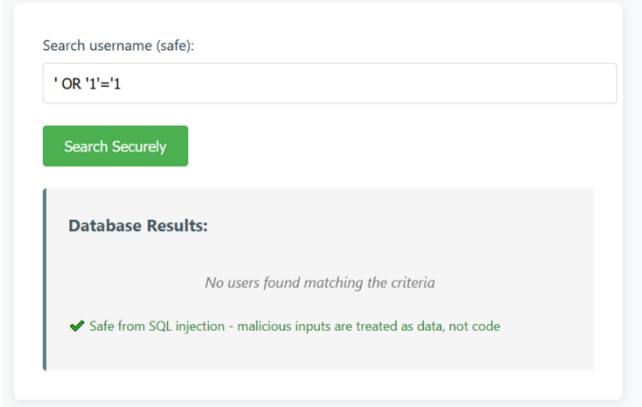

Рисунок 4.15 - Запит за виразом

Як ми бачимо, у першому випадку відбувається вивід за конкретним іменем користувача. У другому випадку теоретичний зловмисник вже нічого не зможе тримати, оскільки такого користувача не було знайдено.

Тепер перейдімо до завантаження файлів. Однією з критичних вразливостей є можливість завантаження шкідливих файлів. Уразливий варіант зберігає файли без перевірки MIME-типу чи вмісту, що дозволяє розміщення шкідливих .jsp або .html.

Обираємо небезпечну версію цієї вразливості. За логіку відповідає метод upload-vulnerable (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.16, відповідає файл upload-vulnerable.html (додаток Р). Код методу представлено на рисунку 4.17.

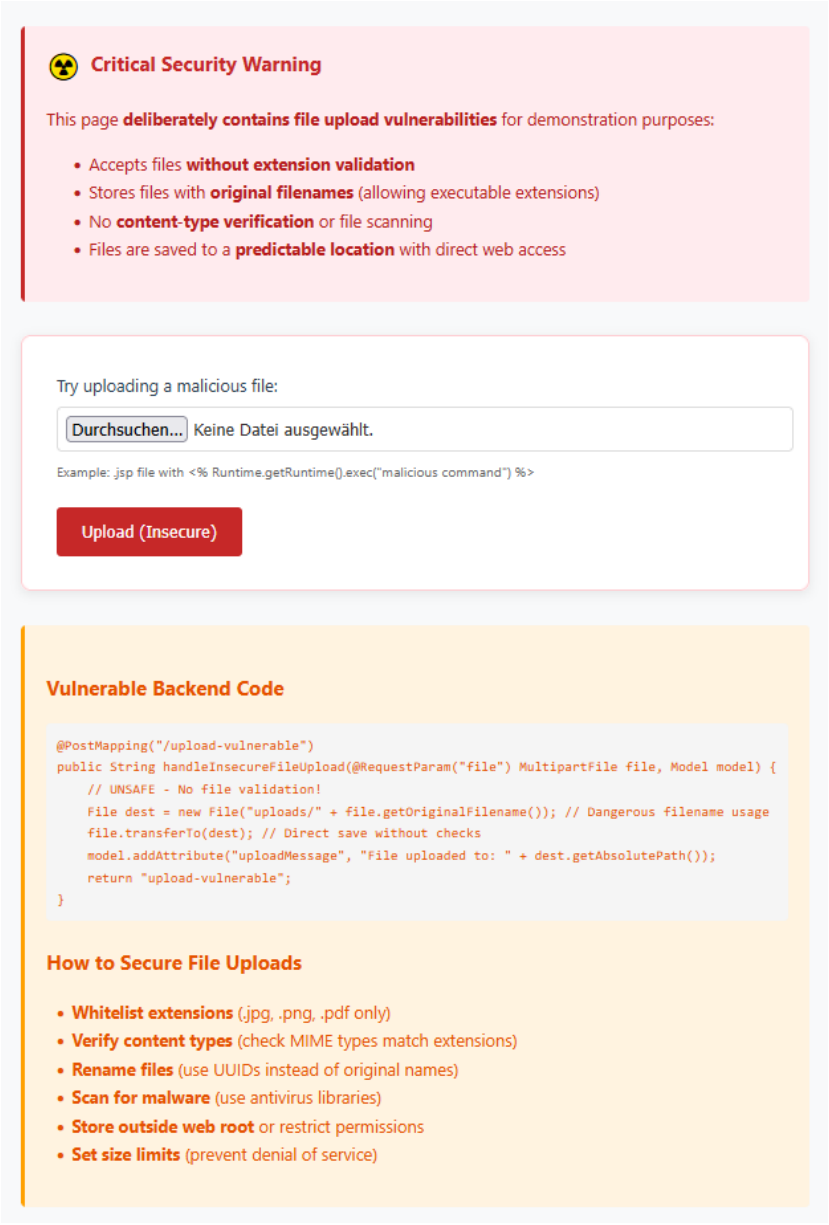

Рисунок 4.16. Завантаження файлів – незахищена версія

```
@GetMapping("/upload-vulnerable")
public String showUploadVulnerable() { return "upload-vulnerable"; }


@RequestMapping(value = "/upload-vulnerable", method = RequestMethod.POST)
public String handleInsecureFileUpload(@RequestParam("file") MultipartFile file, Model model) {
    if (file.isEmpty()) {
        model.addAttribute( attributeName: "uploadMessage",  attributeValue: "No file selected.");
        return "upload-vulnerable";
    }

    try {
        // Зберігаємо файл БЕЗ перевірки розширення або типу
        File dest = new File( pathname: "D:\\Programming\\IntelliJ IDEA Projects\\thesis\\src\\main\\uploads\\" + file.getOriginalFilename());
        file.transferTo(dest);
        model.addAttribute( attributeName: "uploadMessage",  attributeValue: "File uploaded successfully");
    } catch (IOException e) {
        model.addAttribute( attributeName: "uploadMessage",  attributeValue: "Error uploading file");
    }

    return "upload-vulnerable";
}
```

Рисунок 4.17. Завантаження файлів – логіка

На сторінці також міститься інформація про спосіб, за яким ця вразливість є можливою. Принцип полягає у наступному: використовується просто запис файлу у локальне сховище «file.transferTo(new File("uploads/" + file.getOriginalFilename()));»; також відсутня перевірка розширення файлу.

Спробуймо завантажити файл (вміст на рисунку 4.18) й перегляньмо результат на рисунку 4.19

definitely not a script.txt - Editor

Datei Bearbeiten Format Ansicht Hilfe

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test Script</title>
</head>
<body>
    <h1>Test Script for YARA Detection</h1>

    <script>
        // Example of a malicious script
        var x = 10;
        eval("alert('Malicious code executed!')");
    </script>

    <p>This is a dummy page with a suspicious script for YARA rule testing.</p>
</body>
</html>
```

Рисунок 4.18 - Файл з елементами ШПЗ

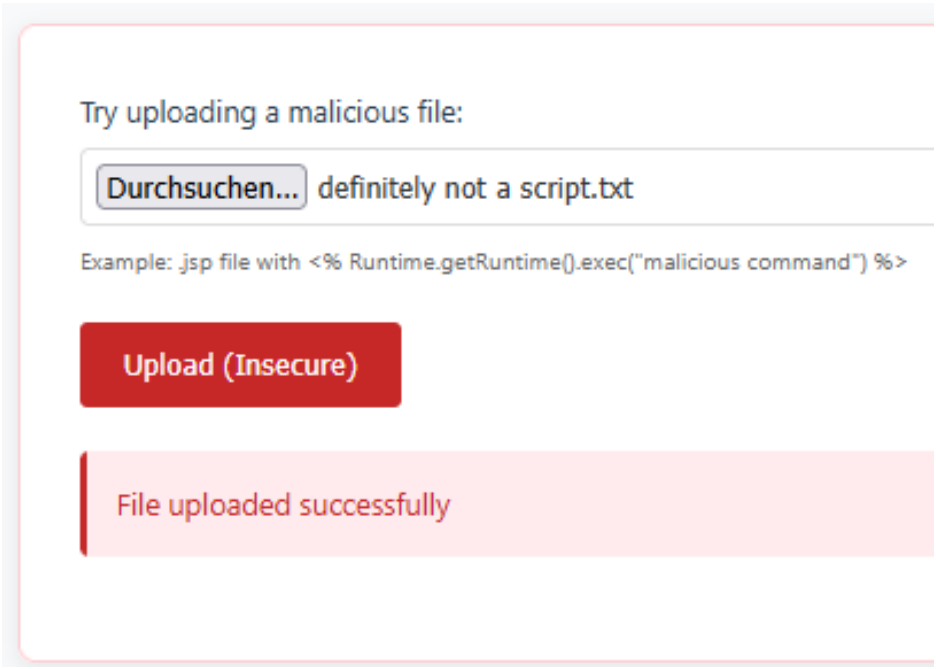

Рисунок 4.19 - Завантаження файлу

Як ми бачимо, файл зі шкідливим вмістом було завантажено.

Обираємо безпечну версію цієї вразливості. За логіку відповідає метод upload-secure (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.21, відповідає файл upload-secure.html (додаток П). Код методу представлено на рисунку 4.20.

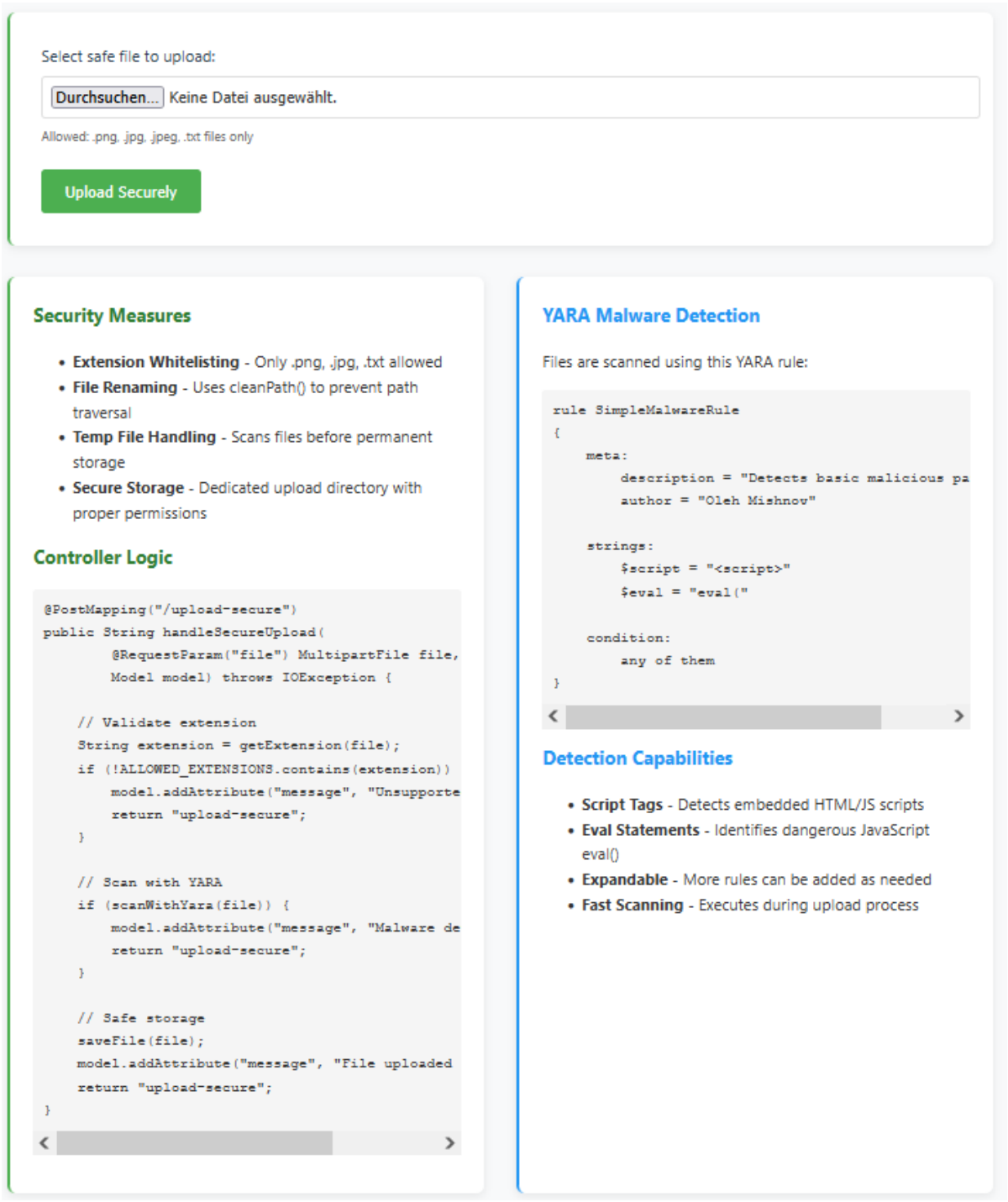

Рисунок 4.20 - Завантаження файлів – безпечна версія

