Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
« » червня 2021р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Засоби та механізми захисту веб-додатків»

Виконавець: студент IV курсу, групи КБ-42

_____ **Макаренко Антон Олександрович**
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Пархоменко І. І.	

Нормоконтроль	Зюбіна Р. В.	

Київ 2021

Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
« » жовтня 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентові **КБ-42** **Макаренку Антону Олександровичу**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Засоби та механізми захисту веб-додатків

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Структури, архітектури, засоби функціонування веб-додатків, стек технологій для розробки веб-додатків, алгоритми хешування та аутентифікації

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Нормативно-правова база у сфері захисту інформації, структура веб-додатків, мережевий протокол прикладного рівня, архітектурний стиль побудови веб-додатків, основні вразливості веб-додатків, захист від загроз порушеної аутентифікації, SQL-ін'єкцій, ескалації привілеїв, рекомендації

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Поєднання механізмів захисту веб-додатків та формування рекомендацій щодо їх впровадження.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видав		І. І. Пархоменко
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		А. О. Макаренко
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	25.01.2021 – 27.01.2021	*виконано*
2	Аналіз літератури	28.01.2021 – 11.02.2021	*виконано*
3	Розгляд структури веб-додатків	12.02.2021 – 24.02.2021	*виконано*
4	Дослідження основних вразливостей	25.02.2021 – 24.03.2021	*виконано*
5	Вибір технологічного стеку	25.03.2021 – 07.04.2021	*виконано*
6	Впровадження засобів та механізмів захисту від загроз порушеної аутентифікації	08.04.2021 – 20.04.2021	*виконано*
7	Впровадження засобів та механізмів захисту від SQL-ін’єкцій	21.04.2021 – 05.05.2021	*виконано*
8	Впровадження засобів та механізмів захисту від ескалації привілеїв	06.05.2021 – 20.05.2021	*виконано*
9	Формування рекомендацій щодо механізмів захисту для веб-додатків	21.05.2021 – 04.06.2021	*виконано*
10	Оформлення пояснювальної записки	05.06.2021 – 08.06.2021	*виконано*
11	Підготовка до захисту	09.06.2021 – 21.06.2021	*виконано*

Завдання видав		І. І. Пархоменко
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		А. О. Макаренко
	(підпис)	(ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 61 сторінку, включає в себе зміст, вступ, три розділи дипломної роботи, висновки та список джерел. Крім того, робота містить 3 додатки із загальною кількістю сторінок 14. У пояснювальній записці дипломної роботи міститься 17 рисунків і 2 таблиці.

Метою роботи є реалізація засобів та механізмів захисту веб-додатків.

Для досягнення зазначеної мети поставлено наступні завдання:

- дослідити структуру веб-додатків
- провести аналіз найбільш поширених вразливостей, характерних для визначеної структури
- побудувати веб-додаток із використанням програмних засобів та механізмів захисту

Об’єктом дослідження є процес виявлення та протидії загрозам, що властиві сучасним веб-додаткам.

Предметом дослідження є набір механізмів, що реалізують методи захисту веб-додатків.

Практичною цінністю отриманих результатів є поєднання та програмна реалізація засобів та механізмів захисту веб-додатків.

Ключові слова: веб-додаток, база даних, вразливості, захист персональних даних, облікові дані, SQL-ін’єкції, ескалація привілеїв.

ЗМІСТ

ВСТУП

Сучасне життя неможливо уявити без веб-додатків. Вони використовуються скрізь: від перегляду смішних відео з котиками в Інтернеті до великих корпоративних мереж. Дані, що циркулюють каналами зв'язку, включають в себе особисту інформацію користувачів, таку як електронна пошта, номери мобільних телефонів, паролі, паспортні дані, дані про реєстрацію тощо. Розкриття цієї інформації зловмисниками загрожує фінансовими та репутаційними збитками для підприємств, втратою особистих даних та доступу до ресурсів для звичайних користувачів.

В умовах дистанційного навчання та роботи об'єм трафіку, що циркулює каналами зв'язку, значно зріс, порівнюючи зі звичайним режимом роботи і навчання. При таких обставинах зростають ризики, пов'язані з цілістю, конфіденційністю та доступністю інформації. На жаль, деякі компанії задумуються про безпеку та впроваджують необхідні міри лише після того, як відбувся інцидент інформаційної безпеки. До числа таких компаній можна віднести «Zoom Video Communications». У 2020 році тисячі відеодзвінків, зроблених за допомогою програми «Zoom», що розроблена і підтримується вищезгаданою компанією, опинилися у відкритому доступі [1]. Не варто забувати і про розважальну частину веб-простору. Через вразливість в ігровому рушії «Source», що належить компанії «Valve Corporation», зловмисники були здатні віддалено виконувати зловмисний код [2].

Отже, безпека веб-додатків є актуальним завданням сьогодення. Ціллю атаки можуть бути як корпоративні мережі, що містять цінну інформацію для бізнесу, так і звичайні користувачі, що користуються веб-послугами задля розваг.

Апробація результатів дипломної роботи відбулася на IV Міжнародній науково-практичній «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» (PCSITS) [3].

РОЗДІЛ 1
СТРУКТУРА ТА МЕХАНІЗМИ ВЗАЄМОДІЇ ВЕБ-ДОДАТКІВ

1.1 Нормативно-правова база у сфері захисту інформації

Говорячи про веб-додатки, що надають послуги на території України, варто у першу чергу згадати Конституцію України. Стаття 31 Конституції України гарантує таємницю листування, телефонних розмов чи іншої кореспонденції [4]. Веб-додатки зобов'язані робити листування чи інший обмін інформацією таємним, а вміст електронних повідомлень недоступним для всіх, окрім учасників листування.

Стаття 32 визначає, що ніхто не має права втручатися у чуже особисте і сімейне життя, крім випадків, передбачених Конституцією України. Не допускається збирання, зберігання, використання та поширення конфіденційної інформації про особу без згоди, крім випадків, визначених законом, і лише в інтересах національної безпеки, економічного добробуту та прав людини [4]. Веб-додатки не мають права збирати інформацію про особисте і сімейне життя користувачів без їхньої згоди. Обробка конфіденційної інформації користувачів має бути лише за їх згоди. Інакше веб-додатку забороняється збір, обробка, зберігання і поширення конфіденційних даних.

Закон України «Про інформацію» визначає терміни і поняття, пов'язані з інформацією і інформаційними відносинами [5]. Закон визначає різницю між документом і інформацією. Так, інформація – відомості чи дані, що можуть бути збережені на матеріальних носіях, а документ – сам матеріальний носій, що зберігає інформацію.

Закон визначає основні принципи інформаційних відносин. Основними принципами є:

- гарантованість права на інформацію
- відкритість, доступність інформації, свобода обміну інформацією
- достовірність і повнота інформації
- свобода вираження поглядів і переконань

- правомірність одержання, використання, поширення, зберігання та захисту інформації
- захищеність особи від втручання в її особисте та сімейне життя

Суб’єктами інформаційних відноси є фізичні особи, юридичні особи, об'єднання громадян, суб'єкти владних повноважень. Об’єктом інформаційних відносин є інформація. З точки зору веб-додатку суб’єктами інформаційних відносин є користувачі веб-додатку, власники веб-додатку і державні органи, що регулюють їхні відносини. Об’єктом є інформація, що циркулює у веб-додатку.

Закон класифікує інформацію з обмеженим доступом на конфіденційну, таємну та службову. Конфіденційною є інформація про фізичну особу; інформація, доступ до якої обмежено фізичною або юридичною особою, крім державних органів. Конфіденційна інформація може поширюватися за бажанням (згодою) відповідної особи у визначеному нею порядку. Для веб-додатку конфіденційною є інформація про паспортні дані користувачів, ідентифікаційні номери платників податків, адреси проживання тощо. До інформації з обмеженим доступом не може бути внесена інформація про стан довкілля; якість харчових продуктів і предметів побуту; аварії, катастрофи, небезпечні природні явища та інші надзвичайні ситуації, що загрожують безпеці людей; стан здоров'я населення, його життєвий рівень; факти порушення прав і свобод людини, включаючи інформацію із архівних документів; незаконні дії органів державної влади, органів місцевого самоврядування, їх посадових та службових осіб; іншу інформацію, що регулюється законами України і міжнародними домовленостями.

Закон України «Про захист персональних даних» регулює правові відносини, пов’язані із захистом і обробкою персональних даних і спрямований на захист права людини на невтручання у особисте життя [6]. Закон визначає персональні дані як відомості чи сукупність відомостей про фізичну особу, яка ідентифікована або може бути конкретно ідентифікована. Термін «суб’єкт персональних даних» означає фізичну особу, чиї особисті дані обробляються. З точки зору веб-додатку суб’єктом персональних даних є користувач системи. Термін «володілець персональних даних» означає фізичну або юридичну особу, яка визначає мету обробки

персональних даних, встановлює склад цих даних та процедури їх обробки. Для веб-додатку розпорядником персональних даних є людина чи компанія, яка володіє веб-додатком, що надає послуги користувачам і збирає особисті дані за їхньої згоди.

Закон визначає права суб'єкта обробки персональних даних, особливості повідомлення про обробку персональних даних, використання персональних даних, підстави для обробки персональних даних, дії щодо збирання, накопичення та зберігання, поширення, видалення чи знищення персональних даних, порядок доступу до персональних даних, умови відстрочення або відмови у доступі до персональних даних, оскарження рішення про відстрочення або відмову у доступі до персональних даних, процес повідомлення про дії з персональними даними, забезпечення захисту персональних даних, відповідальність за порушення законодавства.

Положення «Про організацію заходів із забезпечення інформаційної безпеки в банківській системі України» встановлює обов'язкові мінімальні вимоги щодо організації заходів із забезпечення інформаційної безпеки та кіберзахисту; принципи управління інформаційною безпекою; вимоги до інформаційних систем банку, що взаємодіють з інформаційними системами Національного банку України, з урахуванням напрямів розвитку криптографічного захисту інформації в інформаційних системах Національного банку [7]. Документ містить вимоги щодо системи управління інформаційною безпекою (СУІБ), криптографічного захисту інформації та інших заходів безпеки щодо інформаційних систем. Для веб-додатків, чия діяльність не пов'язана із інформаційними системами національного банку, документ може розглядатися як рекомендація із покращення захищеності системи.

Стандарт NIST 800-63B визначає вимоги для федеральних агентств з упровадження служб цифрової ідентифікації [8]. Для веб-додатків, чия робота не стосується діяльності федеральних агентств, стандарт може бути використаний як набір рекомендацій щодо аутентифікації суб'єктів інформаційної діяльності.

Стандарт RFC 2616: Hypertext Transfer Protocol – HTTP/1.1 визначає діяльність, способи застосування протоколу HTTP версії 1.1 [9]. Документ визначає HTTP як загальний протокол прикладного рівня для обміну інформацією у форматі

гіпертексту. Протокол може бути використаний не лише для гіпертекстових даних за рахунок розширення його методів, кодів помилок, заголовків.

Стандарт ISO 27000 «Інформаційні технології – Методи і засоби забезпечення безпеки – Системи управління інформаційною безпекою – Огляд і термінологія» визначає основні терміни інформаційної безпеки [10], що будуть використовуватися у даній роботі.

1.2 Загальна структура веб-додатків

Архітектура веб-додатків базується на клієнт-серверній взаємодії. Клієнтом є веб-браузер чи інша прикладна програма, що виконується у програмно-апаратному середовищі користувача. Клієнт надсилає запити відповідно до введених користувачем даних і відображає відповідь, що приходить із сервера. Сервером є прикладне програмне забезпечення, що виконується у програмно-апаратному комплексі власника веб-додатку або у хмарному середовищі. Сервер оброблює запити від клієнта і надсилає відповіді. Під час обробки запиту сервер може звертатися до бази даних і/або файлової системи, як це показано на рисунку 1.1.

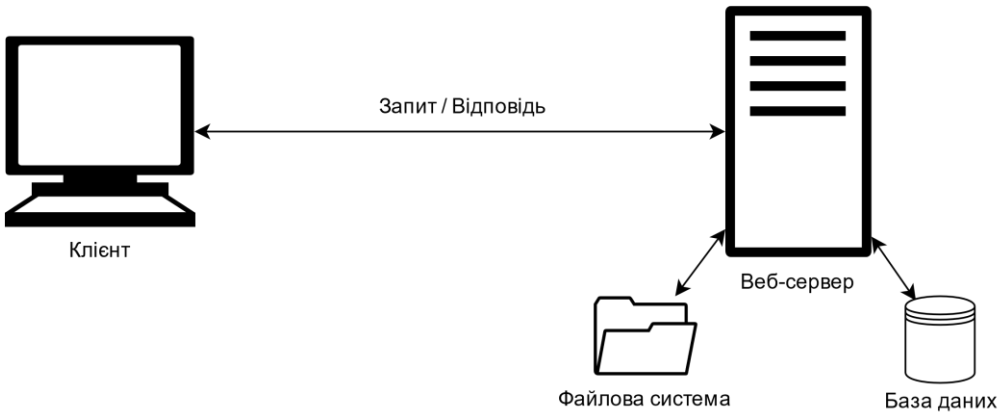


Рисунок 1.1 – Клієнт-серверна архітектура веб-додатків

Сучасні веб-додатки, як правило, представляють собою декілька програм, що обмінюються даними за допомогою мережевих протоколів [11].