```
String originalFilename = StringUtils.cleanPath(file.getOriginalFilename());
String extension = StringUtils.getFilenameExtension(originalFilename);

if (extension == null || !(extension.equalsIgnoreCase( anotherString: "txt") || extension.equalsIgnoreCase( anotherString: "jpg") |
    model.addAttribute( attributeName: "message",  attributeValue: "Unsupported file type");
    return "upload-secure";
}

File tempFile = File.createTempFile( prefix: "upload_",  suffix: "_" + originalFilename);
try (FileOutputStream fos = new FileOutputStream(tempFile)) {
    fos.write(file.getBytes());
}

// Запуск YARA
ProcessBuilder pb = new ProcessBuilder( ...command: "C:\\Users\\misho\\Desktop\\yara64.exe", "D:\\Programming\\IntelliJ IDEA
pb.redirectErrorStream(true);
Process process = pb.start();

String result;
try (InputStream is = process.getInputStream()) {
    result = new String(is.readAllBytes());
}

int exitCode = process.waitFor();
if (exitCode != 0 || !result.isBlank()) {
    model.addAttribute( attributeName: "message",  attributeValue: "Malware detected: Upload Failure.");
} else {
    File targetDir = new File( pathname: "D:\\Programming\\IntelliJ IDEA Projects\\thesis\\src\\main\\uploads\\");
    if (!targetDir.exists()) targetDir.mkdirs();
    File savedFile = new File(targetDir, originalFilename);
    file.transferTo(savedFile);
    model.addAttribute( attributeName: "message",  attributeValue: "File uploaded and passed antivirus scan.");
}

tempFile.delete();
return "upload-secure";
```

Рисунок 4.21. Код, що відповідає за логіку

На сторінці також міститься інформація про спосіб, за яким ця вразливість є нейтралізована. Принцип полягає у наступному:

- Захищена реалізація перевіряє розширення файлу та запускає зовнішній аналізатор YARA: yara64.exe rules/malware_rules.yar temp_uploaded_file
- У разі збігу із сигнатурою (наприклад, eval(), файл не зберігається.

Що таке YARA?

YARA є відкритим фреймворком для виявлення та класифікації зловмисного програмного забезпечення на основі текстових і двійкових патернів з використанням набору правил, кожне з яких складається з опису рядків та логічного виразу [45]. Він надає можливість створювати деталізовані правила для пошуку відомих ознак шкідливості у файлах або процесах, подібно до регулярних виразів, але оптимізованих під завдання кібербезпеки [46].

Спробуймо завантажити шкідливий файл (вміст на рисунку 4.18) і не шкідливий (вміст на рисунку 4.22) й перегляньмо результат на рисунку 4.23-4.24.

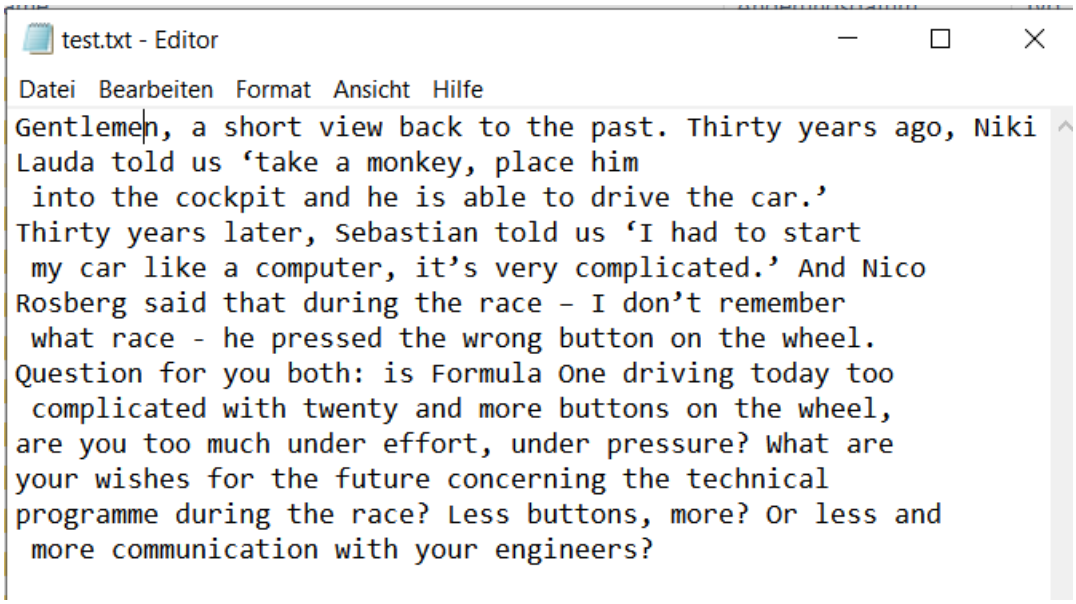

Рисунок 4.22 - Звичайний файл

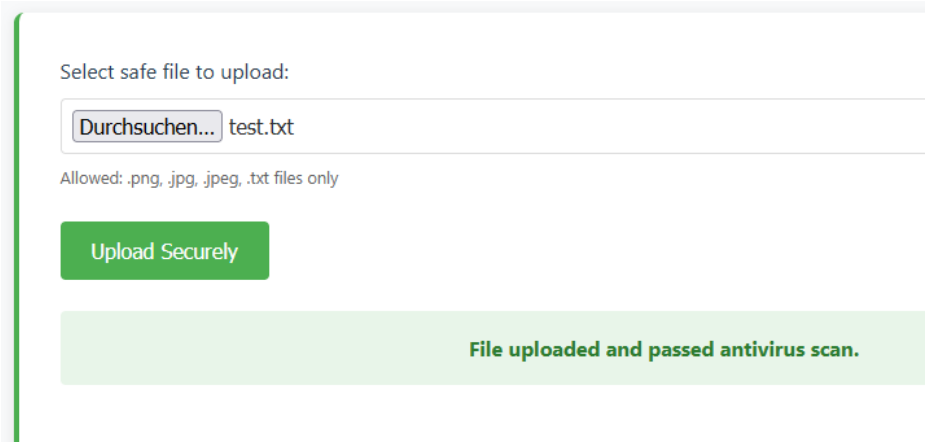

Рисунок 4.23 - Завантаження файлу

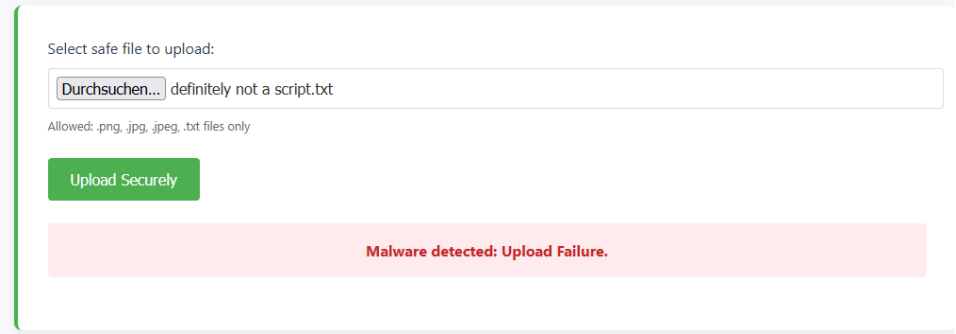

Рисунок 4.24 - Спроба завантаження шкідливого файлу

Як ми бачимо, тільки звичайний файл без шкідливого змісту було завантажено, бо використовувалося правило YARA (додаток Х).

Тепер перейдімо до автентифікації користувачів. Тут було реалізовано два підходи до зберігання паролів: вразливий — з використанням MD5, захищений — через BCrypt.

Обираємо небезпечну версію цієї вразливості. За логіку відповідає метод login-weak-form (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.25, відповідає файл login-weak-form.html (додаток Л). Код методу представлено на рисунку 4.26.

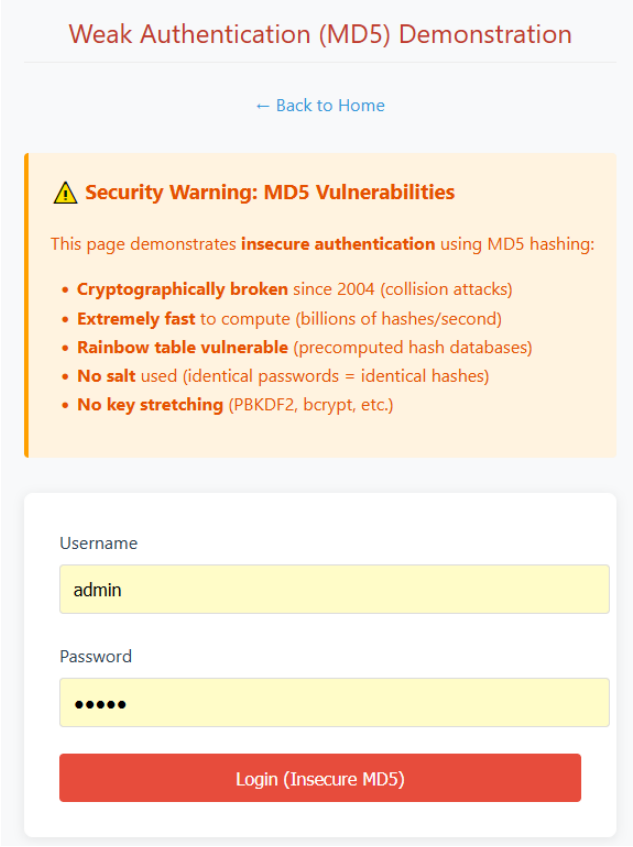

Рисунок 4.25 - Автентифікація - MD5

```
@PostMapping("/login-weak-form")
public String handleLoginWeak(@RequestParam String username,
                              @RequestParam String password,
                              Model model) throws NoSuchAlgorithmException {
    String hashedPassword = hashPasswordMD5(password);
    String query = "SELECT * FROM users_weak WHERE username = ? AND password = ?";
    List<UserWeak> users_weak = jdbcTemplate.query(query, new Object[]{username, hashedPassword
        UserWeak user = new UserWeak();
        user.setUsername(rs.getString( columnLabel: "username"));
        return user;
    });

    if (users_weak.isEmpty()) {
        model.addAttribute( attributeName: "loginError",  attributeValue: true);
    } else {
        model.addAttribute( attributeName: "loginSuccess",  attributeValue: true);
        model.addAttribute( attributeName: "user", users_weak.get(0)); // optional
    }

    return "login-weak-form";
}


1 usage
private String hashPasswordMD5(String password) throws NoSuchAlgorithmException {
    MessageDigest md = MessageDigest.getInstance( algorithm: "MD5");
    byte[] hash = md.digest(password.getBytes());
    StringBuilder sb = new StringBuilder();
    for (byte b : hash) {
        sb.append(String.format("%02x", b));
    }
    return sb.toString();
}
```

Рисунок 4.26 - Код, що відповідає за логіку

На сторінці також міститься інформація про спосіб, за яким ця вразливість є реалізована. Принцип полягає у наступному: Уразлива реалізація порівнює хеш MD5 введеного пароля з записом у БД «String hash = DigestUtils.md5DigestAsHex(password.getBytes());».

Спробуймо увійти (рисунок 4.27).

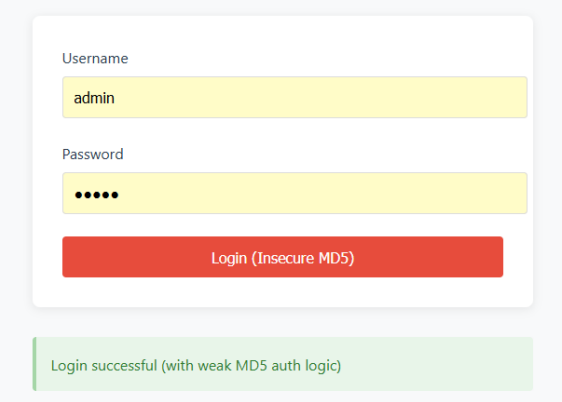

Рисунок 4.27 - Вхід за MD5

Обираємо безпечну версію цієї вразливості. За логіку відповідає метод login-secure (повний код у додатку А). За відображення сторінки, що представлена на рисунку 4.28, відповідає файл login-secure.html (додаток С). Код методу представлено на рисунку 4.29.

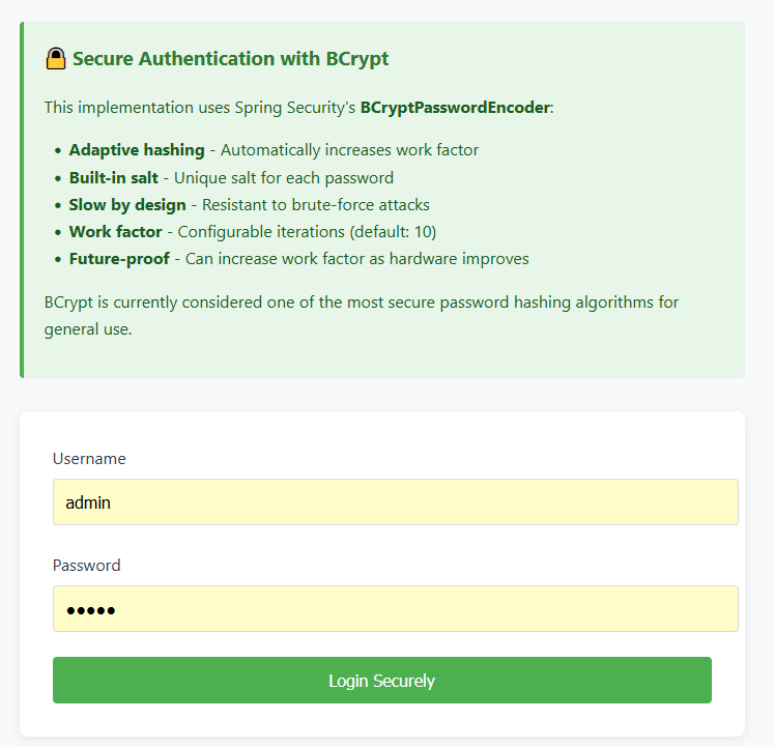

Рисунок 4.28 - Автентифікація - MD5

```
@PostMapping("/secure-login")
public String handleSecureLogin(@RequestParam String username,
                                @RequestParam String password,
                                Model model) {
    String query = "SELECT * FROM users_secure WHERE username = ?";
    List<UserSecure> users = jdbcTemplate.query(query, new Object[]{username}, (rs, rowNum) -> {
        UserSecure user = new UserSecure();
        user.setUsername(rs.getString( columnLabel: "username"));
        user.setPassword(rs.getString( columnLabel: "password"));
        return user;
    });

    if (!users.isEmpty()) {
        UserSecure user = users.get(0);
        if (passwordEncoder.matches(password, user.getPassword())) {
            model.addAttribute( attributeName: "loginSuccess",  attributeValue: true);
        } else {
            model.addAttribute( attributeName: "loginSuccess",  attributeValue: false);
        }
    } else {
        model.addAttribute( attributeName: "loginSuccess",  attributeValue: false);
    }

    return "login-secure-form";
}
```

Рисунок 4.29 - Код, що відповідає за логіку

На сторінці також міститься інформація про спосіб, за яким ця вразливість є нейтралізована. Принцип полягає у наступному: Захищена версія використовує BCryptPasswordEncoder «BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();» - такий підхід формує сольовані хеші, що унеможливлює зворотний підбір.

Спробуймо увійти (рисунок 4.30).

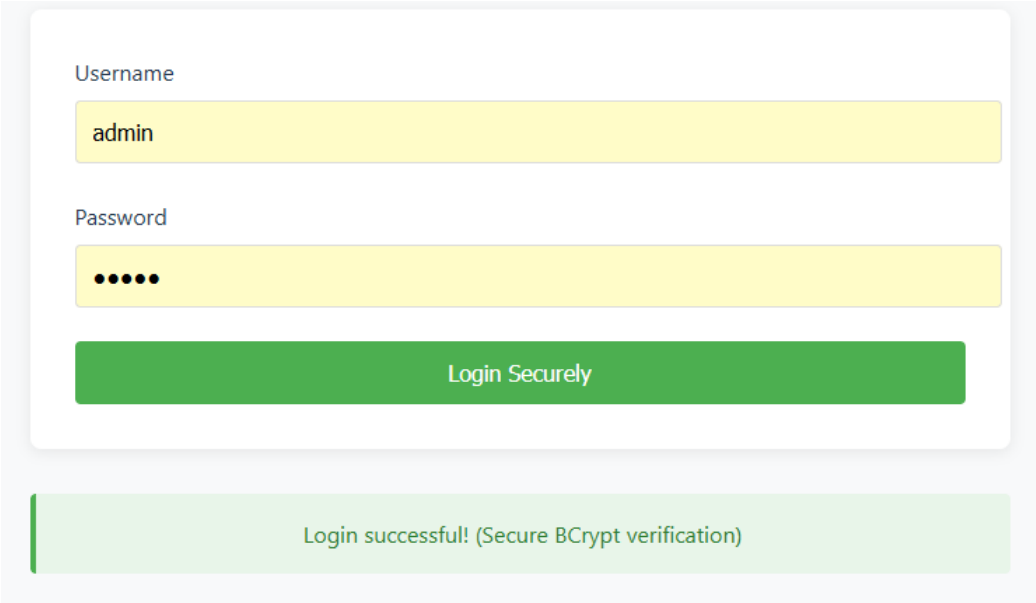

Рисунок 4.30 - Успіх входу

Також для функціонування таблиць БД було створено класи:

Для забезпечення коректної взаємодії вебдодатку з базою даних було реалізовано низку класів, що представляють собою об’єктно-реляційне відображення відповідних таблиць. Клас User моделює сутність користувача, що використовується в експериментальних дослідженнях SQL-запитів та демонстрації потенційної вразливості до SQL-ін’єкцій (див. додаток Д). У межах безпечної обробки автентифікаційних даних було створено два додаткові класи: UserSecure та UserWeak. Перший із них реалізує концепцію безпечного зберігання паролів шляхом застосування стійких алгоритмів хешування, таких як BCrypt (див. додаток Ж), тоді як другий — UserWeak — демонструє приклад небезпечного підходу до зберігання паролів із використанням застарілих або

вразливих алгоритмів хешування, зокрема MD5 (див. додаток Е).

Конфігурація параметрів підключення до бази даних, ініціалізація схеми, а також інші технічні аспекти, пов’язані з запуском та взаємодією з БД, регламентуються в конфігураційному файлі application.properties. Цей файл також визначає джерело даних, шляхи до SQL-скриптів та поведінку при ініціалізації (див. додаток У), що є критично важливим для автоматизованого запуску та коректного функціонування вебдодатку.

Висновки за розділом 4

У межах практичної частини було реалізовано навчальний вебдодаток, який демонструє контраст між уразливими та захищеними підходами до обробки чотирьох критичних загроз: XSS, SQL-ін’єкцій, роботи з файлами та автентифікації. Реалізація спиралася на рекомендації OWASP Top 10 і дозволила наочно перевірити ефективність захисних механізмів у контексті сучасного Java-стека.

Уразливості типу XSS були усунені шляхом використання th:text у шаблонах Thymeleaf замість th:utext, що забезпечує контекстне екранування даних. Для SQL-запитів було продемонстровано, що параметризовані конструкції ефективно блокують можливість виконання ін’єкцій, на відміну від вразливої реалізації з конкатенацією рядків.

Захист при роботі з файлами було реалізовано через фільтрацію розширень та інтеграцію сигнатурного аналізу з YARA, що дозволяє виявляти потенційно шкідливий контент до його збереження. В аспекті автентифікації, використання BCrypt замість MD5 суттєво підвищило захист облікових записів завдяки генерації солі та стійкості до атак перебором.

Отримані результати доводять, що навіть базове впровадження рекомендованих методів захисту суттєво знижує ризик компрометації системи. Такий підхід підтверджує доцільність дотримання принципів «secure by design» при розробці вебдодатків та потребу у регулярному аудиті безпеки.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було проведено ґрунтовне дослідження сучасних підходів до забезпечення безпеки вебдодатків, створених за допомогою технологій Java. Проблематика безпеки в умовах стрімкого розвитку вебсервісів є надзвичайно актуальною, адже з кожним роком збільшується як кількість користувачів онлайн-додатків, так і потенційних загроз, які можуть виникати в результаті дій зловмисників. Вебдодатки дедалі частіше стають мішенню для різноманітних типів атак, що ставить перед розробниками важливе завдання — впроваджувати надійні механізми захисту.

У роботі було проаналізовано основні загрози, притаманні вебдодаткам, зокрема такі як SQL-ін’єкції, міжсайтове виконання скриптів (XSS), міжсайтове підроблення запитів (CSRF), вразливості пов’язані з аутентифікацією та управлінням сесіями, тощо. Детально розглянуто методи їх виявлення, запобігання та нейтралізації в середовищі Java, а також акцентовано увагу на практичних інструментах та фреймворках, зокрема Spring Security, які значно полегшують інтеграцію стандартних методів захисту у процес розробки вебдодатків.

Для досягнення практичної мети дослідження було реалізовано тестовий вебдодаток на Java, який включає типові сценарії потенційної вразливості. Це дало можливість дослідити прикладні аспекти впровадження захисних механізмів і наочно продемонструвати вплив різних типів атак на систему. У процесі експериментальних досліджень було виявлено, що навіть незначні недоліки валідації вхідних даних чи помилки в налаштуванні безпеки можуть призводити до серйозних наслідків. На прикладі модульного тестування були продемонстровані можливості зловмисника при відсутності належного захисту, а також було перевірено ефективність конкретних механізмів протидії.

Результати аналізу довели, що досягнення належного рівня безпеки можливе лише за умови комплексного підходу: поєднання архітектурно

правильного проєктування додатка, використання спеціалізованих фреймворків захисту, впровадження багаторівневої автентифікації, регулярного оновлення компонентів, а також постійного моніторингу активності в системі. Водночас, критично важливо не лише впровадити технічні засоби, а й дотримуватися принципів безпечної розробки — від етапу дизайну системи до супроводу готового продукту.

Таким чином, у роботі виконано всі поставлені завдання:

- Проаналізовано методики створення та структурування вебдодатків на Java, що дозволило чітко визначити джерела потенційних загроз.
- Досліджено механізми захисту та практики кібербезпеки, що застосовуються для попередження критичних вразливостей.
- Розроблено набір рекомендацій, що дозволяють підвищити рівень захищеності вебдодатків, з урахуванням як програмної реалізації, так і специфіки функціонування прикладного програмного забезпечення.

Розроблено тестовий додаток для демонстрації реалізації як вразливих, так і захищених сценаріїв, з метою експериментального аналізу ефективності обраних механізмів безпеки.

Отже, підсумовуючи результати дослідження, можна стверджувати, що жоден окремий метод захисту не здатен гарантувати повну безпеку вебдодатку. Лише використання комплексу взаємодоповнюючих підходів дозволяє значно знизити ризики та забезпечити цілісність, доступність і конфіденційність даних. Виходячи з проведеного аналізу, в роботі було сформовано низку рекомендацій, які можуть бути корисними для розробників та команд програмістів при створенні сучасних і безпечних вебдодатків на Java.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Java EE 7 Tutorial. Oracle. URL: https://docs.oracle.com/javaee/7/tutorial/index.html (дата звернення: 03.06.2025).

2. Java Servlet Technology. Oracle. URL: https://www.oracle.com/technical-resources/articles/javase/servlets-jsp.html (дата звернення: 17.01.2025).

3. Fowler, M. (2019). Patterns of Enterprise Application Architecture. Addison–Wesley.

4. Tudose C. Java Persistence with Spring Data and Hibernate. Manning Publications, 2023. 616 с.

5. Richardson C. Microservices Patterns: With examples in Java. Manning Publications, 2018. 520 с.

6. Long J., Bastani K. Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry. O'Reilly Media, 2017. 648 с.

7. Media. Krochmalski T. Introducing Micronaut: Build, Test, and Deploy Microservices with Oracle Cloud. 1st ed. Apress, 2022. 300 с.

8. Java Platform, Enterprise Edition (Java EE) at a Glance [Електронний ресурс]. Oracle. URL: https://www.oracle.com/java/technologies/java-ee-glance.html (дата звернення: 18.01.2025).

9. Walls C. Spring in Action, Sixth Edition. Manning Publications, 2022. 520 с.

10. Vaadin. URL: https://vaadin.com/docs/latest/building-apps (дата звернення: 17.01.2025).

11. Cross-Site Scripting (XSS) Attacks [Електронний ресурс]. OWASP Foundation. URL: https://owasp.org/www-community/attacks/xss/ (дата звернення: 17.01.2025).

12. Halfond W., Viegas J., Orso A. A Classification of SQL Injection Attacks and Countermeasures. International Symposium on Signals, Systems, and Electronics. 2006. URL:

https://sites.cc.gatech.edu/home/orso/papers/halfond.viegas.orso.ISSSE06.pdf (дата звернення: 18.01.2025).

13. Barth A., Jackson C., Mitchell J. Robust defenses for cross-site request forgery. Proceedings of the 15th ACM Conference on Computer and Communications Security. 2008. С. 75–88. DOI: 10.1145/1455770.1455782. URL: https://seclab.stanford.edu/websec/csrf/csrf.pdf (дата звернення: 18.01.2025).

14. Testing for File Inclusion. OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_File_Inclusion (дата звернення: 20.01.2025).

15. Rajakumaran G., Venkataraman N., Mukkamala R. R. Denial of Service Attack Prediction Using Gradient Descent Algorithm. SN Computer Science. 2020. Т. 1. № 45. DOI: https://doi.org/10.1007/s42979-019-0043-7.

16. Fredj O., Cheikhrouhou O., Krichen M., Hamam H., Derhab A. An OWASP Top Ten Driven Survey on Web Application Protection Methods. Risks and Security of Internet and Systems. Lecture Notes in Computer Science. 2021. С. 235–252. DOI: 10.1007/978-3-030-68887-5_14.

17. Buffer Overflow. OWASP. URL: https://owasp.org/www-community/vulnerabilities/Buffer_Overflow (дата звернення: 24.01.2025).

18. Lee T., Wi S., Lee S., Son S. FUSE: Finding File Upload Bugs via Penetration Testing. Network and Distributed System Security Symposium (NDSS). 2020. DOI: 10.14722/ndss.2020.23126.

19. A05:2021 – Security Misconfiguration. OWASP Top 10:2021. URL: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/ (дата звернення: 24.01.2025).

20. CVE-2019-14900. MITRE CVE. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14900 (дата звернення: 02.02.2025).

21. CVE-2013-6430. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/cve-2013-6430 (дата звернення: 02.02.2025).

22. CVE-2024-4328. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-4328 (дата звернення: 02.02.2025).

23. CVE-2024-2365. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-2365 (дата звернення: 02.02.2025).

24. CVE-2025-30065. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2025-30065 (дата звернення: 02.02.2025).

25. CVE-2025-24813. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2025-24813 (дата звернення: 02.02.2025).

26. CVE-2024-20536. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-20536 (дата звернення: 02.02.2025).

27. CVE-2020-8594. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-8594 (дата звернення: 02.02.2025).

28. CVE-2023-45815. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2023-45815 (дата звернення: 02.02.2025).

29. Apache Struts 2 Remote Code Execution Vulnerability Affecting Multiple Cisco Products: September 2017. Cisco Security Advisory. URL: https://sec.cloudapps.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20170909-struts2-rce (дата звернення: 02.03.2025).

30. Equifax data breach. Wikipedia. URL: https://en.wikipedia.org/wiki/2017_Equifax_data_breach (дата звернення: 03.03.2025).

31. CVE-2018-1273: RCE with Spring Data Commons. Spring. URL: https://spring.io/security/cve-2018-1273 (дата звернення: 03.03.2025).

32. Bulgarian Revenue Agency hack. Wikipedia. URL: https://en.wikipedia.org/wiki/2019_Bulgarian_Revenue_Agency_hack (дата звернення: 03.06.2025).

33. Unveiling CVE-2024-22320: A Novice's Journey to Exploiting Java Deserialization RCE in IBM ODM. Vicarius. URL: https://www.vicarius.io/vsociety/posts/unveiling-cve-2024-22320-a-novices-journey-to-exploiting-java-deserialization-rce-in-ibm-odm (дата звернення: 03.03.2025).

34. Java Virtual Machine Security Features. Oracle. URL: https://docs.oracle.com/en/java/javase/11/security/java-virtual-machine-security-features.html (дата звернення: 03.03.2025).

35. Любчак В. О., Савостян В. В., Козолуп П. Д. Безпека Java-додатків : навчальний посібник / В. О. Любчак, В. В. Савостян, П. Д. Козолуп. – Суми : Сумський державний університет, 2023. – 72 с.

36. O'Connor, F., Nelson, R., & Carter, S. (2021). Testing Java Applications for Security Vulnerabilities: Tools and Techniques. Journal of Cyber Risk Management, 9(3), 40–50.

37. M S, Jasmine & Devi, Kirthiga & George, Geogen. (2017). Detecting XSS Based Web Application Vulnerabilities. International Journal of Computer Technology & Applications. 8. 291-297.

38. Xie, Xin & Ren, Chunhui & Fu, Yusheng & Xu, Jie & Guo, Jinhong. (2019). SQL Injection Detection for Web Applications Based on Elastic-Pooling CNN. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2947527.

39. Ravindran, U., Potukuchi, R.V. (2022). A review on web application vulnerability assessment and penetration testing. Review of Computer Engineering Studies, Vol. 9, No. 1, pp. 1-22. DOI: https://doi.org/10.18280/rces.090101

40. Meng W., Qian C., Hao S. та ін. Rampart: Protecting Web Applications from CPU-Exhaustion DoS Attacks. In: Proceedings of USENIX Security Symposium, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/meng (дата звернення: 03.06.2025).

41. Eka Pratama I. P. A. TCP SYN Flood (DoS) Attack Prevention Using SPI Method on CSF: A PoC. Bulletin of Computer Science and Electrical Engineering. 2020. Vol. 1, no. 2. P. 63–72. DOI: 10.25008/bcsee.v1i2.7.

42. Darus, M. Y., Omar, M. A., Mohamad, M. F., Seman, Z., & Awang, N. (2020). Web vulnerability assessment tool for content management system. International Journal, 9(1.3).

43. Mohammed J., Mehdi S. Web application authentication using ZKP and novel 6D chaotic system. Indonesian Journal of Electrical Engineering and Computer

Science. 2020. Vol. 20, no. 3. P. 1522–1529. DOI: 10.11591/ijeecs.v20.i3.pp1522-1529.

44. Esposito, D., Rennhard, M., Ruf, L., & Wagner, A. Exploiting the Potential of Web Application Vulnerability Scanning. In: Proceedings of ICIMP 2018 – The Thirteenth International Conference on Internet Monitoring and Protection, Spain, July 22–26, 2018. DOI: 10.21256/zhaw-3927.

45. VirusTotal. YARA. URL: https://virustotal.github.io/yara/ (дата звернення: 03.06.2025).

46. Wikipedia. YARA. URL: https://en.wikipedia.org/wiki/YARA (дата звернення: 03.06.2025).

ДОДАТКИ

Додаток А

Повний код класу «MainController» у пакеті «repository»