Залежно від загального числа серверів та служб зберігання даних, можна виділити наступні моделі архітектури веб-додатків [12]:

- модель з одним веб-сервером та однією базою даних
- модель з кількома веб-серверами та однією базою даних
- модель з кількома веб-серверами та кількома базами даних
- мікросервісна модель

Один веб-сервер, одна база даних є найпростішою моделлю архітектури веб-додатків. Перевагою цієї моделі є простота впровадження та обслуговування веб-додатку. Недоліком є низька надійність. Користувач не зможе отримати доступ до необхідних йому ресурсів, якщо веб-сервер стане недоступним з тих чи інших причин.

Ключова особливість моделі із декількома веб-серверами і однією базою даних є використання кількох веб-серверів. Якщо один із веб-серверів виявиться недоступним, запит буде автоматично перенаправлено на інший веб-сервер. Перевагою підходу є підвищена надійність, порівняно із попередньою моделлю з одним веб-сервером. Вузьким місцем є база даних. Якщо сховище зберігання виявиться недоступним, то будь-яке функціонування веб-додатку буде неможливим, поки не відновиться зв’язок із базою даних.

Найбільш надійною є модель із кількома веб-серверами і кількома базами даних. Надійність підвищується за рахунок використання кількох баз даних. Існує два підходи до використання кількох баз даних: вони можуть дублювати одна одну і слугувати резервним сховищем на випадок втрати доступу до основного, або рівномірно розподіляти інформацію одна між одною.

Вищезгадані моделі відносяться до монолітної архітектури веб-додатків. Мікросервісну модель можна віднести до мікросервісної архітектури [12].

1.2.1 Монолітна архітектура

Особливістю монолітної архітектури є реалізація різних функціональних частин, таких як бізнес-логіка, система аутентифікації і авторизації, система

сповіщень тощо в одній програмі [13]. Типова схема монолітної архітектури зображена на рисунку 1.2.

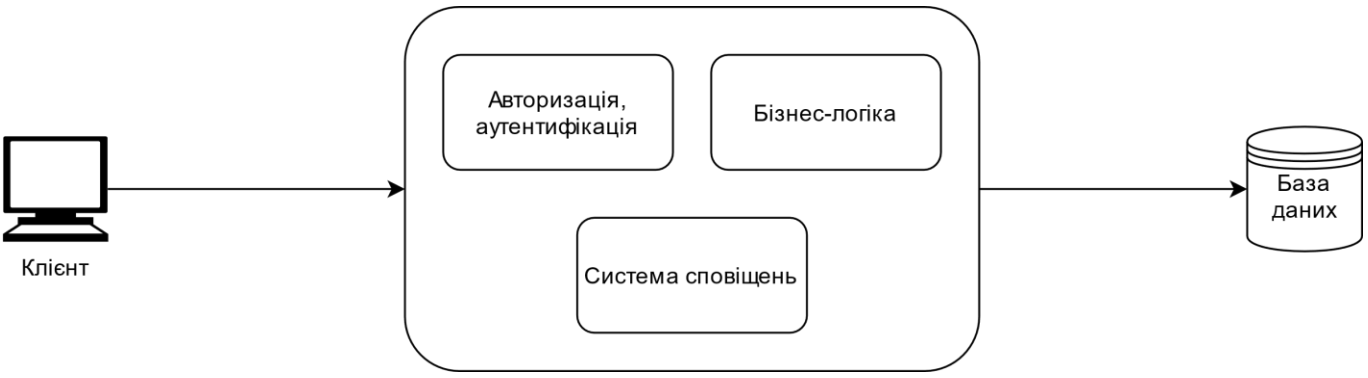


Рисунок 1.2 – Схема роботи веб-додатку з монолітною архітектурою. Стрілками показано запит від клієнта до веб-сервера та запит від веб-сервера до бази даних

Перевагами монолітної архітектури є [14]:

- простота створення і обслуговування веб-додатку
- простота тестування веб-додатку
- легке впровадження сторонніх компонентів, таких як фреймворків, шаблонів і скриптів у веб-додаток
- просте впровадження веб-додатку у використання

Недоліками монолітної архітектури є:

- велика кількість коду в одній програмі, що робить складнішим розуміння принципів роботи програми
- елементи веб-додатку сильно залежні одне від одного, що ускладнює перехід на нові технології чи нові версії раніше використовуваних технологій
- оновлення веб-додатку є складним процесом
- погана масштабованість веб-додатку

Отже, основними перевагами монолітної архітектури є простота її розробки, впровадження та обслуговування. Оперативне внесення змін, оновлення програми, її масштабованість є ключовими недоліками монолітної архітектури.

1.2.2 Мікросервісна архітектура

Мікросервісна архітектура – архітектура, при якій різні функціональні елементи програми розділені на відокремлені одне від одного, кожен з яких виконує свою частину логіки веб-додатку [12, 13]. Типова схема мікросервісної архітектури зображена на рисунку 1.3.

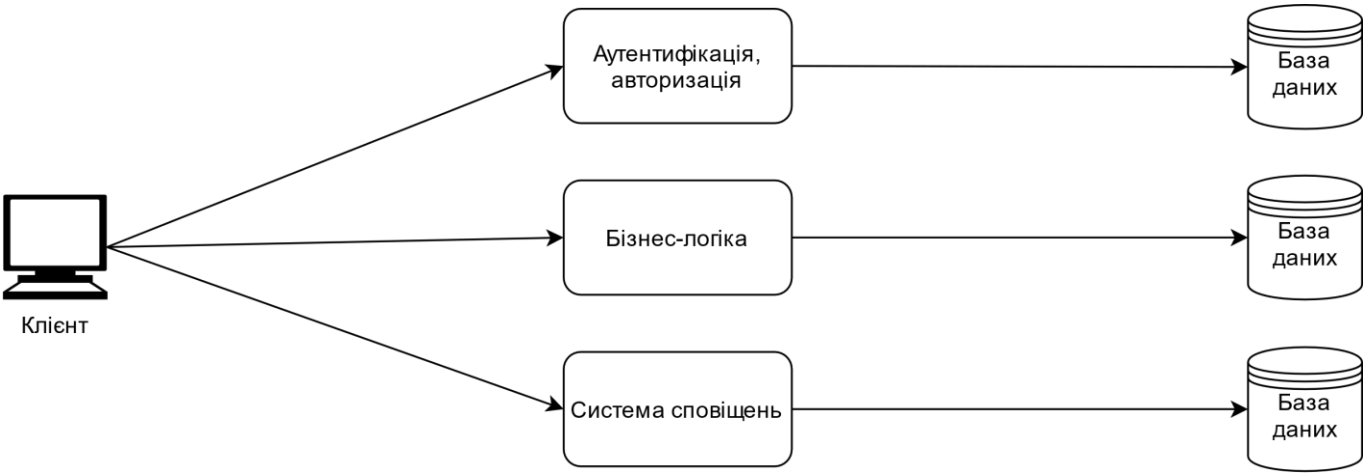


Рисунок 1.3 – Схема роботи веб-додатку з монолітною архітектурою. Стрілками показано запити від клієнта до сервісів та запити від сервісів до баз даних

Переваги мікросервісної архітектури:

- розподілення коду по різним сервісам, що робить простішим розуміння роботи програми
- кожен із сервісів є незалежним від інших, що дозволяє впроваджувати компоненти веб-додатку незалежно, а помилки в коді одного сервісу не впливають на роботу інших

- хороша масштабованість за рахунок окремого, незалежного обслуговування кожного із сервісів
- гнучкість у використанні технологій для роботи кожного із сервісів

Недоліки мікросервісної архітектури:

- кожен сервіс має бути спроектований, розроблений і впроваджений незалежно від інших, що створює додаткову складність
- мікросервісна архітектура є комбінацією різних служб і платформ, що створює додаткові вимоги до конфігурації і підтримки для кожного із сервісів і сховищ даних
- складність тестування архітектури із незалежними сервісами

Отже, мікросервісна архітектура немає недоліків поганої масштабованості чи гнучкість, однак створює додаткову складність при проектуванні, розробці і впровадженні веб-додатку.

1.3 Мережевий протокол прикладного рівня SOAP

SOAP (скорочено від Simple Object Access Protocol) – мережевий протокол прикладного рівня, призначений для формування та обробки запитів і відповідей веб-сервісів. SOAP базується на використанні мови XML для передачі повідомлень чи виклику віддалених процедур. Для передачі запитів і відповідей SOAP може використовувати широкий масив протоколів, таких як SMTP, HTTP, JMS тощо [15].

Для побудови SOAP-запиту необхідно мати опис SOAP-сервісу: ім’я методів, їх аргументи, формат відповіді. Для опису використовується Web Services Definition Language (WSDL), що базується на XML. Типовим прикладом оформлення WSDL-файлу є [16]:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
```

```
</message>

<portType name="glossaryTerms">
 <operation name="getTerm">
  <input message="getTermRequest"/>
  <output message="getTermResponse"/>
 </operation>
</portType>

<binding type="glossaryTerms" name="b1">
  <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
   <soap:operation soapAction="http://example.com/getTerm"/>
   <input><soap:body use="literal"/></input>
   <output><soap:body use="literal"/></output>
 </operation>
</binding>
```

Структура інформації, сформованої за вимогами протоколу SOAP, зображена на рисунку 1.4. Елементами даних у форматі XML за протоколом SOAP є [17, 18]:

- Envelope (конверт) – кореневий елемент XML-документу
- Header (заголовок) – необов'язковий елемент, що містить атрибути повідомлення, такі як інформація про безпеку або мережеву маршрутизацію
- Body (тіло) – містить основну інформацію
- Fault (помилка) – необов'язковий елемент, що містить інформацію про помилки, які виникли при обробці повідомлення

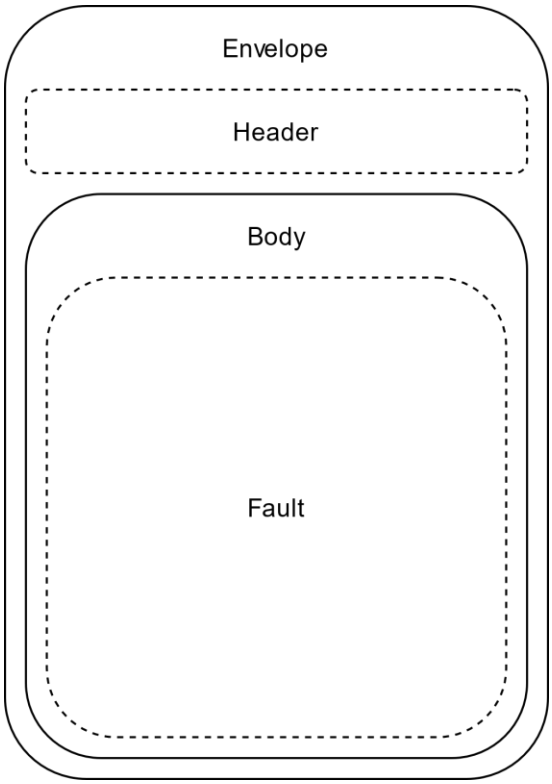


Рисунок 1.4 – Структура повідомлення у протоколі SOAP

Приклад запиту за допомогою SOAP має наступний вигляд [19]:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
```

```
</soap:Body>

</soap:Envelope>
```

У наведеному прикладі SOAP використовує протокол HTTP для передачі даних мережею [18]. Контент передається у вигляді тексту, сформованого відповідно до мови розмітки XML. Відповідь на запит може мати наступний вигляд [19]:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
 <m:GetStockPriceResponse>
  <m:Price>34.5</m:Price>
 </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

У загальному випадку схема роботи веб-служби за допомогою SOAP має наступний вигляд: клієнт, дотримуючись вимог, вказаних у WSDL-структурі, надсилає запит до сервісу. Сервіс формує і надсилає відповідь. Процес обміну даними зображено на рисунку 1.5.

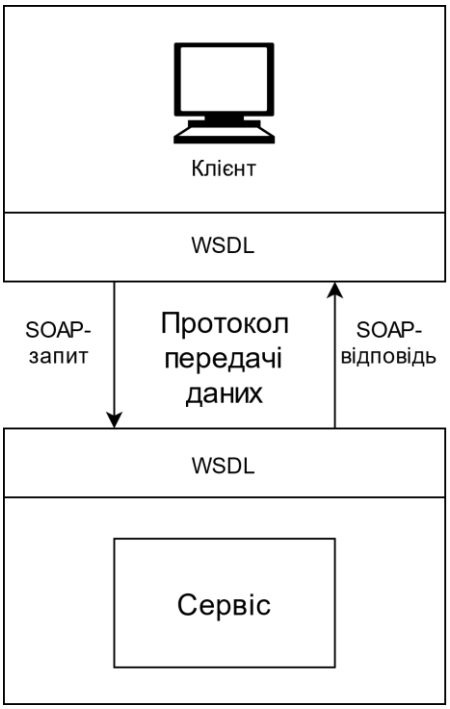


Рисунок 1.5 – Загальна схема роботи веб-сервісу, що використовує формат передачі даних SOAP

1.4 Архітектурний стиль побудови веб-додатків REST

REST (скорочено від Representational State Transfer) – архітектурний стиль побудови веб-додатків. Термін вперше упровадженим Роєм Філдінгом, одним із творців протоколу HTTP, у 2000 році. Згідно з його роботою [20], REST є архітектурним стилем, що ігнорує деталі реалізації компонентів та синтаксис протоколів і фокусується на ролі компонентів, обмеження їх взаємодії з іншими компонентами та інтерпретації важливої інформації.

Рой Філдинг визначає наступні обмеження архітектурного стилю REST:

- Клієнт-сервер. Розділення мережі на клієнтську і серверну частини покращує переносимість клієнтської частини і масштабованість серверної частини.
- Відсутній стан. Кожен запит містить усю необхідну інформацію для обробки сервером і не може використовувати сховище серверу для зберігання стану.

- Кеш. Інформація у відповіді помічена як кешована чи некешована. Якщо інформація кешована, клієнт може використати її для майбутніх запитів з аналогічною структурою.
- Єдиний інтерфейс. За допомогою єдиного інтерфейсу для усіх компонентів системи забезпечується їх незалежність одне від одного, що покращує гнучкість системи. Недоліком є втрата у швидкодії. Усі компоненти мають використовувати загальний стандарт замість специфічних оптимізованих для кожної платформи рішень.
- Розшарована система. Компоненти організовані у ієрархічну структуру. Кожен компоненти знає про взаємодію лише на своєму рівні ієрархії. Таким чином клієнт і сервер не володіють інформацією про проміжні рівні передачі інформації.
- Код за вимогою. REST дозволяє розширювати функціонал системи через обмін і виконання програмного коду між компонентами системи. Це зменшує кількість необхідного для імплементації компонентів функціоналу. Однак це висуває додаткові вимоги до технологій, які використовують клієнт і сервер, що зменшує гнучкість системи. Дотримання обмеження коду за вимогою є необов'язковим, на відміну від попередніх обмежень.

REST не накладає обмежень на формат, в якому передаються дані. Сервіси, побудовані за стилем REST (RESTful-сервіси) можуть використовувати не лише XML, а і куди більш компактні та зрозумілі JSON, HTML чи інші формати. RESTful-сервіси використовують HTTP-методи GET, POST, PUT, DELETE для передачі даних. Загальну схему роботи системи, спроектованої за стилем REST, зображено на рисунку 1.6. Клієнт надсилає дані у необхідному для сервісу форматі (у даному випадку JSON) через один із HTTP-методів. Сервіс формує відповідь, надсилає дані клієнту.

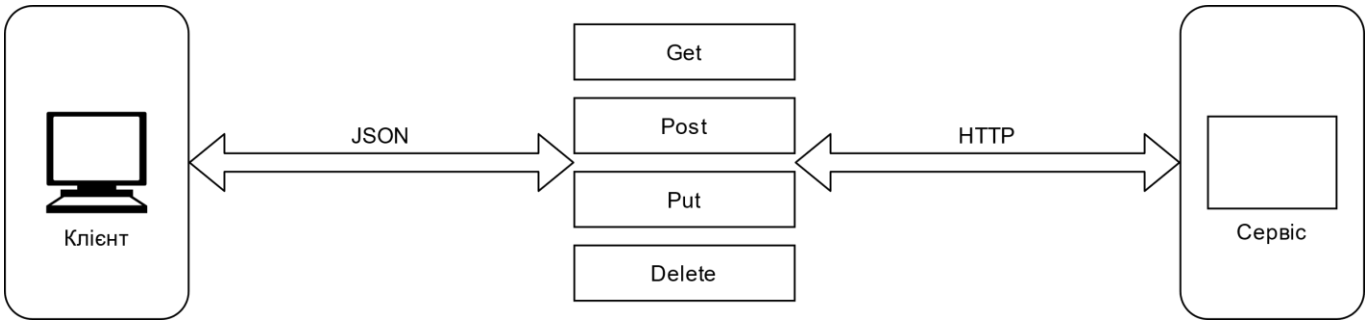


Рисунок 1.6 – Загальна схема роботи RESTful-сервісу

SOAP і REST є різними поняттями у галузі розробки веб-додатків. SOAP – протокол, що визначає можливий формат обміну даними у системі. REST – архітектурний стиль, що описує компоненти клієнт-серверної системи. Поняття SOAP і REST часто згадуються і порівнюються у середовищі веб-розробки. У таблиці 1.1 наведено порівняння SOAP і REST.

Таблиця 1.1

Порівняння протоколу SOAP і архітектурного стилю REST

SOAP	**REST**
Simple Object Access Protocol	Representational State Transfer
SOAP – протокол. Він вимагає WSDL-файлу, що містить необхідний опис веб-сервісу.	REST – архітектурний стиль, що містить наступні обмеження: • Клієнт-серверність • Відсутність збереження стану • Кеш • Розшарованість • Код за вимогою
SOAP не може використовувати REST, оскільки SOAP – протокол, а REST – архітектурний стиль.	REST може використовувати SOAP як протокол для передачі запитів і відповідей.
SOAP використовує сервісні інтерфейси для представлення своїх можливостей клієнтам.	REST використовує Uniform Resource Locator (URL) для доступу до компонентів системи. Наприклад, для доступу про інформацію щодо ціни товару IBM може бути використана наступна URL: http://www.example.org/stock/price/IBM

Продовження таблиці 1.1

Формат повідомлень SOAP громісткий, вимагає широкої пропускної здатності каналів зв'язку. <?xml version=»1.0»?> <soap:Envelope xmlns:soap=»http://www.w3.org/2003/05/soap-envelope/» soap:encodingStyle=»http://www.w3.org/2003/05/soap-encoding»> <soap:Body xmlns:m=»http://www.example.org/stock»> <m:GetStockPriceResponse> <m:Price>34.5</m:Price> </m:GetStockPriceResponse> </soap:Body> </soap:Envelope>	REST має менш строгі вимоги до пропускної здатності за рахунок можливості використовувати менш громіздкі формати, такі як JSON. {"price":"34.5"}
SOAP використовує винятково XML для передачі повідомлень.	REST може використовувати JSON, XML, HTML, простий текст та інші типи передачі інформації.

1.5 Постановка завдання

У першому розділі проаналізовано структуру веб-додатків, архітектурні моделі веб-додатків, сучасні підходи і архітектурні підходи до побудови веб-додатків. На основі проведеного аналізу щодо функціонування веб-додатків та згідно з визначеною нормативно-правовою базою необхідно:

- визначити найбільш поширені вразливості, властиві сучасним веб-додаткам
- проаналізувати можливі засоби та механізми захисту від загроз, що стосуються визначених вразливостей
- створити веб-застосунок і впровадити деякі із зазначених засобів та механізмів захисту
- надати відповідні рекомендації щодо упровадження засобів та механізмів захисту веб-додатків на основі створеного захищеного веб-застосунку

Висновки за розділом 1

У розділі 1 було проаналізовано структуру сучасних веб-додатків, проведено аналіз архітектур веб-додатків та найбільш поширених підходів до їх побудови.

Веб-додатки проектуються за допомогою однієї із чотирьох архітектурних моделей: один веб-сервер і одна база даних, декілька веб-серверів і одна база даних, декілька веб-серверів і декілька баз даних, мікросервіс. Перші три моделі можна класифікувати як монолітну архітектуру. Остання є мікросервісною архітектурою.

Монолітна архітектура – архітектура, де вся логіка сконцентрована у одній програмі. Перевагою мікросервісної архітектури є простота розробки і підтримки. Недоліком є висока залежність компонентів веб-додатку одне від одного, що ускладнює оновлення використовуваних технологій і зменшує надійність веб-додатку.

Мікросервісна архітектура – архітектура, де логіка рознесена по різним модулям (сервісам), кожен з яких є незалежним одне від одного. Перевагою є висока надійність даного підходу, відносна легкість оновлення окремих компонентів та технологій, що використовуються у системі. Недоліком є складність розробки.

Найбільш розповсюдженими технологіями для побудови сучасних веб-додатків є Simple Object Access Protocol та Representational State Transfer. SOAP – протокол для формування та обробки запитів і відповідей веб-сервісів. Для передачі інформації по каналам зв'язку використовується мова розмітки XML. Кожен XML-файл із повідомленням SOAP має містити Envelope (конверт), який є кореневим елементом XML-документу, та Body (тіло), що містить інформацію, яка передається. Необов'язковими елементами є Header (заголовок), що містить атрибути повідомлення, такі як інформація про безпеку або мережеву маршрутизацію, та Fault (помилка), що містить інформацію про помилки, які виникли при обробці повідомлення. У якості протоколу передачі даних веб-додатки, що працюють із SOAP, можуть використовувати HTTP, SMTP чи інші протоколи прикладного рівня. REST – архітектурний стиль, що має наступні обмеження: клієнт-серверність, відсутність стану, кешованість, єдиний інтерфейс. розшарованість система, код за

вимогою. REST може використовувати будь-який формат для передачі даних, не обмежуючись XML чи JSON. Додатки, розроблені за допомогою REST, працюють поверх HTTP. REST є архітектурним стилем, а SOAP – протоколом. SOAP може бути побудованим за принципами REST, хоч і не обмежується ними.

РОЗДІЛ 2
ОСНОВНІ ВРАЗЛИВОСТІ ВЕБ-ДОДАТКІВ

Документ OWASP Top Ten, створений Open Web Application Security Project, виділяє наступні 10 найбільш поширених вразливостей за 2017 рік [21]:

1. Ін’єкції
2. Порушена аутентифікація
3. Викриття чутливої інформації
4. Зовнішні сутності XML (XXE)
5. Порушений контроль доступу
6. Недоліки у конфігураціях безпеки
7. Міжсайтовий скриптинг (XSS)
8. Вразлива десеріалізація
9. Використання компонентів із відомими вразливостями
10. Неефективні моніторинг і журналювання

2020 CWE Top 25 Most Dangerous Software Weaknesses містить більш широкий перелік із 25 найбільш поширених вразливостей програм, актуальних для 2020 року [22]. Перелік включає у себе:

1. Міжсайтовий скриптинг (XSS)
2. Запис поза межами [буфера]
3. Неналежна перевірка вхідних даних
4. Читання поза межами [буфера]
5. Неналежне обмеження операцій у межах буфера пам’яті
6. Неналежна обробка спеціальних елементів при обробці SQL-команд (SQL ін’єкція)
7. Викриття чутливої інформації неавторизованим особам
8. Використання [пам’яті] після вивільнення
9. Міжсайтова підробка запиту (CSRF)
10. Неналежна обробка спеціальних елементів при обробці команд операційної системи (ін’єкція команд)

11. Переповнення цілого числа
12. Неналежне обмеження на шлях до директорії із обмеженим доступом
13. Вказівник на NULL
14. Неналежна аутентифікація
15. Відсутність обмеження на вивантаження файлів небезпечного типу
16. Невірне розподілення дозволів для критичних ресурсів
17. Неналежний контроль за генерацією коду (ін’єкція коду)
18. Недостатньо ефективно захищені облікові дані
19. Неналежне обмеження на посилання на зовнішні сутності XML
20. Використання жорстко закодованих облікових даних
21. Десеріалізація підозрілих даних
22. Неналежне керування привілеями
23. Неконтрольоване споживання ресурсів
24. Відсутня аутентифікація для критичних функцій
25. Відсутня авторизація

Список CWE є більш обширним, ніж список OWASP, оскільки включає вразливості, властиві не лише веб-додаткам. Списки мають декілька спільних елементів:

- Ін’єкції
- Порушена аутентифікація
- Викриття чутливої інформації
- XEE
- Порушення контролю доступу
- Неналежні конфігурації безпеки
- Проблеми десеріалізації

Джерело [23] містить статистику вразливостей веб-додатків за 2019 рік, складену на основі двох попередніх списків. Результати дослідження зображено на рисунку 2.1.