```
package main;


import
jakarta.servlet.http.HttpServletRequest
;
import main.entity.User;
import main.entity.UserSecure;
import main.entity.UserWeak;
import
org.springframework.beans.factory.annot
ation.Autowired;
import
org.springframework.context.annotation.
Bean;
import
org.springframework.jdbc.core.JdbcTempl
ate;
import
org.springframework.security.crypto.bcr
ypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.pas
sword.PasswordEncoder;
import
org.springframework.stereotype.Controll
er;
import org.springframework.ui.Model;
import main.repository.UserRepository;
import
org.springframework.util.StringUtils;
import
org.springframework.web.bind.annotat
ion.*;
import
org.springframework.web.multipar
t.MultipartFile;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import
java.nio.file.StandardCopyOption
;
import
java.security.MessageDigest;
import
java.security.NoSuchAlgorithmExc
eption;
import java.security.Principal;
import java.util.ArrayList;
import java.util.List;

@Controller
public class MainController {

    @Autowired
    private PasswordEncoder
passwordEncoder;
    @GetMapping("/")
    public String
redirectToLogin() {
        return
"redirect:/login";
    }

    @GetMapping("/login")
    public String login() {
        return "login";
    }
    @GetMapping("/home")
    public String home(Model
model, Principal principal) {

model.addAttribute("username",
principal.getName());
        return "home";
    }
    @GetMapping("/xss")
    public String xssForm() {
        return "xss";
    }
    @PostMapping("/xss")
    public String
handleXss(@RequestParam String
comment, Model model) {

model.addAttribute("comment",
comment);
        return "xss";
    }
    @GetMapping("/xss-
vulnerable")
    public String xssVulnForm()
{
        return "xss-vulnerable";
    }

    @PostMapping("/xss-
vulnerable")
    public String
handleXssVuln(@RequestParam
String comment, Model model) {
          model.addAttribute("co
mment", comment);
        return "xss-vulnerable";
    }
    @Autowired
    private JdbcTemplate
jdbcTemplate;
    @PostMapping("/sql-
injection")
    public String
searchUser(@RequestParam String
username, Model model) {
        String query = "SELECT *
FROM users WHERE username = '" +
username + "'";
        List<User> users =
jdbcTemplate.query(query, (rs,
rowNum) -> {
            User user = new
User();

user.setUsername(rs.getString("u
sername"));

user.setEmail(rs.getString("emai
l"));
            return user;
        });

System.out.println(users);

model.addAttribute("results",
users);
        return "sql-injection";
    }

    @GetMapping("/sql-
injection")
    public String showForm(Model
model) {

model.addAttribute("results",
new ArrayList<>());
        return "sql-injection";
    }
    @GetMapping("/debug/users")
    @ResponseBody
    public String listUsers() {
        List<User> users =
jdbcTemplate.query("SELECT *
FROM users", (rs, rowNum) -> {
            User user = new
User();

user.setUsername(rs.getString("u
sername"));

user.setEmail(rs.getString("emai
l"));
            return user;
        });
        return users.toString();
    }
    @PostMapping("/sql-
injection-safe")
    public String
searchUserSafe(@RequestParam
String username, Model model) {
        String query = "SELECT *
FROM users WHERE username = ?";
        List<User> users =
jdbcTemplate.query(query, new
Object[]{username}, (rs, rowNum)
-> {
            User user = new
User();

user.setUsername(rs.getString("u
sername"));

user.setEmail(rs.getString("emai
l"));
            return user;
        });

model.addAttribute("results",
users);
        return "sql-injection-
safe";
    }
    @GetMapping("/sql-injection-
safe")
    public String
showSafeForm(Model model) {

model.addAttribute("results",
new ArrayList<>());
        return "sql-injection-
safe";
    }
    @GetMapping("/login-weak-
form")
    public String
loginWeakForm() {
        return "login-weak-
form";
    }
    @PostMapping("/login-weak-
form")
    public String
handleLoginWeak(@RequestParam
String username,

@RequestParam String password,

Model model) throws
NoSuchAlgorithmException {
        String hashedPassword =
hashPasswordMD5(password);
        String query = "SELECT *
FROM users_weak WHERE username
= ? AND password = ?";
        List<UserWeak>
users_weak =
jdbcTemplate.query(query, new
Object[]{username,
hashedPassword}, (rs, rowNum) ->
{
            UserWeak user = new
UserWeak();

user.setUsername(rs.getString("u
sername"));

            return user;

        });
        if
(users_weak.isEmpty()) {

model.addAttribute("loginError",
true);
        } else {

model.addAttribute("loginSuccess
", true);

model.addAttribute("user",
users_weak.get(0));
        }

        return "login-weak-
form";
    }
    private String
hashPasswordMD5(String password)
throws NoSuchAlgorithmException
{
        MessageDigest md =
MessageDigest.getInstance("MD5")
;
        byte[] hash =
md.digest(password.getBytes());
        StringBuilder sb = new
StringBuilder();
        for (byte b : hash) {

sb.append(String.format("%02x",
b));
        }
        return sb.toString();
    }
    @GetMapping("/login-secure")
    public String
showSecureLoginForm() {
        return "login-secure-
form";
    }

    @PostMapping("/secure-
login")
    public String
handleSecureLogin(@RequestParam
String username,

@RequestParam String password,

Model model) {
        String query = "SELECT *
FROM users_secure WHERE username
= ?";
        List<UserSecure> users =
```

Продовження додатку А

```
jdbcTemplate.query(query, new
Object[]{username}, (rs, rowNum)

          -> {


            UserSecure user =
new UserSecure();

user.setUsername(rs.getString("u
sername"));

user.setPassword(rs.getString("p
assword"));
            return user;
        });

        if (!users.isEmpty()) {
            UserSecure user =
users.get(0);
            if
(passwordEncoder.matches(passwor
d, user.getPassword())) {

model.addAttribute("loginSuccess
", true);
            } else {

model.addAttribute("loginSuccess
", false);
            }
        } else {

model.addAttribute("loginSuccess
", false);
        }

        return "login-secure-
form";
    }
    @GetMapping("/upload-
vulnerable")
    public String
showUploadVulnerable() {
        return "upload-
vulnerable";
    }
    @RequestMapping(value =
"/upload-vulnerable", method =
RequestMethod.POST)
    public String
handleInsecureFileUpload(@Reques
tParam("file") MultipartFile
file, Model model) {
        if (file.isEmpty()) {

model.addAttribute("uploadMessag
e", "No file selected.");
            return "upload-
vulnerable";
        }

        try {
            File dest = new
File("D:\\Programming\\IntelliJ
IDEA
Projects\\thesis\\src\\main\\upl
oads\\" +
file.getOriginalFilename());
file.transferTo(dest);

model.addAttribute("uploadMessag
e", "File uploaded
successfully");
        } catch (IOException e)
{



          model.addAttribute("up
loadMessage", "Error uploading
file");
        }

        return "upload-
vulnerable";
    }
    @GetMapping("/upload-
secure")
    public String
showSecureUploadForm() {
        return "upload-secure";
    }
    @PostMapping("/upload-
secure")
    public String
handleSecureUpload(@RequestParam
("file") MultipartFile file,