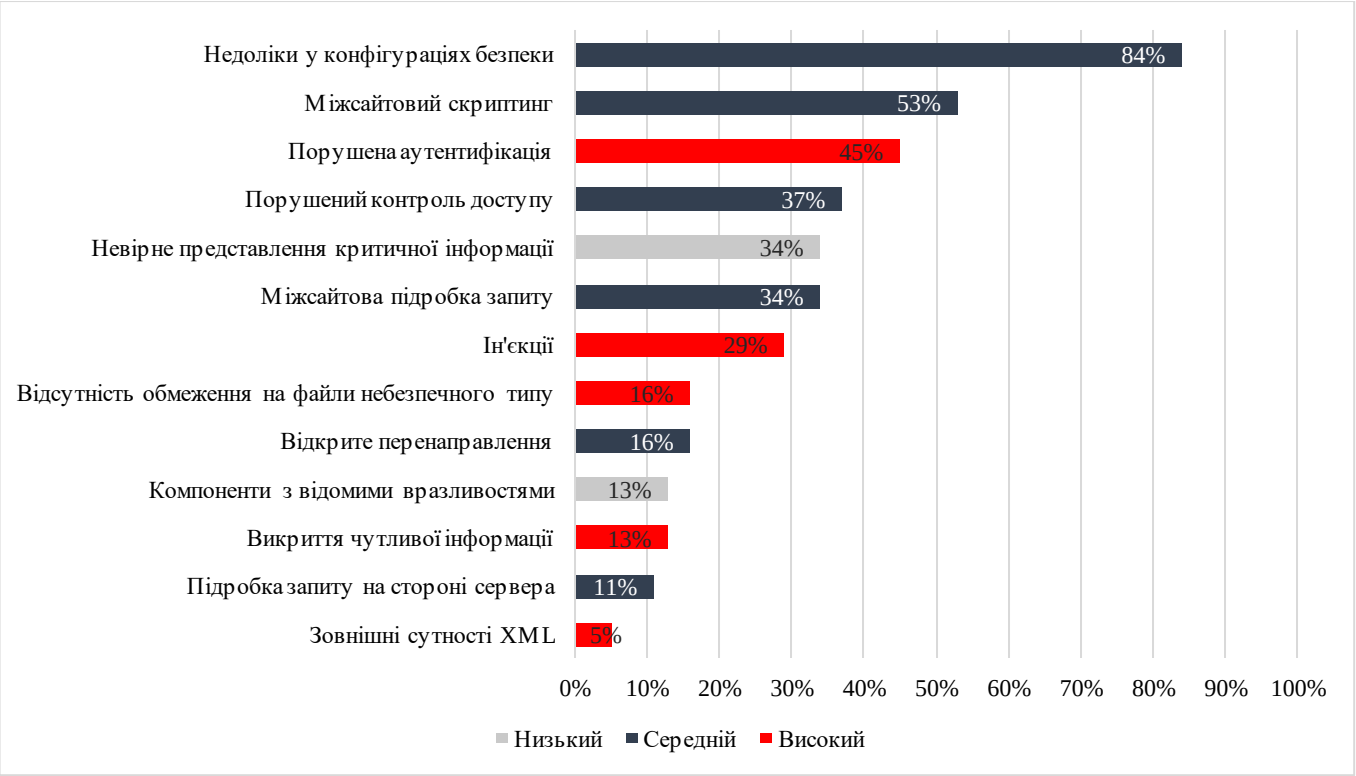


Рисунок 2.1 – Найбільш розповсюджені вразливості (у відсотках вразливих веб-додатків). Кольорами позначено різні рівні ризику

Як можна бачити із представленого на рисунку 2.1 графіку, трьома найбільш поширеними вразливостями із високим рівнем ризику є проблеми аутентифікації (45% додатків), ін’єкції (29% додатків), відсутність обмеження на вивантаження файлів небезпечного типу (16% додатків). Крім них, вразливостями із високим рівнем ризику є викриття чутливої інформації (13% додатків) і зовнішні сутності XML (5% додатків).

Найбільш розповсюдженою вразливістю як серед вразливостей із середнім рівнем ризику, так і серед вразливостей уцілому, є недоліки у конфігураціях безпеки (84% додатків). Міжсайтовий скриптинг властивий більш ніж половині веб-додатків (53% додатків). Слідом за ними за рівнем поширеності серед вразливостей із середнім ризиком ідуть проблеми контролю доступу (37% додатків), міжсайтова підробка запиту (34% додатків), відкрите перенаправлення (16% додатків) та підробка запиту на стороні сервера (11% додатків).

Серед усіх вразливостей є лише дві із низьким рівнем ризику: невірне представлення критичної інформації, що властиве для 34% веб-додатків, та використання компонентів із відомими вразливостями, властиве для 13% веб-додатків.

2.1 Порушена аутентифікація

Порушена аутентифікація є загальним терміном для вразливостей, експлуатація яких дозволяє зловмиснику видавати себе за легітимного користувача. У широкому розумінні термін пов’язаний із двома областями: керування сесіями та керування обліковими даними [24].

2.1.1 Керування сесіями

Для розуміння вразливості порушеної аутентифікації, пов’язаної із керуванням сесіями, варто розуміти алгоритм виконання процесу керування сесією. Перший етап процесу відбувається коли користувач вводить облікові дані у форму, що відображається у програмі-клієнті. Сервер, отримавши інформацію, надсилає запит до бази даних з метою знаходження і перевірки введених облікових даних. Якщо перевірка пройшла успішно, для користувача генерується унікальний ідентифікатор сесії, дійсний протягом обмеженого проміжку часу [25]. Схематично процес зображений на рисунку 2.2. Запит1 містить введені користувачем дані. Запит2 надсилається сервером для перевірки наявності введеної інформації у базі даних та її коректності. Відповідь2 містить знайдену у базі даних інформацію або пустий масив, якщо інформація не була знайдена. Відповідь1 містить згенерований ідентифікатор.

За типом перехоплення сесії виділяють активне перехоплення сесії, пасивне перехоплення сесії, змішане перехоплення сесії. У свою чергу змішане перехоплення сесії поділяється на сліпий та несліпий спуфінг [26, 27].

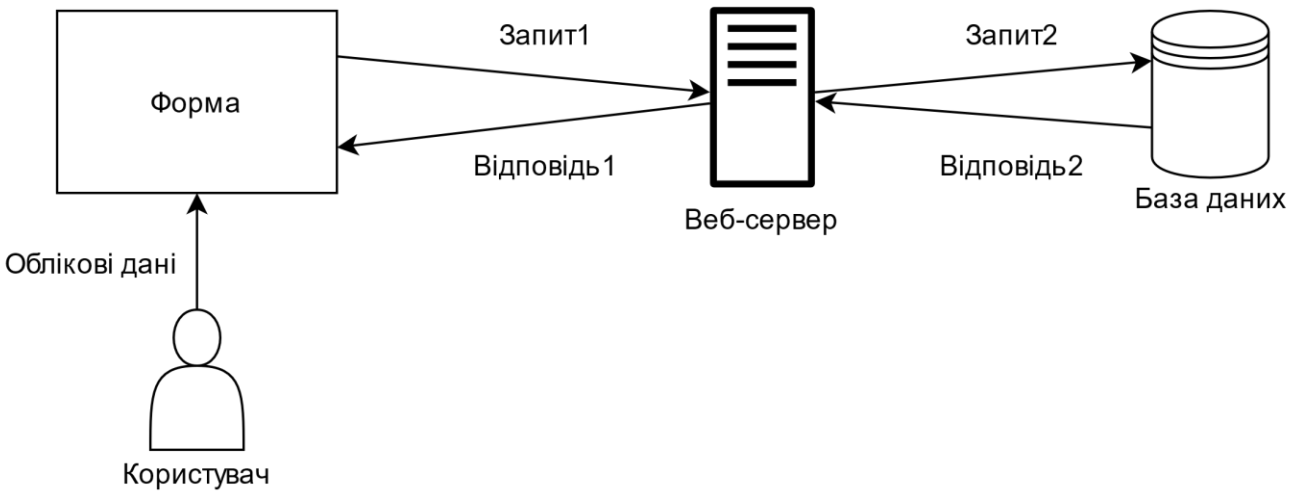


Рисунок 2.2 – Схема генерації ідентифікатора сесії.

Активне перехоплення сесії – техніка, за якої зловмисник проводить атаку на активну сесію між клієнтом і сервером. Результатом виконання атаки є перехоплення сесії та маскування під легітимного користувача. Одним із способів виконання атаки є переповнення мережі таким чином, щоб цільові машини не були у змозі обробити велику кількість трафіку і розірвали з'єднання (атака відмови у обслуговуванні, DoS). Коли сервер робить повторний запит на з'єднання, зловмисник видає себе за легітимного користувача, таким чином перехоплюючи сесію і отримуючи привілеї легітимного користувача [26, 27]. Схематично активне перехоплення сесії зображено на рисунку 2.3.

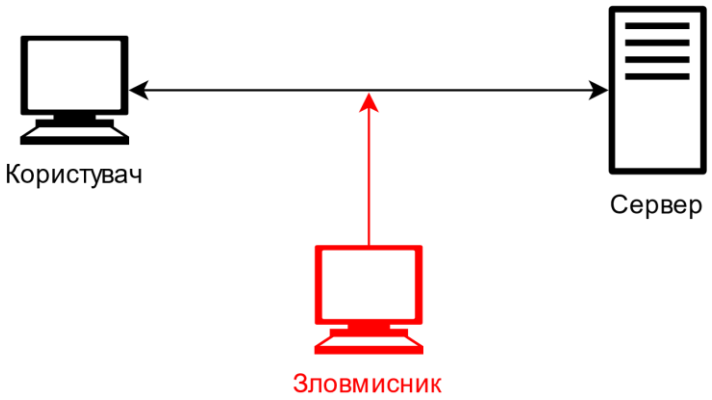


Рисунок 2.3 – Активне перехоплення сесії

Пасивне перехоплення сесії – метод, при якому зловмисник знаходиться між легітимним користувачем і сервером. Для легітимного користувача зловмисник прикидається сервером, а для сервера – легітимним користувачем. При цьому зловмисник може читати і/або змінювати вміст пакетів, що передаються між користувачем і сервером. Недоліком пасивного перехоплення сесії для зловмисника є те, що він може проводити атаку лише за умови активної сесії легітимного користувача. Якщо користувач завершує сесію або сервер з тих чи інших причин переустановлює сесійне з'єднання, зловмисник не зможе користуватися методом пасивного перехоплення [26, 27]. Схематично пасивне перехоплення сесії зображено на рисунку 2.4.

Рисунок 2.4 – Пасивне перехоплення сесії

Змішане перехоплення сесії – метод, що є комбінацією активного і пасивного перехоплення сесії. При реалізації даної атаки зловмисник аналізує трафік мережі і обирає найкращу для себе сесію для перехоплення. Як було згадано вище, змішане перехоплення поділяється на сліпий спуфінг та несліпий спуфінг.

Сліпий спуфінг є найскладнішою атакою з точки зору виконання. Сліпий спуфінг полягає у вгадуванні порядкових номерів TCP. Порядковий номер TCP – це ціле беззнакове число, що займає 32 біти і варіюється від 1 до 4 294 967 295. Певні програмні рішення можуть спростити зловмиснику вгадування порядкового номеру TCP. Згідно зі стандартом RFC 793, порядковий номер TCP має збільшуватися на одиницю кожні 4 мікросекунди, проходячи таким чином повний круг протягом чотирьох з половиною годин [28]. Більшість систем дотримуються власних правил, не закладених у стандарті. Так, BSD та Linux підвищують порядковий номер TCP на

128 000 кожну секунду, що дає повний цикл тривалістю у трохи більше 9 годин [29]. Ці фактори роблять порядковий номер TCP трохи більш передбачуваним. Тим не менш, дана атака вимагає вгадування числа серед великого масиву варіантів, що робить її малопрактичною.

Несліпий спуфінг – атака, при якій зловмисник додатково аналізує трафік, що проходить між клієнтом і сервером. Це спрощує завдання аналізу пакетів і видавання себе як легітимного користувача. За замовчуванням аналіз сторонніх пакетів неможливий у комутованій мережі, оскільки пакети направляються лише до тієї машини, якій вони призначені на отримання. За допомогою маніпуляцій із конфігурацією VLAN зловмисник здатен перехоплювати чужі пакети у комутованій мережі і на основі отриманих даних передбачати порядковий номер TCP [29].

Заходи безпеки проти перехоплення сесії ґрунтуються в основному на використанні захищених протоколів для передачі даних і програмних засобів захисту. До захищених протоколів відносяться:

- TLS (Transport Layer Security) – криптографічний протокол, що використовує асиметричні алгоритми шифрування для безпечної передачі даних у мережі із довжиною ключа 2048 чи 4096 біт.
- HTTPS (Hyper Text Transport Protocol Secure) – протокол HTTP, що використовує TLS для захисту інформації, що передається. Якщо зловмисник перехопить інформацію, що передається, він не зможе її використати у своїх цілях без розшифрування.

Крім захищених протоколів, можна упровадити додаткові міри захисту, такі як:

- Ідентифікатори сесії великої довжини. Чим більша довжина інформації, яку зловмисник має відгадати для успішного проведення атаки, тим довше процес угадування займе часу і тим менша ймовірність успіху перехоплення.
- Обмежений час тривалості сесії. Якщо користувач не здійснив вихід із системи при завершенні роботи, зловмисник має можливість перехопити ідентифікатор сесії та видати себе за легітимного користувача. Сесії з обмеженим часом дії не будуть поновлюватися, якщо користувач не працює у системі.

- Переаутентифікація. Суть методу полягає у тому, що термін дії ідентифікатора сесії обмежений, а користувач мусить повторно проходити процедуру аутентифікації по завершені терміну дії. Таким чином, якщо зловмисник перехопить попередньо згенерований ідентифікатор сесії, він не зможе отримати жодної практичної користі. Якщо зловмисник перехопить дійсний ідентифікатор, використати його він зможе протягом обмеженого проміжку часу. Недоліком даного підходу є незручності для користувача, викликані постійною необхідністю перезаходити у систему.
- CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) – система, що дозволяє відрізнити людину від написаної програми. CAPTCHA дозволяє створити єдину сесію для одного користувача і обмежує використання кількох сесій [30].

2.1.2 Керування обліковими даними

Атаки, що стосуються керування обліковими даними, пов’язані із слабкими паролями чи скомпрометованими обліковими даними. Згідно із джерелом [31], фішингові атаки та використання викрадених облікових записів є двома найбільш поширеними причинами витоку даних. Викрадені облікові дані дозволяють зловмиснику знаходитися у мережі, переглядати конфіденційну інформацію, змінювати чи видаляти необхідні дані тощо.

Набивання облікових даних (Credential Staffing) – атака, що базується на експлуатації недоліків проектування бази даних веб-додатку. Якщо зловмисник має доступ до бази даних, де паролі зберігаються у відкритому вигляді, він може використати облікові дані користувачів для входу у систему. Крім того, за допомогою перебору отриманих даних і підстановки їх у інші системи зловмисник може отримати доступ до сторонніх веб-додатків, оскільки логіни і/або паролі користувачів, зареєстрованих на різних платформах, може співпадати.

Розпилювання паролів (Password Spraying) – атака, схожа на набивання облікових даних. Відмінністю є те, що зловмисник не використовує інформацію із бази даних. Замість цього він опирається на набір відомих слабких паролів.

Дослідження 2019 року, проведене Національним центром безпеки Великої Британії, виявило, що найбільш використовуваним паролем є «123456» (23,2 млн), на другому місці – «123456789» (7,7 млн), а на третьому – «qwerty» (3,8 млн) [32]. Очевидно, що простота паролів у парі із їх популярністю створює великий ризик для користувачів і їхньої інформації.

Фішинг – атака, у ході якої зловмисник маскується під надійне джерело. Надсилаючи електронні листи, що містять спеціально сформовані посилання чи форми, зловмисник отримує від користувача дані облікового запису чи іншу корисну для себе інформацію. Фішингові атаки можуть бути націлені як на підприємство уцілому, так і на конкретного користувача. Дослідження 2020 року виявило, що 35% усіх витоків даних починалися із фішингу [33].

Захистом проти атак, пов’язаних із керуванням обліковими даними впровадження сильної політики безпеки на підприємстві. Політика безпеки має вимагати від користувачів системи створювати надійні паролі. Хороші рекомендації щодо створення надійних паролів можна знайти у документі [8]. Крім того, двофакторна аутентифікація може допомогти зменшити ризики.

Паролі у базі даних ні в якому випадку не мають зберігатися у відкритому вигляді. Замість цього можна використати хешування. Хеш дозволяє однозначно ідентифікувати правильність чи неправильність введеного пароля, оскільки один і той же рядок символів дасть один і той же хеш. Однак якщо різні користувачі мають однаковий пароль, його хешування дасть один і той же результат, що буде говорити зловмиснику про однаковість даних. Крім того, якщо пароль скомпрометований, його хеш міститься у відкритому доступі і зловмисник може легко відгадати вміст паролю. Для вирішення даної проблеми використовують так звану сіль – випадкові значення, що додаються до паролю кожного із користувачів. Ймовірність генерації однакових даних для двох різних користувачів із однаковим паролем практично рівна нулю, тому однаковість даних майже неможлива.

Для веб-додатків важливо мати засоби захисту від атак типу грубої сили. Трафік мережі може збільшитися до 180 разів під час атаки, що підбирає логін і пароль для входу у систему [34].

2.2 SQL-Ін’єкції

Вразливості ін’єкцій виникають, коли необроблена інформація надсилається на обробку як частина запиту. Дані, сформовані зловмисником, можуть змусити веб-додаток виконувати задані команди чи надати неавторизований доступ до інформації. Безпосередньо ін’єкція виникає, коли зловмисник відправляє спеціально сформовані дані, що при обробці змушують програму поводитися не так, як це було заплановано розробниками. Ін’єкції є одними із найбільш давніх і найбільш розповсюджених вразливостей [35].

Вразливості ін’єкцій можуть виникнути у тих місцях веб-додатку, де відбувається обробка користувацького вводу. SQL-ін’єкції є найбільш поширеним типом атаки згідно з джерелом [21].

Якщо веб-додаток має вразливість до SQL-ін’єкцій, то зловмисник може безперешкодно отримати доступ до інформації, що зберігається у базі даних. Крім очевидного порушення конфіденційності, цілісність і доступність також знаходяться під загрозою. Зловмисник, що експлуатує вразливість SQL-ін’єкцій, здатен сформувати запит на видалення інформації із бази даних, чим порушить доступність, або запит на зміну записів у базі даних, чим порушить цілісність.

Джерело [36] виділяє сім найбільш відомих типів SQL-ін’єкцій, що зображено у таблиці 2.1.

Таблиця 2.1

Найбільш відомі атаки типу SQL-ін’єкції

Тип	Опис
Тавтології	Ін’єкційні запити введені у одну чи більше логічну умову так, щоб завжди повертався результат true
Логічно невірні запити	Використання повідомлень про помилки для отримання корисної інформації про базу даних
Запити UNION	Ін’єкційний запит об’єднаний зі звичайним запитом за допомогою ключового слова UNION. Це дозволяє отримати інформацію, що стосується інших таблиць у базі даних

Продовження таблиці 2.1

Процедури, що зберігаються	Зловмисник може виконати вбудовану у базу даних функцію (процедуру) за допомогою ін'єкційного SQL-коду
Piggy-Backed запити	До початкового ін'єктованого запиту додається додатковий зловмисний код
Inference	Зловмисник робить необхідні йому висновки щодо бази даних на основі true/false відповідей Зловмисник робить необхідні йому висновки щодо бази даних на основі
Нестандартне кодування	Уникнення виявлення засобами безпеки за рахунок використання нестандартного для системи кодування. Використовується у парі з іншими техніками

За джерелами [37-40] можна знайти приклади SQL-ін'єкцій. Розглянемо загальний приклад ін'єкції, що можна знайти у джерелі [37]:

*String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "';* (2.1)

Вразливість ін'єкції виникає через пряме використання методу *request.getParameter("id")* і конкатенації отриманого результату із рядком запиту. Зловмисник може підставити будь-яку інформацію у параметр *"id"* і, за умови що введена інформація відповідає синтаксису мови обробки запитів, успішно провести зловмисні дії.

Допустимо, зловмисник проексплуатував вразливість SQL-ін'єкції і вираз *request.getParameter("id")* повернув значення *' or '1'='1*. Підставивши значення у вираз (2.1), отримаємо

*String query = "SELECT * FROM accounts WHERE custID='' or '1'='1';* (2.2)

Подібний вираз можна інтерпретувати як «Вибери все із *accounts*, що відповідає умові *custID={значення пустого рядка}* або *'1'='1'*». *custID*, судячи із

назви, є полем для збереження первинного ключа. Порівняння *custID=''* поверне true, якщо ключовим полем є пустий рядок, і false в усіх інших випадках. *'1'* завжди рівний самому собі, тому вираз *'1'='1'* повертатиме true. true або true, за законами булевої алгебри, поверне true; false або true поверне true. Оскільки умова *WHERE* запиту (2.2) буде виконуватися для кожного рядка таблиці, зловмиснику буде доступний весь вміст *accounts*.

Джерело [38] надає приклади атак, здійснених через веб-форму. Ресурс виділяє наступні типи ін'єкцій:

- Ін'єкції, що базуються на виразі 1=1 чи інших виразах, що завжди повертають true. Даний тип був розглянутий на прикладі виразу (2.1). Зловмисник може отримати доступ до даних у таблиці, що порушує властивість конфіденційності.
- Ін'єкції, що базуються на серії запитів. Серія запитів – це група із кількох запитів, що виконуються послідовно та розділені крапкою з комою. У випадку із виразом (2.1) зловмиснику необхідно буде ввести дані так, щоб *request.getParameter("id")* повернув результат, схожий на *' ; DROP TABLE accounts*. Тоді запит матиме вигляд *SELECT * FROM accounts WHERE custID='; DROP TABLE accounts*, що видалить таблицю акаунтів і порушить властивість доступності. За допомогою ключових слів *ALTER TABLE* зловмисник може змінити вміст записів таблиці, що порушить властивість цілісності.

Для захисту від ін'єкцій у базу даних потрібно уникати динамічних запитів, що утворюються конкатенацією SQL-команди і введеного користувачем тексту. Більшість мов програмування підтримують параметризовані запити. Програма самостійно підставляє користувацький ввід у запит, виконуючи необхідні міри проти ін'єкції. Повідомлення про помилки при обробці запитів мають бути оброблені на стороні сервера і не доступні клієнту. Якщо можливо, варто використовувати білий список із дозволених команд чи запитів. Білий список може містити як уже відомі команди, так і регулярні вирази для перевірки нових [40].

2.3 Недоліки у конфігураціях безпеки

Недоліки у конфігурації можуть виникнути на будь-якому рівні виконання програми, включаючи мережеві служби, платформу, веб-сервер, базу даних, фреймворки, код, віртуальні машини, контейнери чи файлове сховище [41]. Подібні вразливості можуть дати зловмиснику контроль над компонентами системи чи інформації, що зберігається в них. Додаток є вразливим, якщо

- невірні чи відсутні параметри безпеки на будь-якому із рівнів функціонування додатку
- непотрібні функції є активованими (зайві порти, сервіси, акаунти чи привілеї)
- акаунти та паролі за замовчуванням активовані і не змінені
- обробка помилок надає доступ користувачам до стеку виклику чи інших інформативних повідомленнь про помилки
- при оновленні системи не використовуються найновіші функції безпеки чи їх налаштування відоме звичайним користувачам
- параметри безпеки для серверу додатків (Spring, ASP.NET), бібліотеках, базах даних не налаштовані до значень, що дають найбільшу захищеність
- додаток використовує застаріле і/або вразливе програмне забезпечення

У наступних абзацах наведено приклади вразливих додатків.

Приклад 1. Сервер додатків комплектується із кількома прикладами розгорнутих програм. Дані приклади містять відомі вразливості, якими зловмисники користуються для проведення атак. Якщо один із цих додатків є консоллю адміністративного керування і якщо обліковий запис по замовчування не змінено, зловмисники отримують можливість атаки на систему.

Приклад 2. Перегляд списку каталогів не відключений на сервері. Якщо зловмисник виявить можливість перегляду списку каталогів, він зможе переглянути їх на предмет корисної для себе інформації і завантажити її. Наприклад, зловмисник може виявити шлях до скомпільованих класів Java (чи будь-якої іншої мови

програмування), завантажити їх і декомпілювати, порушивши цим властивість конфіденційності.