Model model,

HttpServletRequest request)
throws IOException,
InterruptedException {

        if (file.isEmpty()) {

model.addAttribute("message",
"Please select a file to
upload");
            return "upload-
secure";
        }
        String originalFilename
=
StringUtils.cleanPath(file.getOr
iginalFilename());
        String extension =
StringUtils.getFilenameExtension
(originalFilename);

        if (extension == null
|| !(extension.equalsIgnoreCase(
"txt") ||
extension.equalsIgnoreCase("jpg"
) ||
extension.equalsIgnoreCase("png"
))) {

model.addAttribute("message",
"Unsupported file type");
            return "upload-
secure";

                }

        File tempFile =
File.createTempFile("upload_",
"_" + originalFilename);
        try (FileOutputStream
fos = new
FileOutputStream(tempFile)) {

fos.write(file.getBytes());
        }

        ProcessBuilder pb = new
ProcessBuilder("C:\\Users\\misho
\\Desktop\\yara64.exe",
"D:\\Programming\\IntelliJ IDEA
Projects\\thesis\\src\\main\\yar
a\\SimpleYaraRule.yara",
tempFile.getAbsolutePath());

pb.redirectErrorStream(true);
        Process process =
pb.start();

        String result;
        try (InputStream is =
process.getInputStream()) {
            result = new
String(is.readAllBytes());
        }
        int exitCode =
process.waitFor();
        if (exitCode != 0
|| !result.isBlank()) {

model.addAttribute("message",
"Malware detected: Upload
Failure.");
        } else {
            File targetDir = new
File("D:\\Programming\\IntelliJ
IDEA
Projects\\thesis\\src\\main\\upl
oads\\");
            if
(!targetDir.exists())
targetDir.mkdirs();
            File savedFile = new
File(targetDir,
originalFilename);

file.transferTo(savedFile);

model.addAttribute("message",
"File uploaded and passed
antivirus scan.");
        }

        tempFile.delete();
        return "upload-secure";
    }

}
```

Додаток Б

Повний код класу «SecurityConfig» у пакеті «repository»

```
package main;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;


@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .csrf().disable() // 🔧 Вимикає CSRF
                .authorizeHttpRequests()
                .requestMatchers("/login", "/css/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .defaultSuccessUrl("/home", true)
                .permitAll()
                .and()
                .logout()
                .permitAll();
        return http.build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        return new InMemoryUserDetailsManager(

User.withUsername("user").password("$2a$12$tdlS63AjX2cobUL3Uu1Twuu6uCd/DWlEmWfUZH88FBt.cF8
pxcHDi").roles("USER").build(),

User.withUsername("admin").password("$2a$12$ecci2D/3cM6ouL/oBCm74eO9Uhv/Ql52jrfnYaVWL5OOAq
eKKtyla").roles("ADMIN").build()
        );
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Додаток В

Повний код класу «SecurityDemoApplication» у пакеті «repository»

```
package main;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SecurityDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SecurityDemoApplication.class, args);
    }
}
```

Додаток Д

Повний код класу «User» у пакеті «entity»

```
      package main.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String email;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    @Override
    public String toString() {
        return username + " (" + email + ")";
    }

}
```

Додаток Е

Повний код класу «UserSecure» у пакеті «entity»

```
      package main.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "users_secure")
public class UserSecure {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    @Override
    public String toString() {
        return username  + password;
    }

}
```

Додаток Ж

Повний код класу «UserWeak» у пакеті «entity»

```
package main.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "users_weak")
public class UserWeak {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    @Override
    public String toString() {
        return username  + password;
    }

}
```

Додаток 3

Повний код «home.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Web Security Demonstration</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color: #f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
            display: flex;
            flex-direction: column;
            justify-content: center;
        }
        h1 {
            color: #2c3e50;
            text-align: center;
            margin-bottom: 1.5rem;
            font-weight: 300;
        }
        p {
            text-align: center;
            color: #7f8c8d;
            margin-bottom: 2rem;
        }

        nav {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            padding: 1.5rem;
            margin-bottom: 2rem;
        }

        ul {
            list-style-type: decimal;
            padding-left: 1.5rem;
        }

        li {
            margin-bottom: 0.8rem;
            padding-left: 0.5rem;
        }

        a {
            color: #3498db;
            text-decoration: none;
            transition: color 0.2s;
        }

        a:hover {
            color: #2980b9;
            text-decoration: underline;
        }



        .logout-form {
            text-align: center;
        }

        button {
            background-color: #e74c3c;
            color: white;
            border: none;
            padding: 0.6rem 1.2rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            transition: background-color 0.2s;
        }

        button:hover {
            background-color: #c0392b;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid #ecf0f1;
            margin-top: auto;
        }
    </style>
</head>
<body>
<div class="container">
    <h1>Welcome, <span th:text="${username}" style="font-weight: 500;">Username</span>!</h1>
    <p>This is a secured page for cybersecurity research demonstration</p>

    <nav>
        <ul>
            <li><a th:href="@{/xss}">Cross-Site Scripting (Safe Implementation)</a></li>
            <li><a th:href="@{/xss-vulnerable}">XSS (Vulnerable Implementation)</a></li>
            <li><a th:href="@{/sql-injection-safe}">SQL Injection (Safe Implementation)</a></li>
            <li><a th:href="@{/sql-injection}">SQL Injection (Vulnerable Implementation)</a></li>
            <li><a th:href="@{/login-weak-form}">Login Weak Form</a></li>
            <li><a th:href="@{/login-secure}">Login Secure Form</a></li>
            <li><a th:href="@{/upload-vulnerable}">Vulnerable File Upload</a></li>
            <li><a th:href="@{/upload-secure}">Secure File Upload</a></li>
        </ul>
    </nav>

    <div class="logout-form">
        <form th:action="@{/logout}" method="post">
            <button type="submit">Logout</button>
        </form>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's Thesis in Cybersecurity | Web Application Security Demonstration | 2025
</footer>
</body>
</html>
```

Додаток И

Повний код «login.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf
.org">
<head>
    <title>Secure Login |
Cybersecurity Demo</title>
    <style>
        body {
            font-family:
'Segoe UI', Tahoma, Geneva,
Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color:
#f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction:
column;
            min-height: 100vh;
            justify-content:
center;
        }

        .container {
            max-width: 500px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
            display: flex;
            flex-direction:
column;
            justify-content:
center;
        }

        h2 {
            color: #2c3e50;
            text-align:
center;
            margin-bottom:
2rem;
            font-weight: 300;
            font-size: 1.8rem;
            letter-spacing: -
0.5px;
        }

        .login-card {
            background-color:
white;
            border-radius:
8px;
            box-shadow: 0 4px
12px rgba(0, 0, 0, 0.08);
            padding: 2.5rem;
            margin-bottom:
1.5rem;
            position:
relative;
            overflow: hidden;
            width: 100%;
            max-width: 520px;
        }
        .login-card::before {
            content: "⚡";
            position:
absolute;
            top: 10px;
            right: 15px;
            font-size: 1.2rem;
            opacity: 0.1;
        }
        .form-group {
            margin-bottom:
1.5rem;
        }

        label {
            display: block;
            margin-bottom:
0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }

        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid
#ddd;
            border-radius:
4px;
            font-size: 1rem;
            transition:
border-color 0.2s;
            margin: 0;
        }

        input:focus {
            border-color:
#3498db;
            outline: none;
            box-shadow: 0 0 0
2px rgba(52, 152, 219, 0.2);
        }

        .btn-submit {
            width: 100%;
            background-color:
#3498db;
            color: white;
            border: none;
            padding: 0.8rem;
            border-radius:
4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition:
background-color 0.2s;
            margin-top:
0.5rem;
        }

        .btn-submit:hover {
            background-color:
#2980b9;
        }

        .alert-wrapper {
            min-height: 56px;
            margin-bottom:
1rem;
            position:
relative;
        }

        .alert {
            padding: 0.8rem;
            border-radius:
4px;
            text-align:
center;
        }

        .alert-spacer {
            visibility:
hidden;
            height: 56px;
        }


        .alert-error {
            background-color:
#fdecea;
            color: #c62828;
            border: 1px solid
#ef9a9a;
        }

        .alert-success {
            background-color:
#e8f5e9;
            color: #2e7d32;
            border: 1px solid
#a5d6a7;
        }

        footer {
            text-align:
center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px
solid #ecf0f1;
            margin-top: auto;
        }
        .alert, .form-group
input {
            width: 100%;
        }

        .container {
            display: flex;
            flex-direction:
column;
            justify-content:
center;
            align-items:
center;
            min-height: 100vh;
        }
    </style>
</head>
<body>
<div class="container">
    <div class="login-card">
        <h2>Secure Access
Portal</h2>

        <div class="alert-
wrapper">
            <div
th:if="${param.error}"
class="alert alert-
error">Invalid username or
password</div>
            <div
th:if="${param.logout}"
class="alert alert-
success">You have been logged
out successfully</div>
            <div
th:if="${param.error == null
and param.logout == null}"
class="alert-spacer"></div>
        </div>


        <form
th:action="@{/login}"
method="post">
            <div class="form-
group">
                <label
for="username">Username</label
>
                <input
type="text" id="username"
name="username" required>
            </div>

            <div class="form-
group">
                <label
for="password">Password</label
>
                <input
type="password" id="password"
name="password" required>
            </div>

            <button
type="submit" class="btn-
submit">Authenticate</button>
        </form>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's
Thesis in Cybersecurity | Web
Application Security
Demonstration | 2025
</footer>
</body>
</html>
```

Додаток К

Повний код «login-secure-form.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.o
rg">
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.o
rg">
<head>
    <title>Secure Login |
Cybersecurity Demo</title>
    <style>
        body {
            font-family: 'Segoe
UI', Tahoma, Geneva, Verdana,
sans-serif;
            line-height: 1.6;
            color: #333;
            background-color:
#f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction:
column;
            min-height: 100vh;
        }

        .container {
            max-width: 700px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #2e7d32;
            text-align: center;
            margin-bottom:
1.5rem;
            font-weight: 400;
            border-bottom: 1px
solid #e0e0e0;
            padding-bottom:
0.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #3498db;
            text-decoration:
none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration:
underline;
        }

        .security-info {
            background-color:
#e8f5e9;
            border-left: 4px
solid #4caf50;
            color: #1b5e20;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .security-info h3 {
            margin-top: 0;
            color: #2e7d32;
        }

        .security-info ul {
            padding-left:
1.5rem;
        }
        .demo-card {
            background-color:
white;
            border-radius: 8px;
            box-shadow: 0 2px
10px rgba(0, 0, 0, 0.08);
            padding: 2rem;
            margin-bottom: 2rem;
        }

        .form-group {
            margin-bottom:
1.5rem;
        }

        label {
            display: block;
            margin-bottom:
0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }

        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid
#ddd;
            border-radius: 4px;
            font-size: 1rem;
            transition: border-
color 0.2s;
        }

        input:focus {
            border-color:
#4caf50;
            outline: none;
            box-shadow: 0 0 0
2px rgba(76, 175, 80, 0.2);
        }

        .btn-submit {
            background-color:
#4caf50;
            color: white;
            border: none;
            padding: 0.8rem
1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition:
background-color 0.2s;
            width: 100%;
        }

        .btn-submit:hover {
            background-color:
#388e3c;
        }

        .alert {
            padding: 1rem;
            border-radius: 4px;
            margin: 1.5rem 0;
            text-align: center;
        }

        .alert-success {
            background-color:
#e8f5e9;
            color: #2e7d32;
            border-left: 4px
solid #4caf50;
        }

        .alert-error {
            background-color:
#ffebee;
            color: #c62828;
            border-left: 4px
solid #f44336;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px
solid #ecf0f1;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>Secure Login (Fixed
Authentication)</h2>

    <div class="nav-back">
        <a th:href="@{/home}">←
Back to Home</a>
    </div>

    <div class="security-info">
        <h3>🔒 Secure
Authentication with BCrypt</h3>
        <p>This implementation
uses Spring Security's
<strong>BCryptPasswordEncoder</s
trong>:</p>
        <ul>
            <li><strong>Adaptive
hashing</strong> - Automatically
increases work factor</li>
            <li><strong>Built-in
salt</strong> - Unique salt for
each password</li>
            <li><strong>Slow by
design</strong> - Resistant to
brute-force attacks</li>
            <li><strong>Work
factor</strong> - Configurable
iterations (default: 10)</li>
            <li><strong>Future-
proof</strong> - Can increase
work factor as hardware
improves</li>
        </ul>
        <p>BCrypt is currently
considered one of the most
secure password hashing
algorithms for general use.</p>
    </div>

    <div class="demo-card">
        <form
th:action="@{/secure-login}"
method="post">
            <div class="form-
group">
                <label
for="username">Username</label>
                <input
type="text" id="username"
name="username" required>
            </div>

            <div class="form-
group">
                <label
for="password">Password</label>
                <input
type="password" id="password"
name="password" required>
            </div>

            <button
type="submit" class="btn-
submit">Login Securely</button>
        </form>
    </div>

    <div
th:if="${loginSuccess != null}">
        <div
th:if="${loginSuccess}"
class="alert alert-success">
            Login successful!
(Secure BCrypt verification)
        </div>
        <div
th:if="${!loginSuccess}"
class="alert alert-error">
            Invalid username or
password
        </div>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's
Thesis in Cybersecurity | Web
Application Security
Demonstration | 2025
</footer>
</body>
</html>
```

Додаток Л

Повний код «login-weak-form.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf
.org">
<head>
    <title>Weak Authentication
(MD5) | Security Demo</title>
    <style>
        body {
            font-family:
'Segoe UI', Tahoma, Geneva,
Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color:
#f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction:
column;
            min-height: 100vh;
        }

        .container {
            max-width: 700px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #c0392b;
            text-align:
center;
            margin-bottom:
1.5rem;
            font-weight: 400;
            border-bottom: 1px
solid #eee;
            padding-bottom:
0.5rem;
        }

        .nav-back {
            text-align:
center;
            margin-bottom:
2rem;
        }

        .nav-back a {
            color: #3498db;
            text-decoration:
none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration:
underline;
        }

        .demo-card {
            background-color:
white;
            border-radius:
8px;
            box-shadow: 0 2px
10px rgba(0, 0, 0, 0.08);
            padding: 2rem;
            margin-bottom:
2rem;
        }

        .form-group {
            margin-bottom:
1.5rem;
        }

        label {
            display: block;
            margin-bottom:
0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }
        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid
#ddd;
            border-radius:
4px;
            font-size: 1rem;
        }



        .btn-submit {
            background-color:
#e74c3c;
            color: white;
            border: none;
            padding: 0.8rem
1.5rem;
            border-radius:
4px;
            cursor: pointer;
            font-size: 1rem;
            transition:
background-color 0.2s;
            width: 100%;
        }

        .btn-submit:hover {
            background-color:
#c0392b;
        }

        .alert {
            padding: 1rem;
            border-radius:
4px;
            margin: 1.5rem 0;
        }

        .alert-error {
            background-color:
#fdecea;
            color: #c62828;
            border-left: 4px
solid #ef9a9a;
        }

        .alert-success {
            background-color:
#e8f5e9;
            color: #2e7d32;
            border-left: 4px
solid #a5d6a7;
        }

        .security-warning {
            background-color:
#fff3e0;
            border-left: 4px
solid #ffa000;
            color: #e65100;
            padding: 1.25rem;
            border-radius:
4px;
            margin: 2rem 0;
        }

        .security-warning h3 {
            margin-top: 0;
            color: #e65100;
        }

        .security-warning ul {
            padding-left:
1.5rem;
        }

        footer {
            text-align:
center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px
solid #ecf0f1;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>Weak Authentication
(MD5) Demonstration</h2>

    <div class="nav-back">
        <a
th:href="@{/home}">← Back to
Home</a>
    </div>

    <div class="security-
warning">
        <h3>⚠ Security
Warning: MD5
Vulnerabilities</h3>
        <p>This page
demonstrates <strong>insecure
authentication</strong> using
MD5 hashing:</p>
        <ul>

<li><strong>Cryptographically
broken</strong> since 2004
(collision attacks)</li>


<li><strong>Extremely
fast</strong> to compute
(billions of
hashes/second)</li>



<li><strong>Rainbow table
vulnerable</strong>
(precomputed hash
databases)</li>
            <li><strong>No
salt</strong> used (identical
passwords = identical
hashes)</li>
            <li><strong>No key
stretching</strong> (PBKDF2,
bcrypt, etc.)</li>
        </ul>
    </div>

    <div class="demo-card">
        <form
th:action="@{/login-weak-
form}" method="post">
            <div class="form-
group">
                <label
for="username">Username</label
>
                <input
type="text" id="username"
name="username" required>
            </div>

            <div class="form-
group">
                <label
for="password">Password</label
>
                <input
type="password" id="password"
name="password" required>
            </div>

            <button
type="submit" class="btn-
submit">Login (Insecure
MD5)</button>
        </form>
    </div>

    <div th:if="${loginError}"
class="alert alert-error">
        Invalid username or
password (weak auth check)
    </div>

    <div
th:if="${loginSuccess}"
class="alert alert-success">
        Login successful (with
weak MD5 auth logic)
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's
Thesis in Cybersecurity | Web
Application Security
Demonstration | 2025
</footer>
</body>
</html>
```

Додаток М

Повний код «sql-injection.html» у пакеті «templates»

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>SQL Injection Vulnerability Demo |
                Cybersecurity</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva,
                Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color: #f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #c62828;
            text-align: center;
            margin-bottom: 1.5rem;
            font-weight: 400;
            border-bottom: 1px solid #ffcdd2;
            padding-bottom: 0.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #c62828;
            text-decoration: none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration: underline;
        }

        .danger-warning {
            background-color: #ffebee;
            border-left: 4px solid #c62828;
            color: #b71c1c;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .danger-warning h3 {
            margin-top: 0;
            color: #c62828;
        }

        .demo-card {
            background-color: white;
            border-radius: 8px;


            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.08);
            padding: 2rem;
            margin-bottom: 2rem;
            border: 1px solid #ffcdd2;
        }

        .form-group {
            margin-bottom: 1.5rem;
        }

        label {
            display: block;
            margin-bottom: 0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }

        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid #ddd;
            border-radius: 4px;
            font-size: 1rem;
            transition: border-color 0.2s;
        }

        input:focus {
            border-color: #c62828;
            outline: none;
            box-shadow: 0 0 0 2px rgba(198, 40, 40, 0.2);
        }

        .btn-submit {
            background-color: #c62828;
            color: white;
            border: none;
            padding: 0.8rem 1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition: background-color 0.2s;
        }

        .btn-submit:hover {
            background-color: #b71c1c;
        }

        .results-panel {
            background-color: #ffebee;
            border-radius: 4px;
            padding: 1.5rem;
            margin-top: 1.5rem;
            border-left: 4px solid #c62828;
        }

        .results-panel h3 {
            margin-top: 0;
            color: #c62828;
        }

        .results-list {
            list-style-type: none;
            padding: 0;


        }

        .results-list li {
    background-color: white;
            padding: 0.8rem;
            margin-bottom: 0.5rem;
            border-radius: 4px;
            border: 1px solid #ffcdd2;
        }

        .no-results {
            color: #757575;
            font-style: italic;
            text-align: center;
            padding: 1rem;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid #ecf0f1;
        }

        .code {
            font-family: 'Courier New', monospace;
            background-color: #f5f5f5;
            padding: 2px 4px;
            border-radius: 3px;
            font-size: 0.9em;
        }

        .danger-icon {
            color: #c62828;
            font-size: 1.2em;
            vertical-align: middle;
            margin-right: 5px;
        }

        .payload-example {
            background-color: #f5f5f5;
            padding: 10px;
            border-radius: 4px;
            margin: 10px 0;
            font-family: monospace;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>⚠ SQL Injection Vulnerability
                Demonstration</h2>

    <div class="nav-back">
        <a th:href="@{/home}">← Back to Home</a>
    </div>

    <div class="danger-warning">
        <h3><span class="danger-icon">☢</span>
                Critical Security Warning</h3>
        <p>This page <strong>deliberately contains an
                SQL injection
                vulnerability</strong> for
                demonstration purposes:</p>
        <ul>
            <li>Uses <strong>string concatenation</strong>
                in SQL queries instead of prepared
                statements</li>
            <li>Allows <strong>arbitrary SQL
                execution</strong> through user
                input</li>
            <li>Demonstrates how attackers could
                <strong>extract, modify, or delete
                data</strong></li>
            <li>Controller passes input <strong>directly to
                vulnerable query</strong>:
                <pre style="background: #f5f5f5; padding:
                10px; border-radius: 4px; margin-
                top: 5px;">
@PostMapping("/sql-injection")
public String handleVulnerableQuery(@RequestParam
                String username, Model model) {
    // UNSAFE - concatenates input directly into query!

    String query = "SELECT * FROM users WHERE
                username = '" + username + "'";

    model.addAttribute("results", results);
    return "sql-injection";
}</pre>
            </li>
        </ul>
        <div class="payload-example">
            <strong>Example malicious
                payloads:</strong><br>
            <code>' OR '1'='1</code> - Returns all
                users<br>
            <code>'; DROP TABLE users; --</code> -
                Deletes table (if permissions
                allow)<br>
            <code>' UNION SELECT password FROM
                users; --</code> - Extracts sensitive
                data
        </div>
        <p><strong>Never</strong> concatenate user
                input directly into SQL queries in
                production!</p>
    </div>

    <div class="demo-card">
        <form th:action="@{/sql-injection}"
                method="post">
            <div class="form-group">
                <label for="username">Try SQL injection
                payload:</label>
                <input type="text" id="username"
                name="username"
                th:value="${param.username}"
                placeholder="e.g., ' OR '1'='1">
            </div>

            <button type="submit" class="btn-
                submit">Demonstrate SQL
                Injection</button>
        </form>

        <div class="results-panel">
            <h3>Database Results (Vulnerable):</h3>
            <div th:if="${#lists.isEmpty(results)}"
                class="no-results">
                No users found matching the criteria
            </div>
            <ul th:if="${not #lists.isEmpty(results)}"
                class="results-list">
                <li th:each="user : ${results}"
                th:text="${user.username + ' (' + user.email
                + ')'}">
                User info
                </li>
            </ul>
            <p style="color: #c62828; font-size: 0.9rem;
                margin-top: 0.5rem;">
                ⚠ Warning: This query is vulnerable to SQL
                injection - Extremely dangerous in
                production!
            </p>
        </div>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's Thesis in Cybersecurity |
                Web Application Security
                Demonstration | 2025
</footer>
</body>
</html>
```

Додаток Н

Повний код «sql-injection-safe.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.org">
<head>
    <title>SQL Injection Protection Demo
| Cybersecurity</title>
    <style>
        body {
            font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color: #f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #2e7d32;
            text-align: center;
            margin-bottom: 1.5rem;
            font-weight: 400;
            border-bottom: 1px solid
#e0e0e0;
            padding-bottom: 0.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #2e7d32;
            text-decoration: none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration: underline;
        }

        .security-info {
            background-color: #e8f5e9;
            border-left: 4px solid
#4caf50;
            color: #1b5e20;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .security-info h3 {
            margin-top: 0;
            color: #2e7d32;
        }

        .demo-card {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px
rgba(0, 0, 0, 0.08);
            padding: 2rem;
            margin-bottom: 2rem;
        }

        .form-group {
            margin-bottom: 1.5rem;
        }


        label {

            display: block;
            margin-bottom: 0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }

        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid #ddd;
            border-radius: 4px;
            font-size: 1rem;
            transition: border-color
0.2s;
        }
        input:focus {
            border-color: #4caf50;
            outline: none;
            box-shadow: 0 0 0 2px
rgba(76, 175, 80, 0.2);
        }

        .btn-submit {
            background-color: #4caf50;
            color: white;
            border: none;
            padding: 0.8rem 1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition: background-color
0.2s;
        }

        .btn-submit:hover {
            background-color: #388e3c;
        }

        .results-panel {
            background-color: #f5f5f5;
            border-radius: 4px;
            padding: 1.5rem;
            margin-top: 1.5rem;
            border-left: 4px solid
#607d8b;
        }

        .results-panel h3 {
            margin-top: 0;
            color: #455a64;
        }

        .results-list {
            list-style-type: none;
            padding: 0;
        }

        .results-list li {
            background-color: white;
            padding: 0.8rem;
            margin-bottom: 0.5rem;
            border-radius: 4px;
            border: 1px solid #e0e0e0;
        }

        .no-results {
            color: #757575;
            font-style: italic;
            text-align: center;
            padding: 1rem;
        }
        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid
#ecf0f1;

                }


                .code {
                    font-family:
 'Courier New', monospace;
                    background-color:
 #f5f5f5;
                    padding: 2px 4px;
                    border-radius: 3px;
                    font-size: 0.9em;
                }
                .shield-icon {
                    color: #4caf50;
                    font-size: 1.2em;
                    vertical-align:
 middle;
                    margin-right: 5px;
                }
    </style>
</head>
<body>
<div class="container">
    <h2>SQL Injection Protection
 Demonstration</h2>
    <div class="nav-back">
        <a th:href="@{/home}">←
 Back to Home</a>
    </div>
    <div class="security-info">
        <h3><span class="shield-
 icon">🛡️</span> Secure SQL
 Practices</h3>
        <p>This implementation
demonstrates <strong>protection
against SQL injection</strong>
through:</p>
        <ul>
            <li><strong>Prepared
statements</strong>: Uses
parameterized queries to
separate SQL logic from
data</li>
            <li><strong>ORM
protection</strong>: Spring Data
JPA automatically sanitizes
inputs</li>
            <li><strong>Input
validation</strong>: Additional
validation could be implemented
at service layer</li>

<li><strong>Principle of least
privilege</strong>: Database
user has limited
permissions</li>
        </ul>
        <p>Example safe
repository method:</p>
        <pre style="background:
#f5f5f5; padding: 10px; border-
radius: 4px;">
@Query("SELECT u FROM User u
WHERE u.username = :username")
List&lt;User&gt;
findByUsername(@Param("username"
) String username);</pre>
    </div>

    <div class="demo-card">
        <form th:action="@{/sql-
injection-safe}" method="post">
            <div class="form-
group">
                <label
for="username">Search username
(safe):</label>
                <input
type="text" id="username"
name="username"
th:value="${param.username}"

placeholder="Try ' OR '1'='1
(won't work)">
            </div>

            <button
type="submit" class="btn-
submit">Search Securely</button>
        </form>

        <div class="results-
panel">
            <h3>Database
Results:</h3>
            <div
th:if="${#lists.isEmpty(results)
}" class="no-results">
                No users found
matching the criteria
            </div>
            <ul th:if="${not
#lists.isEmpty(results)}"
class="results-list">
                <li
th:each="user : ${results}"

th:text="${user.username + ' ('
+ user.email + ')'}">
                    User info
                </li>
            </ul>
            <p style="color:
#2e7d32; font-size: 0.9rem;
margin-top: 0.5rem;">
                ✔ Safe from SQL
injection - malicious inputs are
treated as data, not code
            </p>
        </div>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's
Thesis in Cybersecurity | Web
Application Security
Demonstration | 2025
</footer>
</body>
</html>
```