Приклад 3. Додаток надає детальні повідомлення про помилки звичайним користувачам, що потенційно може призвести до поширення інформації з обмеженим доступом для сторонніх осіб. Крім того, детальні помилки можуть вказати зловмиснику на вразливості, що містяться у програмі.

Для захисту від вразливості необхідно уважно налаштовувати кожен компонент системи і регулярно проводити аудит, що дозволить виявити вразливість на ранніх етапах.

2.4 Міжсайтовий скриптинг (XSS)

Міжсайтовий скриптинг (Cross-Site Scripting, XSS) – вразливість, пов’язана із роботою веб-сторінок. Для експлуатації вразливості зловмисник вводить зловмисний код у веб-сторінку. Коли користувач переходить за посиланням на сторінку, скрипт, написаний зловмисником, завантажується, виконується і передає необхідні зловмиснику дані.

Міжсайтовий скриптинг виконується на сторінках, що містять код JavaScript.. JavaScript може використовуватися зловмисником для [42]:

- маніпуляцій із користувацьким комп’ютером, таких як копіювання чи видалення файлів.
- моніторингу діяльності користувача, такої як введення даних через клавіатуру
- взаємодії з іншими сторінками, відкритими у веб-браузері користувача

Перша та друга проблеми можуть бути вирішені за допомогою виконання браузера у ізольованому середовищі. Вирішення третьої проблеми є дещо складнішим, оскільки перегляд сусідніх вкладок браузера також пов’язаний із апаратними вразливостями деяких процесорів архітектури ARM і x86: Spectre, що дозволяє зчитати дані із адресного простору жертви [43] і Meltdown, через яку процес може проігнорувати права доступу до сторінок пам’яті [44]. Дослідження [43] містить приклад експлойту, написаного на мові JavaScript. Для запобігання

третьої вразливості можливо використовувати спеціальні програмні чи архітектурні рішення [45].

Виділяють наступні типи кроссайтового скриптингу [42]:

- **Відображені XSS.** Додаток чи API оброблює користувацький ввід без валідації чи escape-символів як частину коду HTML. Найбільш поширеним методом виконання атаки є використання зловмисного посилання. Параметри посилання, що необхідні для відображення веб-сторінки, містять зловмисний код, який виконується у клієнтській програмі.
- **Збережені XSS.** Також відомі як ін'єкції HTML. Різниця між попереднім типом полягає у тому, що додаток чи API зберігає (не лише оброблює) користувацький ввід без валідації чи escape-символів. Класичним прикладом є форуми, на яких користувачі мають змогу напряму вставляти HTML-код при формуванні вводу.
- **XSS на основі DOM.** Особливий тип відображеного XSS, де помилки у надійному JavaScript-коді та необережне використання інформації з клієнтської сторони може результувати у XSS. Загроза є особливо небезпечною, бо інструменти із виявлення XSS не завжди можуть виявити дану вразливість.

Способи захисту від XSS описані у джерелах [46, 47, 48]. Джерело [46] визначає перелік правил, яких мають дотримуватися розробники веб-додатків, щоб захиститися від XSS уцілому:

0. Ніколи не вставляти невідому інформацію у програму, крім спеціально дозволених місць. Принцип «Забороняй усе, що не дозволено явно» є одним із ключових у безпеці. Якщо невідомі дані з тих чи інших причин мають бути вставлені у програму, програма має бути ретельно протестована і перевірена.

1. Використовувати кодування для HTML перед вставленням невідомих даних у елемент контенту HTML. Так, спеціальні символи мови HTML, такі як &, <, >, “, ‘ тощо мають бути закодовані у відповідно &, &lt, &gt, &quot, &#x27 тощо.

2. Використовувати кодування для атрибутів перед вставленням невідомих даних у атрибут HTML. Деякі атрибути можуть бути використані для

експлуатування вразливості навіть із використанням кодування, тому вони не мають використовуватися динамічно.

3. Використовувати кодування для JavaScript перед вставленням невідомих даних у параметри JavaScript. Деякі функції JavaScript не можуть бути використані безпечно навіть із кодуванням, тому їх використання має бути обмеженим.

4. Використовувати кодування CSS та строгу валідацію перед вставленням невідомих даних у значення HTML-стилю. Деякі значення стилів CSS не можуть безпечно використовуватися навіть із кодуванням, тому їх використання має бути обмеженим.

5. Використовувати кодування URL перед вставленням невідомих даних у параметри URL HTML-коду.

6. Дезінфікувати розмітку HTML спеціально створеними бібліотеками.

7. Уникати JavaScript URL'и.

Восьме правило звучить як «Запобігати XSS на основі DOM», проте не дає деталей чи пояснень. Натомість, восьме правило відсилає на джерело [47], що місить наступні правила:

1. Спочатку ескейпувати HTML, потім ескейпувати JavaScript перед вставленням невідомих даних у підконтекст HTML у межах контексту виконання.

2. Ескейпувати JavaScript перед вставленням невідомих даних у підконтекст атрибутів HTML у межах контексту виконання.

3. Бути обережним перед вставленням невідомих даних у обробник подій та підконтекст JavaScript-коду у межах контексту виконання.

4. Ескейпувати JavaScript перед вставленням невідомих даних у підконтекст атрибутів CSS у межах контексту виконання.

5. Спочатку ескейпувати URL, потім ескейпувати JavaScript перед вставленням невідомих даних у підконтекст атрибутів URL у межах контексту виконання.

6. Наповнювати DOM з використанням безпечних функцій чи властивостей JavaScript.

Політика безпеки контенту (Content Security Policy, CSP) – політика, що допомагає захисти веб-додаток від атак XSS, у тому числі XSS на основі DOM. CSP визначає наступні міри захисту проти XSS:

1. Обмеження вбудованих скриптів
2. Обмеження віддалених скриптів
3. Обмеження небезпечного JavaScript
4. Обмеження подання форм
5. Обмеження об’єктів

2.5 Порушений контроль доступу

Контроль доступу у веб-додатках – це механізм, за допомогою якого веб-додаток надає доступ до деяких функцій певним групам користувачів і забороняє іншим групам. Перевірки, необхідні для реалізації механізму, проводяться після аутентифікації користувача і визначають що авторизований користувач має право робити. Контроль доступу може бути розділений на наступні типи [49]:

- горизонтальний контроль доступу
- вертикальний контроль доступу
- контекстний контроль доступу

Механізм вертикального контролю доступу базується на обмеженні функцій веб-додатку для користувачів, що не мають права користуватися даними функціями. Наприклад, адміністратор має право модифікувати чи видаляти акаунти інших користувачів, у той час як звичайний користувач не має права взаємодіяти з іншими акаунтами подібним чином.

Механізм горизонтального контролю доступу базується на обмеженні взаємодії із ресурсами системи. Наприклад, користувач банківського додатку має право переглядати здійснені транзакції та робити нові. Користувачі інших акаунтів не мають права переглядати транзакції користувача і здійснювати нові від його імені. Для кожного існує свій набір інформації і функцій, що відповідає особистим транзакціям.

Механізм контекстного контролю доступу базується на обмеженні функцій і ресурсів веб-додатку залежно від стану об’єктів, з якими взаємодіє користувач. Користувач онлайн-магазину не може змінити вміст корзини, якщо товар оплачений.

2.5.1 Вертикальна ескалація привілеїв

Вертикальна ескалація привілеїв – ситуація, при якій користувач отримує доступ до заборонених для себе функцій. У базовому варіанті вертикальна ескалація повноважень виникає, якщо веб-додаток має доступні для усіх користувачів посилання, незалежно від того, повертають вони звичайну веб-сторінку чи сторінку із адміністративними функціями. Веб-додаток *www.example.com* є вразливим, якщо посилання *www.example.com/admin* є доступним для відвідування будь-якому користувачеві. Навіть якщо посилання не згадується у документації, що міститься у відкритому доступі, додаток все є вразливим. Зловмисник може скористатися атакою грубої сили, щоб виявити адміністративні посилання.

Деякі додатки визначають привілеї користувача під час входу у систему і зберігають цю інформацію в місці, що легко контролюється користувачем. Веб-додаток *www.example.com* є вразливим, якщо посилання *www.example.com/login/home* містить логічний параметр admin, що передається як частина URL. Таким чином, будь-який користувач може стати адміністратором, якщо перейде за посиланням *www.example.com/login/home?admin=true*.

Вертикальна ескалація привілеїв може виникнути у результаті невірних конфігурацій безпеки веб-додатку. Питання невірної конфігурації веб-додатків обговорювалося у підрозділі 2.4.

Захистом від вертикальної ескалації є обмеження використання посилань, що ведуть на сторінки із високопривілейованими функціями, для користувачів із низькими привілеями. Відомості про подібні посилання не мають публічно оголошуватися, а їхнє виконання має бути недоступним для низькопривілейованих користувачів.

2.5.2 Горизонтальна ескалація привілеїв

Горизонтальна ескалація привілеїв – ситуація, при якій користувач здатен отримати доступ до ресурсів, що належать іншому користувачу додатка. Доступ одного користувача банківської системи до записів про транзакції інших користувачів є прикладом горизонтальної ескалації повноважень.

Проведення горизонтальної ескалації повноважень є схожим до вертикальної. Допустимо, додаток *www.example.com* відображає ресурси користувача з ідентифікатором 24 за посиланням *www.example.com/account?id=24*. У такому випадку користувач може переглянути чужі ресурси, якщо змінить ідентифікатор у посиланні на сторінку.

У деяких випадках замість ідентифікатора використовується інший, значно менш передбачуваний тип даних, такий як глобальний унікальний ідентифікатор (Globally Unique Identifier, GUID). Хоч зловмисник не може передбачити значення ідентифікатора, даний метод не є надійним способом захисту. Глобальний ідентифікатор може бути відображений у інших місцях додатку, таких як особисті повідомлення чи коментарі.

Певні додатки здатні виявити спробу вертикальної ескалації повноважень і перенаправити на сторінку входу, якщо ідентифікатор активного користувача не відповідає вказаному. Це не є надійним способом захисту, бо під час перенаправлення може передатися частина чутливої інформації.

Захист від горизонтальної ескалації є схожим на захист від вертикальної ескалації. Перехід за посиланням, призначеним для інших користувачів, має бути забороненим. Замість перенаправлення на сторінку входу користувач має отримати статус-код 403 Forbidden, що говорить про недостатність прав для перегляду контенту, який розміщений за посиланням [9].

2.5.3 Ескалація привілеїв від горизонтальної до вертикальної

Іноді горизонтальна ескалація привілеїв може бути перетворена на вертикальну ескалацію привілеїв за рахунок компрометації більш привілейованого

користувача. Зловмисник може використати горизонтальну ескалацію для скидання чи перегляду паролів від адміністративних акаунтів, увійти в один із них і отримати доступ до адміністративних функцій, цим самим виконавши вертикальну ескалацію.

Для захисту від ескалації привілеїв від горизонтальної до вертикальної слід впровадити методи захисту від вертикальної і горизонтальної ескалації привілеїв.

2.6 Міжсайтова підробка запиту (CSRF)

Міжсайтова підробка запиту (Cross Site Request Forgery, CSRF) – вразливість, що дозволяє зловмиснику змусити жертву виконати ту чи іншу дію на веб-сайті. Для більшості веб-сайтів запити браузера містять облікові дані користувача, такі як cookies, IP-адресу, дані домену Windows тощо. Якщо користувач аутентифікований, веб-сайт не матиме жодної можливості відрізнити дії користувача від дій зловмисника, що експлуатує вразливість [50].

Атаки, що експлуатують вразливість до CSRF, орієнтуються на функції, що так чи інакше змінюють стан на сервері. Під зміною стану мається на увазі зміна логіна користувача, його паролю чи здійснення транзакцій від його імені. Отримання інформації про проведені дії не приносить користі зловмиснику, бо відповідь від сервера про проведені дії надходить на машину користувача, а не зловмисника.

Для CSRF потрібно виконання наступних ключових умов [51]:

- Відсутність непередбачуваних параметрів запиту. Дія, яку хоче виконати зловмисник від імені користувача, не містить непередбачуваних чи невідомих для зловмисника даних. Наприклад, функція зміни паролю є захищеною від CSRF, якщо для зміни необхідно знати попередній пароль, що не є доступним для зловмисника.
- Сесія, що базується на cookies. Виконання дії вимагає одного чи кількох запитів HTTP. Додаток, вразливість якого експлуатує зловмисник, орієнтується на session cookies щоб ідентифікувати користувача, що робить запити, та не містить іншого механізму перевірки користувацьких запитів.
- Релевантна дія. Додаток містить дію, яку вигідно виконати зловмиснику. Це може бути високопривілейована дія (зміна дозволів інших користувачів, зміна

конфігурації системи тощо) чи дія пов’язана із обліковим записом користувача (зміна пароля, здійснення транзакцій тощо).