Додаток П

Повний код «upload-secure.html» у пакеті «templates»

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Secure File Upload | Cybersecurity Demo</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color: #f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        .container {
            max-width: 1000px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
            width: 100%;
        }

        h2 {
            color: #2e7d32;
            text-align: center;
            margin-bottom: 1.5rem;
            font-weight: 400;
            border-bottom: 1px solid #e0e0e0;
            padding-bottom: 0.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #2e7d32;
            text-decoration: none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration: underline;
        }

        .demo-card {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.
            padding: 2rem;
            margin-bottom: 2rem;
            border-left: 4px solid #4caf50;
        }

        .form-group {
            margin-bottom: 1.5rem;
        }

        label {
            display: block;
            margin-bottom: 0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }
        input[type="file"] {
            width: 100%;
            padding: 0.5rem;
            border: 1px solid #ddd;


            border-radius: 4px;


        font-size: 1rem;
        }

        .btn-submit {
            background-color: #4caf50;
            color: white;
            border: none;
            padding: 0.8rem 1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition: background-color 0.2s;
        }

        .btn-submit:hover {
            background-color: #388e3c;
        }

        .message {
            padding: 1rem;
            border-radius: 4px;
            margin: 1.5rem 0;
            text-align: center;
            font-weight: bold;
        }

        .success {
            background-color: #e8f5e9;
            color: #2e7d32;
        }

        .error {
            background-color: #ffebee;
            color: #c62828;
        }

        .code {
            font-family: 'Courier New', monospace;
            background-color: #f5f5f5;
            padding: 10px;
            border-radius: 4px;
            overflow-x: auto;
            font-size: 0.9em;
        }

        .comparison {
            display: flex;
            flex-wrap: wrap;
            gap: 2rem;
            justify-content: center; /* Center the columns */
            margin: 2rem 0;
        }

        .column {
            flex: 1;
            min-width: 300px; /* Ensure columns don shrink too small */
            max-width: 45%; /* Control the max size the columns */
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.
            padding: 1.5rem;
        }

        .security {
            border-left: 4px solid #4caf50;
        }

        .yara {
            border-left: 4px solid #2196f3;
        }

        h3 {
            color: #2e7d32;
            margin-top: 0;
        }



        .yara h3 {
            color: #2196f3;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid #ecf0f1;
        }

        /* Media query for smaller screens */
        @media (max-width: 768px) {
            .comparison {
                flex-direction: column;
                align-items: center;
            }

            .column {
                max-width: 100%;
                min-width: 90%; /* Ensure it doesn' take the full screen on mobile */
            }
        }
    </style>
</head>
<body>
<div class="container">
    <h2>Secure File Upload Demonstration</h2>

    <div class="nav-back">
        <a th:href="@{/home}">← Back to Home</a>
    </div>

    <div class="demo-card">
        <form th:action="@{/upload-secure}" method="post" enctype="multipart/form-data">
            <div class="form-group">
                <label for="file">Select safe file upload:</label>
                <input type="file" id="file" name="file" accept=".png,.jpg,.jpeg,.txt" required>
                <p style="font-size: 0.8rem; color: #666; margin-top: 0.5rem;">
                    Allowed: .png, .jpg, .jpeg, .txt files only
                </p>
            </div>

            <button type="submit" class="btn-submit">Upload Securely</button>
        </form>

        <div th:if="${message != null}" class="message" th:classappend="${message.contains('uploaded') ? 'success' : 'error'}">
            <span th:text="${message}"></span>
        </div>
    </div>

    <div class="comparison">
        <div class="column security">
            <h3>Security Measures</h3>
            <ul>
                <li><strong>Extension Whitelisting</strong> - Only .png, .jpg, .txt allowed</li>
                <li><strong>File Renaming</strong> - Uses cleanPath() to prevent path traversal</li>
                <li><strong>Temp File Handling</strong> - Scans files before permanent storage</li>
                <li><strong>Secure Storage</strong> - Dedicated upload directory with proper permissions</li>
            </ul>

            <h3>Controller Logic</h3>
            <pre class="code">@PostMapping("/upload-secure")
public String handleSecureUpload(
        @RequestParam("file") MultipartFile file,
        Model model) throws IOException {
    String extension = getExtension(file);
    if (!ALLOWED_EXTENSIONS.contains(extension)) {
        model.addAttribute("message", "Unsupported file type");
        return "upload-secure";
    }




    if (scanWithYara(file)) {
        model.addAttribute("message", "Malware detected");
        return "upload-secure";
    }


    saveFile(file);
    model.addAttribute("message", "File uploaded safely");
    return "upload-secure";
}</pre>
        </div>

        <div class="column yara">
            <h3>YARA Malware Detection</h3>
            <p>Files are scanned using this YARA rule:</p>
            <pre class="code">rule SimpleMalwareRule {
    meta:
        description = "Detects basic malicious patterns"
        author = "Oleh Mishnov"

    strings:
        $script = "&lt;script&gt;"
        $eval = "eval("

    condition:
        any of them
}</pre>

            <h3>Detection Capabilities</h3>
            <ul>
                <li><strong>Script Tags</strong> - Detects embedded HTML/JS scripts</li>
                <li><strong>Eval Statements</strong> - Identifies dangerous JavaScript eval()</li>
                <li><strong>Expandable</strong> - More rules can be added as needed</li>
                <li><strong>Fast Scanning</strong> - Executes during upload process</li>
            </ul>
        </div>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's Thesis in Cybersecurity | Web Application Security Demonstration | 2025
</footer>
</body>
</html>
```

Додаток Р

Повний код «upload-vulnerable.html» у пакеті «templates»

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>File Upload Vulnerability | Cybersecurity Demo</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color: #f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #c62828;
            text-align: center;
            margin-bottom: 1.5rem;
            font-weight: 400;
            border-bottom: 1px solid #ffcdd2;
            padding-bottom: 0.5rem;
        }

        h3 {
            color: #c62828;
            margin-top: 1.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #c62828;
            text-decoration: none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration: underline;
        }

        .danger-warning {
            background-color: #ffebee;
            border-left: 4px solid #c62828;
            color: #b71c1c;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .danger-warning h3 {
            margin-top: 0;
            color: #c62828;
        }

        .demo-card {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.08);



            padding: 2rem;
            margin-bottom: 2rem;
            border: 1px solid #ffcdd2;

                }

        .form-group {
            margin-bottom: 1.5rem;
        }

        label {
            display: block;
            margin-bottom: 0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }

        input[type="file"] {
            width: 100%;
            padding: 0.5rem;
            border: 1px solid #ddd;
            border-radius: 4px;
            font-size: 1rem;
        }

        .btn-submit {
            background-color: #c62828;
            color: white;
            border: none;
            padding: 0.8rem 1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition: background-color 0.2s;
        }

        .btn-submit:hover {
            background-color: #b71c1c;
        }

        .alert-message {
            padding: 1rem;
            border-radius: 4px;
            margin: 1.5rem 0;
        }

        .alert-danger {
            background-color: #ffebee;
            color: #c62828;
            border-left: 4px solid #c62828;
        }

        .vulnerability-details {
            background-color: #fff3e0;
            border-left: 4px solid #ffa000;
            color: #e65100;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .vulnerability-details h3 {
            color: #e65100;
        }

        .vulnerability-details ul {
            padding-left: 1.5rem;
        }

        .code {
            font-family: 'Courier New', monospace;
            background-color: #f5f5f5;
            padding: 2px 4px;
            border-radius: 3px;
            font-size: 0.9em;
        }

        .danger-icon {
            color: #c62828;


                    font-size: 1.2em;
            vertical-align: middle;
            margin-right: 5px;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid #ecf0f1;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>⚠ Insecure File Upload Demonstration</h2>

    <div class="nav-back">
        <a th:href="@{/home}">← Back to Home</a>
    </div>

    <div class="danger-warning">
        <h3><span class="danger-icon">☢</span> Critical Security Warning</h3>
        <p>This page <strong>deliberately contains file upload vulnerabilities</strong> for demonstration purposes:</p>
        <ul>
            <li>Accepts files <strong>without extension validation</strong></li>
            <li>Stores files with <strong>original filenames</strong> (allowing executable extensions)</li>
            <li>No <strong>content-type verification</strong> or file scanning</li>
            <li>Files are saved to a <strong>predictable location</strong> with direct web access</li>
        </ul>
    </div>
    <div class="demo-card">
        <form th:action="@{/upload-vulnerable}" method="post" enctype="multipart/form-data">
            <div class="form-group">
                <label for="file">Try uploading a malicious file:</label>
                <input type="file" id="file" name="file" required>
                <p style="font-size: 0.8rem; color: #666; margin-top: 0.5rem;">
                    Example: .jsp file with &lt;% Runtime.getRuntime().exec("malicious command") %&gt;
                </p>
            </div>

            <button type="submit" class="btn-submit">Upload (Insecure)</button>
        </form>

        <div th:if="${uploadMessage != null}" class="alert-message alert-danger" th:text="${uploadMessage}">
            <!-- Upload message will appear here -->
        </div>
    </div>
    <div class="vulnerability-details">
        <h3>Vulnerable Backend Code</h3>
        <pre style="background: #f5f5f5; padding: 10px; border-radius: 4px; overflow-x: auto;">
@PostMapping("/upload-vulnerable")
public String handleInsecureFileUpload(@RequestParam("file") MultipartFile file, Model model) {
    // UNSAFE - No file validation!
    File dest = new File("uploads/" + file.getOriginalFilename()); // Dangerous filename usage
    file.transferTo(dest); // Direct save without checks
    model.addAttribute("uploadMessage", "File uploaded to: " + dest.getAbsolutePath());
    return "upload-vulnerable";
}</pre>
        <h3>How to Secure File Uploads</h3>
        <ul>
            <li><strong>Whitelist extensions</strong> (.jpg, .png, .pdf only)</li>
            <li><strong>Verify content types</strong> (check MIME types match extensions)</li>
            <li><strong>Rename files</strong> (use UUIDs instead of original names)</li>
            <li><strong>Scan for malware</strong> (use antivirus libraries)</li>
            <li><strong>Store outside web root</strong> or restrict permissions</li>
            <li><strong>Set size limits</strong> (prevent denial of service)</li>
        </ul>
    </div>
</div>
<footer>
    Oleh Mishnov | Bachelor's Thesis in Cybersecurity | Web Application Security Demonstration | 2025
</footer>
</body>
</html>
```