Допустимо, користувач виконує наступний HTTP-запит:

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

email=wiener@normal-user.com
```

Це відповідає усім вищеназваним умовам для існування CSRF. Зловмисник може легко передбачити необхідні для виконання дій значення параметрів. Додаток використовує session cookies для ідентифікації користувача, що виконав запит. Запит призначений для зміни особистої інформації про користувача, що зберігається у середовищі веб-додатку. Зловмисник може сконструювати HTML-сторінку наступного вмісту:

```
<html>
  <body>
    <form action="https://vulnerable-website.com/email/change" method="POST">
      <input type="hidden" name="email" value="pwned@evil-user.net" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

Якщо користувач перейде на веб-сторінку зловмисника, що містить даний HTML-код, виконаються наступні дії:

1. Сторінка зловмисника направить запит до вразливого веб-сайту.
2. Якщо користувач авторизувався на платформі вразливого веб-сайту, браузер користувача автоматично включить до запиту session cookies.

3. Вразливий веб-сайт обробить запит, сприймаючи його як такий, що зроблений користувачем системи, а не зловмисником. Пошту буде змінено на pwned@evil-user.net, а не wiener@normal-user.com.

Механізм доставки CSRF є дещо схожим на механізм доставки відображених XSS. Зловмисник вставляє HTML-код у підконтрольний веб-сайт і примушує користувача перейти на підконтрольний веб-сайт. Це можна виконати за допомогою посилання у соціальних мережах, на інших веб-сайтах чи у електронній пошті. Деякі CSRF ґрунтуються на GET-запитах. Для них немає необхідності конструювати зловмисний HTML-код і вставляти його у веб-сторінку. Усі необхідні параметри передаються у URL. Якби у наведеному вище прикладі використовувався запит GET для зміни електронної пошти, зловмиснику необхідно було б сконструювати лише один рядок, схожий на наступний:

```
<img src="https://vulnerable-website.com/email/change?email=pwned@evil-user.net">
```

Для захисту від CSRF необхідно використовувати CSRF-токени. CSRF-токен – унікальне, непередбачуване значення, згенероване сервером і передане клієнту таким чином, що кожен наступний запит від клієнта буде містити CSRF-токен. CSRF-токен тримається у таємниці [52].

Використання CSRF-токену не звільняє розробників від додаткових заходів тестування і перевірки додатку на безпеку. Так, використання токену втрачає сенс, якщо валідація залежить від конкретного HTTP-методу. Перевірка токену має виконуватися як для методів POST, так для GET і інших типів HTTP-методів.

У деяких випадках додаток виконує перевірку токену лише у випадку його присутності у запиті. Якщо токену у запиті немає, перевірка упускається. Очевидно, що таке рішення не містить додаткової безпеки. Зловмисник може виключити токен з побудованого запиту і досягти необхідних для себе результатів. Якщо токену немає у запиті, валідація має коректно спрацьовувати і не пропускати такий запит.

У деяких випадках токен не прив’язаний до сесії окремого користувача. Натомість використовується глобальний пул токенів, кожен з яких є валідним для будь-якої користувацької сесії. У такому випадку зловмисник може просто авторизуватися на вразливому сайті і використовувати власний токен для

проведення атак на CSRF. Один токен має бути валідним строго для одного користувача.

Іноді може виникнути ситуація, коли токен не прив'язаний до session cookies. Наприклад, коли додаток використовує різні фреймворки для роботи з сесіями і захистом від CSRF. Якщо додаток містить функціонал, що дозволяє зловмиснику встановлювати значення cookies жертви, проблема вразливості не є вирішеною. Розробникам варто використовувати однакові інструменти для роботи або, якщо такої можливості немає, інтегрувати різні рішення так, щоб уникати пов'язаних із їх одночасним використанням вразливостей.

Висновки за розділом 2

У розділі 2 було розглянуто основні вразливості для веб-додатків. У першому підрозділі було проаналізовано топ-10 вразливостей для веб-додатків, сформованого джерелом [21]. Крім цього, було розглянуто список 25 вразливостей для програмного забезпечення уцілому, що також може бути використано для веб-додатків. На основі отриманих даних, а також джерела [23] було побудовано графік вразливостей, що включав у себе відсоток додатків, які мають ту чи іншу вразливість, а також рівень ризику. У наступних підрозділах був проведений аналіз вразливостей, чий ризик є вищим за низький і чий відсоток перевищує 20%.

У першому підрозділі було проаналізовано вразливість, пов'язаною із порушеною аутентифікацією. Порушену аутентифікацію можна розділити на ту, що виникає унаслідок невірного керування сесіями і на ту, що виникає унаслідок невірного керування обліковими даними. Для запобігання вразливості варто використовувати захищені протоколі, такі як TLS чи HTTPS, впроваджувати строгі політики безпеки. Паролі ні у якому випадку не мають зберігатися у відкритому вигляді у базі даних чи інших середовищах зберігання.

У другому підрозділі було розглянуто вразливості SQL-ін'єкції. SQL-ін'єкції – тип ін'єкцій, що базується на вставленні коду мови структурованих запитів (Structured Query Language). SQL-ін'єкції можуть призвести до порушення усіх трьох основних властивостей інформації: цілісність, конфіденційність, доступність.

Для захисту від ін’єкцій допоможуть використання параметризованих запитів, використання білого списку дозволених команд, уникнення потрапляння інформації про помилки у базі даних на сторону клієнта.

У третьому підрозділі було розглянуто вразливість недоліків у конфігураціях безпеки веб-додатку. Вразливість виникає у результаті однієї чи кількох із наступних умов: невірні чи відсутні параметри; непотрібні функції є активованими; акаунти та паролі за замовчуванням активовані і не змінені; обробка помилок повідомляє користувачам інформацію про стек виклику чи інші інформативні повідомлення про помилки; при оновленні системи не використовуються найновіші функції безпеки чи їх налаштування відоме звичайним користувачам; параметри безпеки для серверу додатків, бібліотек, баз даних не налаштовані до захищених значень. Для захисту від вразливості треба адекватно налаштовувати систему і регулярно проводити аудит.

У четвертому підрозділі було проаналізовано вразливість міжсайтового скриптингу (Cross-Site Scripting, XSS). Експлуатація вразливості дозволяє зловмиснику виконувати зловмисний код, написаний зазвичай на JavaScript у середовищі клієнта. XSS ділиться на наступні типи: відображені XSS, збережені XSS, XSS на основі DOM. Загроза XSS на основі DOM є особливо небезпечною, бо інструменти із виявлення XSS не завжди можуть виявити дану вразливість. Для захисту від XSS необхідно, у першу чергу, проводити ескейпування символів коду HTML і JavaScript при різних сценаріях обробки даних. Проти вразливостей, пов’язаних із апаратними платформами, допоможе вірно продумана та реалізована архітектура програми [45].

У п’ятому підрозділі було розглянуто вразливості, пов’язані із порушеним контролем доступу. Порушений контроль доступу призводить до ескалації привілеїв. Підрозділ містить аналіз вертикальної ескалації привілеїв, горизонтальної ескалації привілеїв та ескалації привілеїв від горизонтальної до вертикальної. Вертикальна ескалація привілеїв – ситуація, при якій користувач отримує доступ до функцій, до яких йому заборонений доступ. Захист від вертикальної ескалації полягає у тому, щоб обмежити використання посилань, що

ведуть на сторінки із високопривілейованими функціями, користувачам із недостатніми привілеями. Виникнення вразливості горизонтальної ескалації схоже із виникненням вразливості вертикальної ескалації і ґрунтується на невірно сконфігурованій системі посилань. Захист від вразливості горизонтальної ескалації є аналогічним до захисту від вразливості вертикальної ескалації. Якщо користувач із недостатніми привілеями намагається переглянути ресурси інших користувачів, йому має повернутися статус-код 403 Forbidden. Ескалація привілеїв від горизонтальної до вертикальної є комбінованим типом, що поєднує у собі горизонтальну і вертикальну ескалацію привілеїв. Захист від вразливості ґрунтується на захисті від вертикальної і горизонтальної ескалації привілеїв.

У шостому підрозділі було розглянуто вразливості міжсайтової підробки запиту. Міжсайтова підробка запиту виникає, коли легітимний користувач є аутентифікованим, а веб-сайт не має можливості відрізнити дії легітимного користувача від дій зловмисника. Для захисту від CSRF необхідно використовувати CSRF-токени. CSRF-токен – унікальне, непередбачуване значення, згенероване сервером та яке тримається у таємниці.

РОЗДІЛ 3
ПОБУДОВА ВЕБ-ДОДАТКУ З ВИКОРИСТАННЯМ ЗАСОБІВ ТА МЕХАНІЗМІВ ЗАХИСТУ

При виконанні практичної частини дипломної роботи був побудований веб-додаток із функціоналом інтернет-магазину, що реалізує певні механізми захисту від загроз. Мовою програмування для веб-додатку була обрана Java версії 16.

3.1 Огляд використовуваних технологій

Для побудови веб-додатку використовується програмний каркас. Spring автоматизує процес під'єднання та керування запитами до бази даних за допомогою Spring Data; спрощує розробку і підтримку веб-компонентів за допомогою Spring MVC; Spring Security дозволяє налаштовувати параметри безпеки компонентів веб-додатку. Для розгортання веб-сервера використовується Apache Tomcat, запуск якого відбувається автоматично за допомогою Spring. Вихідний код веб-додатку розташований за посиланням [53].

Для зберігання даних використовується реляційна база даних MySQL. Перевагою MySQL є безкоштовність, можливість установити на різні платформи, такі як Linux, Windows, macOS [54], і велика кількість навчального матеріалу у відкритому доступі. На рисунку 3.1 зображено структуру бази даних для розробленого веб-додатку. Ім'ям бази даних є shopdb2. Основними таблицями є products, clothes, categories, orders і users. Повний скрипт для генерації бази даних shopdb2 наведено у додатках.

Таблиця users пов'язана із таблицею orders зв'язком «Один до Багатьох»: один користувач може мати багато замовлень, одне замовлення може належати лише одному користувачу. Таблиця orders пов'язана із таблицею products зв'язком «Багато до Багатьох»: один продукт може міститися у багатьох замовленнях, одне замовлення може містити багато продуктів. Зв'язок «Багато до Багатьох» відбувається через допоміжну таблицю orders_products.

Таблиця products має зв’язок із двома іншими таблицями бази даних: categories і clothes. Із таблицею clothes таблиця products зв’язана як «Один до Одного». Продукт є абстрактною сутністю, яка розширюється іншими сутностями, такими як одяг (clothes), тому одному запису у таблиці clothes відповідає один запис у таблиці products. Із таблицею products таблиця categories пов’язана як «Один до Багатьох»: одна категорія може містити багато продуктів.

Таблиця categories є дещо особливою. Крім зв’язку із products, categories зв’язана із самою собою як «Один до багатьох»: одна категорія може мати багато підкатегорій. Зв’язок одного запису таблиці із іншим записом таблиці визначає категорію і її підкатегорії. Якщо зовнішній ключ категорії на підкатегорію вище має пусте значення (NULL), то категорія є корінною, що містить усі вказані підкатегорії.

Для розподілення доступу до бази даних shopdb2 створений користувач бази даних shopdb2@localhost з паролем «Super1_Secret2_Password3_db». Пароль містить літери малого регістру, великого регістру, цифри і спецсимвол.

У якості операційної системи для розгортання веб-додатку використовується дистрибутив Ubuntu 20.04.2.0 LTS, запущений у середовищі VMware Workstation.