Додаток С

Повний код «xss-vulnerable.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.org">
<head>
    <title>XSS Vulnerability Demo |
Cybersecurity</title>
    <style>
        body {
            font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color:
#f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #c62828;
            text-align: center;
            margin-bottom: 1.5rem;
            font-weight: 400;
            border-bottom: 1px solid
#ffcdd2;
            padding-bottom: 0.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #c62828;
            text-decoration: none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration:
underline;
        }

        .danger-warning {
            background-color:
#ffebee;
            border-left: 4px solid
#c62828;
            color: #b71c1c;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .danger-warning h3 {
            margin-top: 0;
            color: #c62828;
        }

        .demo-card {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px
rgba(0, 0, 0, 0.08);
            padding: 2rem;
            margin-bottom: 2rem;
            border: 1px solid
#ffcdd2;
        }
        .form-group {
            margin-bottom: 1.5rem;

        }



        label {
            display: block;
            margin-bottom: 0.5rem;
            color: #2c3e50;
            font-weight: 500;
        }

        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid #ddd;
            border-radius: 4px;
            font-size: 1rem;
            transition: border-color
0.2s;
        }

        input:focus {
            border-color: #c62828;
            outline: none;
            box-shadow: 0 0 0 2px
rgba(198, 40, 40, 0.2);
        }

        .btn-submit {
            background-color:
#c62828;
            color: white;
            border: none;
            padding: 0.8rem 1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition: background-
color 0.2s;
        }

        .btn-submit:hover {
            background-color:
#b71c1c;
        }

        .output-panel {
            background-color:
#ffebee;
            border-radius: 4px;
            padding: 1.5rem;
            margin-top: 1.5rem;
            border-left: 4px solid
#c62828;
        }

        .output-panel h3 {
            margin-top: 0;
            color: #c62828;
        }

        .output-content {
            background-color: white;
            padding: 1rem;
            border-radius: 4px;
            border: 1px solid
#ffcdd2;
            min-height: 20px;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid
#ecf0f1;
        }

        .code {
            font-family: 'Courier
New', monospace;
            background-color:
#f5f5f5;
            padding: 2px 4px;
            border-radius: 3px;
            font-size: 0.9em;
        }

        .danger-icon {


          color: #c62828;
            font-size: 1.2em;
            vertical-align: middle;
            margin-right: 5px;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>⚠ XSS Vulnerability
Demonstration</h2>

    <div class="nav-back">
        <a th:href="@{/home}">← Back
to Home</a>
    </div>

    <div class="danger-warning">
        <h3><span class="danger-
icon">☢</span> Critical Security
Warning</h3>
        <p>This page
<strong>deliberately contains an XSS
vulnerability</strong> for
demonstration purposes:</p>
        <ul>
            <li>Uses <span
class="code">th:utext</span> instead
of <span class="code">th:text</span>
which <strong>does not escape
HTML</strong></li>
            <li>Allows <strong>raw
HTML/JS execution</strong> from user
input</li>
            <li>Demonstrates how
<strong>malicious scripts</strong>
could steal cookies or redirect
users</li>
            <li>Controller passes
input <strong>without
sanitization</strong>:
                <pre
style="background: #f5f5f5; padding:
10px; border-radius: 4px; margin-top:
5px;">
@PostMapping("/xss-vulnerable")
public String
handleVulnerableXss(@RequestParam
String comment, Model model) {
    model.addAttribute("comment",
comment); // UNSAFE - no escaping!
    return "xss-vulnerable";
}</pre>
            </li>
        </ul>
        <p><strong>Never use</strong>
<span class="code">th:utext</span>
with untrusted input in
production!</p>
    </div>

    <div class="demo-card">
        <form th:action="@{/xss-
vulnerable}" method="post">
            <div class="form-group">
                <label
for="comment">Try XSS payload (will
execute):</label>
                <input type="text"
id="comment" name="comment"
placeholder='e.g.,
<script>alert("Hacked!")</script>'>
            </div>

            <button type="submit"
class="btn-submit">Demonstrate XSS
Attack</button>
        </form>

        <div class="output-panel">
            <h3>Vulnerable
Output:</h3>
            <div class="output-
content" th:utext="${comment} ?: 'No
comment submitted'">
                No comment submitted
            </div>
            <p style="color: #c62828;
font-size: 0.9rem; margin-top:
0.5rem;">
                ⚠ Warning: This
output directly interprets HTML/JS -
Extremely dangerous in production!
            </p>
        </div>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's Thesis
in Cybersecurity | Web Application
Security Demonstration | 2025
</footer>
</body>
</html>
```

Додаток Т

Повний код «xss.html» у пакеті «templates»

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.or
g">
<head>
    <title>XSS Protection Demo |
Cybersecurity</title>
    <style>
        body {
            font-family: 'Segoe
UI', Tahoma, Geneva, Verdana,
sans-serif;
            line-height: 1.6;
            color: #333;
            background-color:
#f8f9fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction:
column;
            min-height: 100vh;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 2rem;
            flex: 1;
        }

        h2 {
            color: #1565c0;
            text-align: center;
            margin-bottom:
1.5rem;
            font-weight: 400;
            border-bottom: 1px
solid #e0e0e0;
            padding-bottom:
0.5rem;
        }

        .nav-back {
            text-align: center;
            margin-bottom: 2rem;
        }

        .nav-back a {
            color: #1565c0;
            text-decoration:
none;
            font-weight: 500;
        }

        .nav-back a:hover {
            text-decoration:
underline;
        }

        .security-info {
            background-color:
#e3f2fd;
            border-left: 4px
solid #1976d2;
            color: #0d47a1;
            padding: 1.25rem;
            border-radius: 4px;
            margin: 2rem 0;
        }

        .security-info h3 {
            margin-top: 0;
            color: #1565c0;
        }

        .demo-card {
            background-color:
white;
            border-radius: 8px;
            box-shadow: 0 2px
10px rgba(0, 0, 0, 0.08);
            padding: 2rem;
            margin-bottom: 2rem;
        }

        .form-group {
            margin-bottom:
1.5rem;
        }
        label {


          display: block;
            margin-bottom:
0.5rem;
            color: #2c3e50;
            font-weight: 500;

 </html>
        }

        input {
            width: 100%;
            padding: 0.75rem;
            border: 1px solid
#ddd;
            border-radius: 4px;
            font-size: 1rem;
            transition: border-
color 0.2s;
        }

        input:focus {
            border-color:
#1976d2;
            outline: none;
            box-shadow: 0 0 0 2px
rgba(25, 118, 210, 0.2);
        }

        .btn-submit {
            background-color:
#1976d2;
            color: white;
            border: none;
            padding: 0.8rem
1.5rem;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1rem;
            font-weight: 500;
            transition:
background-color 0.2s;
        }

        .btn-submit:hover {
            background-color:
#1565c0;
        }

        .output-panel {
            background-color:
#f5f5f5;
            border-radius: 4px;
            padding: 1.5rem;
            margin-top: 1.5rem;
            border-left: 4px
solid #607d8b;
        }

        .output-panel h3 {
            margin-top: 0;
            color: #455a64;
        }

        .output-content {
            background-color:
white;
            padding: 1rem;
            border-radius: 4px;
            border: 1px solid
#e0e0e0;
            min-height: 20px;
        }

        footer {
            text-align: center;
            padding: 1.5rem;
            color: #7f8c8d;
            font-size: 0.9rem;
            border-top: 1px solid
#ecf0f1;
        }

        .code {
            font-family: 'Courier
New', monospace;
            background-color:
#f5f5f5;
            padding: 2px 4px;
            border-radius: 3px;
            font-size: 0.9em;
        }
    </style>
</head>

<body><div class="container">



    <h2>Cross-Site Scripting
(XSS) Protection Demo</h2>

    <div class="nav-back">
        <a th:href="@{/home}">←
Back to Home</a>
    </div>

    <div class="security-info">
        <h3>🛡️ How This Page
Prevents XSS Attacks</h3>
        <p>This implementation
demonstrates <strong>automatic
XSS protection</strong>
through:</p>
        <ul>

<li><strong>Thymeleaf's HTML
escaping</strong>: All dynamic
content in <span
class="code">th:text</span>
attributes is automatically
escaped</li>

<li><strong>Contextual
encoding</strong>: Special
characters are converted to HTML
entities (e.g., <span
class="code">&lt;</span> becomes
<span
class="code">&amp;lt;</span>)</li>
            <li><strong>Safe by
default</strong>: Unlike direct
<span
class="code">innerHTML</span>
insertion, Thymeleaf prevents
script execution</li>
        </ul>
        <p>The controller code
simply passes the input through
without manual sanitization
because Thymeleaf handles it:</p>
        <pre style="background:
#f5f5f5; padding: 10px; border-
radius: 4px;">
@PostMapping("/xss")
public String
handleXss(@RequestParam String
comment, Model model) {
    model.addAttribute("comment",
comment); // Thymeleaf will
escape this
    return "xss";
}</pre>
    </div>

    <div class="demo-card">
        <form th:action="@{/xss}"
method="post">
            <div class="form-
group">
                <label
for="comment">Try XSS payload
(will be neutralized):</label>
                <input
type="text" id="comment"
name="comment" placeholder="e.g.,
<script>alert(1)</script>">
            </div>

            <button type="submit"
class="btn-submit">Test XSS
Protection</button>
        </form>

        <div class="output-
panel">
            <h3>Sanitized
Output:</h3>
            <div class="output-
content" th:text="${comment} ?:
'No comment submitted'">
                No comment
submitted
            </div>
            <p style="color:
#666; font-size: 0.9rem; margin-
top: 0.5rem;">
                Note: The output
above shows how your input would
be safely rendered in a real
application.
            </p>
        </div>
    </div>
</div>

<footer>
    Oleh Mishnov | Bachelor's
Thesis in Cybersecurity | Web
Application Security
Demonstration | 2025
</footer>
</body>
```

Додаток У

Повний код «application.properties» у пакеті «resources»

```
spring.application.name=thesis
spring.sql.init.mode=always
spring.jpa.defer-datasource-initialization=true
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB
```

Додаток Ф

Повний код «schema.sql» у пакеті «resources»

```
      -- Schema for SQL Injection demonstration
CREATE TABLE IF NOT EXISTS users (
                        id INT PRIMARY KEY AUTO_INCREMENT,
                        username VARCHAR(50) NOT NULL,
                        email VARCHAR(100) NOT NULL
);

-- Sample data
INSERT INTO users (username, email) VALUES ('alice', 'alice@example.com');
INSERT INTO users (username, email) VALUES ('bob', 'bob@example.com');
INSERT INTO users (username, email) VALUES ('charlie', 'charlie@example.com');


-- Schema for Weak Authentication demo
CREATE TABLE IF NOT EXISTS users_weak (
                            id INT PRIMARY KEY AUTO_INCREMENT,
                            username VARCHAR(50) NOT NULL,
                            password VARCHAR(32) NOT NULL
);

-- Sample weak user-- Weak password hashes (md5 of 'password')
INSERT INTO users_weak (username, password) VALUES ('admin',
'482c811da5d5b4bc6d497ffa98491e38');
INSERT INTO users_weak (username, password) VALUES ('admin1',
'5f4dcc3b5aa765d61d8327deb882cf99');
INSERT INTO users_weak (username, password) VALUES ('user1',
'e10adc3949ba59abbe56e057f20f883e');


CREATE TABLE IF NOT EXISTS users_secure (
                                          id INT PRIMARY KEY AUTO_INCREMENT,
                                          username VARCHAR(50) NOT NULL,
                                          password VARCHAR(32) NOT NULL
);

-- Sample weak user-- Weak password hashes (md5 of 'password')
-- Sample data (password is 'password' hashed with BCrypt)
INSERT INTO users_secure (username, password) VALUES ('alice',
'$2a$12$Biv1ZhtRp6L6rRrjSTNvYOn61I/TF3BNJwy0TAIXDuVRgIaEfvtzO');
INSERT INTO users_secure (username, password) VALUES ('bob',
'$2a$12$Biv1ZhtRp6L6rRrjSTNvYOn61I/TF3BNJwy0TAIXDuVRgIaEfvtzO');
INSERT INTO users_secure (username, password) VALUES ('charlie',
'$2a$12$Biv1ZhtRp6L6rRrjSTNvYOn61I/TF3BNJwy0TAIXDuVRgIaEfvtzO');
```

Додаток Х

Повний код «SimpleYaraRule.yara» у пакеті «yara»

```
rule SimpleMalwareRule
{
    meta:
        description = "Simple malware detection rule"
        author = "Oleh Mishnov"
        last_modified = "2025-05-13"

    strings:
            $script = "<script>"
            $eval = "eval("
    condition:
            any of them
}
```