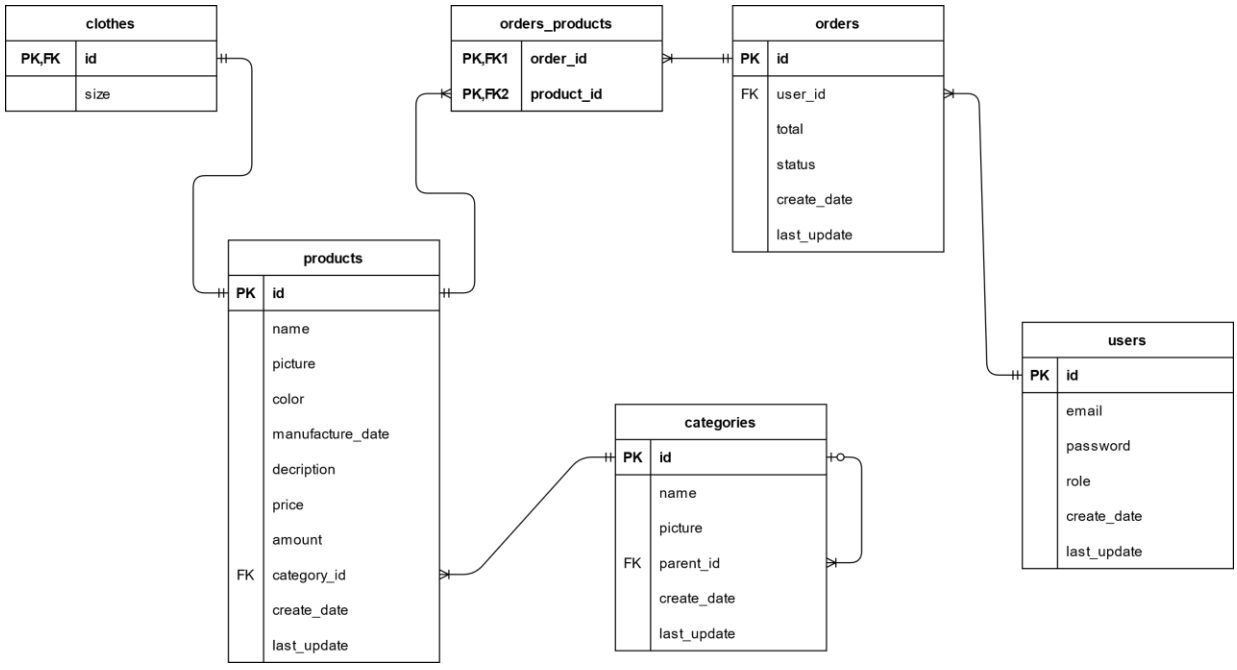


Рисунок 3.1 – Схема бази даних веб-додатку

3.2 Захист від загроз порушеної аутентифікації

3.2.1 Захист від перехоплення сесії

Допустимо, зловмисник виконує сканування трафіку з метою отримання необхідної інформації для перехоплення сесії. Для цього він використовує один із мережевих сніферів, таких як Wireshark, знаходячись у одній із клієнтом мережі. Для ефективного перехоплення мережевих пакетів зловмисник має перенаправляти мережевий трафік через свою машину або користуватися разом із клієнтом бездротовою передачею даних.

Щоб протидіяти атакам перехоплення сесії, як було визначено у розділі 2, необхідно використовувати протоколи, що містять криптографічний захист, такі як HTTPS. Для цього необхідно мати файл цифрового сертифікату.

Інструменти розробки Java (Java Development Kit, JDK) надають не лише компілятор і дебагер, а й утиліту для роботи із цифровими сертифікатами keytool. На рисунку 3.2 наведено приклад роботи утиліти. За допомогою параметрів можна вказати довжину ключа, алгоритм шифрування, термін дії, спосіб зберігання тощо. Згенерований файл слід помістити у папку ресурсів проекту (resources), а у файлі application.properties вказати розташування цифрового сертифікату, його тип, пароль та alias.

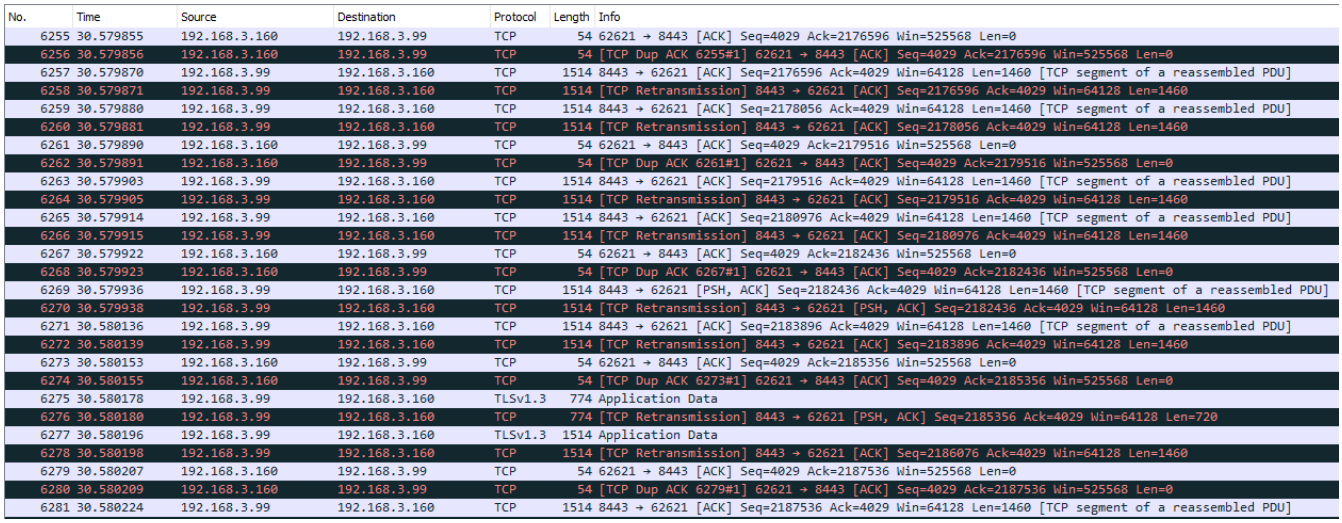
```
PS C:\Users\makar\Documents> keytool -genkey -alias diploma -keyalg RSA -keystore certificate.p12
 -storetype PKCS12 -storepass Super1_Secret2_Password3_p12 -keysize 4096 -validity 365
What is your first and last name?
  [Unknown]:  First Last
What is the name of your organizational unit?
  [Unknown]:  Cybersecurity
What is the name of your organization?
  [Unknown]:  KNU
What is the name of your City or Locality?
  [Unknown]:  Kyiv
What is the name of your State or Province?
  [Unknown]:  Kyiv
What is the two-letter country code for this unit?
  [Unknown]:  UA
Is CN=First Last, OU=Cybersecurity, O=KNU, L=Kyiv, ST=Kyiv, C=UA correct?
  [no]:  yes

Generating 4 096 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of
365 days
        for: CN=First Last, OU=Cybersecurity, O=KNU, L=Kyiv, ST=Kyiv, C=UA
PS C:\Users\makar\Documents>
```

Рисунок 3.2 – Генерація цифрового сертифікату за допомогою утиліти keytool

Розташувавши файл і вказавши необхідні параметри, Spring автоматично налаштує захищений протокол HTTPS. Як можна бачити із рисунку 3.3, на якому зображений аналіз трафіку між клієнтом і сервером, дані зашифровані за допомогою протоколу TLS версії 1.3.

TLS не гарантує 100% захищеності. Маючи необхідну інформацію пор секрети, зловмисник може розшифрувати перехоплений трафік [55]. В даному випадку захищеність мережі від прослуховування і перехоплення користувацьких даних залежить від наявності захищеності ключів та інших секретів.



No.	Time	Source	Destination	Protocol	Length	Info
6255	30.579855	192.168.3.160	192.168.3.99	TCP	54	62621 → 8443 [ACK] Seq=4029 Ack=2176596 Win=525568 Len=0
6256	30.579856	192.168.3.160	192.168.3.99	TCP	54	[TCP Dup ACK 6255#1] 62621 → 8443 [ACK] Seq=4029 Ack=2176596 Win=525568 Len=0
6257	30.579870	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [ACK] Seq=2176596 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
6258	30.579871	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [ACK] Seq=2176596 Ack=4029 Win=64128 Len=1460
6259	30.579880	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [ACK] Seq=2178056 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
6260	30.579881	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [ACK] Seq=2178056 Ack=4029 Win=64128 Len=1460
6261	30.579890	192.168.3.160	192.168.3.99	TCP	54	62621 → 8443 [ACK] Seq=4029 Ack=2179516 Win=525568 Len=0
6262	30.579891	192.168.3.160	192.168.3.99	TCP	54	[TCP Dup ACK 6261#1] 62621 → 8443 [ACK] Seq=4029 Ack=2179516 Win=525568 Len=0
6263	30.579903	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [ACK] Seq=2179516 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
6264	30.579905	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [ACK] Seq=2179516 Ack=4029 Win=64128 Len=1460
6265	30.579914	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [ACK] Seq=2180976 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
6266	30.579915	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [ACK] Seq=2180976 Ack=4029 Win=64128 Len=1460
6267	30.579922	192.168.3.160	192.168.3.99	TCP	54	62621 → 8443 [ACK] Seq=4029 Ack=2182436 Win=525568 Len=0
6268	30.579923	192.168.3.160	192.168.3.99	TCP	54	[TCP Dup ACK 6267#1] 62621 → 8443 [ACK] Seq=4029 Ack=2182436 Win=525568 Len=0
6269	30.579936	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [PSH, ACK] Seq=2182436 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
6270	30.579938	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [PSH, ACK] Seq=2182436 Ack=4029 Win=64128 Len=1460
6271	30.580136	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [ACK] Seq=2183896 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
6272	30.580139	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [ACK] Seq=2183896 Ack=4029 Win=64128 Len=1460
6273	30.580153	192.168.3.160	192.168.3.99	TCP	54	62621 → 8443 [ACK] Seq=4029 Ack=2185356 Win=525568 Len=0
6274	30.580155	192.168.3.160	192.168.3.99	TCP	54	[TCP Dup ACK 6273#1] 62621 → 8443 [ACK] Seq=4029 Ack=2185356 Win=525568 Len=0
6275	30.580178	192.168.3.99	192.168.3.160	TLSv1.3	774	Application Data
6276	30.580180	192.168.3.99	192.168.3.160	TCP	774	[TCP Retransmission] 8443 → 62621 [PSH, ACK] Seq=2185356 Ack=4029 Win=64128 Len=720
6277	30.580196	192.168.3.99	192.168.3.160	TLSv1.3	1514	Application Data
6278	30.580198	192.168.3.99	192.168.3.160	TCP	1514	[TCP Retransmission] 8443 → 62621 [ACK] Seq=2186076 Ack=4029 Win=64128 Len=1460
6279	30.580207	192.168.3.160	192.168.3.99	TCP	54	62621 → 8443 [ACK] Seq=4029 Ack=2187536 Win=525568 Len=0
6280	30.580209	192.168.3.160	192.168.3.99	TCP	54	[TCP Dup ACK 6279#1] 62621 → 8443 [ACK] Seq=4029 Ack=2187536 Win=525568 Len=0
6281	30.580224	192.168.3.99	192.168.3.160	TCP	1514	8443 → 62621 [ACK] Seq=2187536 Ack=4029 Win=64128 Len=1460 [TCP segment of a reassembled PDU]

Рисунок 3.3 – Аналіз трафіку між 192.168.3.160 (клієнт) і 192.168.3.99 (сервер)

Крім захисту від перехоплення сесії, використання захищеного HTTPS надає додатковий захист від викриття чутливої інформації. Логіни, паролі користувачів, їхні особисті дані, переписки тощо шифруються засобами криптографічного захисту із використанням надійних алгоритмів шифрування з достатньо довгими і сильними ключами.

3.2.2 Захист від атак, що стосуються керування обліковими даними

Щоб захисти облікові дані користувачів від зловмисників, у веб-додаток упроваджена політика створення сильних паролів. У полі введення паролю при

реєстрації використовується регулярний вираз (?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}. Будь-який введений текст, що менший за 8 символів у довжину, або не містить цифр, або містить літери лише одного регістру, не відповідатиме регулярному виразу і введена у форму інформація не буде надіслана на обробку сервером.

Навіть використовуючи політику сильних паролів, ризик викрадення облікових записів є достатньо високим, якщо вони зберігаються у базі даних у відкритому вигляді. Захищеність системи має бути комплексною і не покладатися лише на обмеження доступу до ресурсів. Паролі користувачів мають бути хешовані з використанням солі і надійних алгоритмів шифрування.

Для хешування паролів використовується алгоритм bcrypt, створений Нільсом Провосом (Niels Provos) і Девідом Маз'єресом (David Mazières) у 1999 році на базі алгоритму шифрування Blowfish. Типовий приклад даних, хешованих bcrypt має наступний вигляд:

$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy,

де *2a* – ідентифікатор алгоритму (bcrypt);

10 – вартість (2^{10} = 1024 раундів розширення ключа);

N9qo8uLOickgx2ZMRZoMye – закодована до 22 символів 128-бітна сіль;

IjZAgcfl7p92ldGxad68LJZdL17lhWy – закодований до 31 символу 192-бітний хеш.

Допустимо, зловмисник отримав доступ до машини, де розгорнута база даних. У такому випадку йому необхідно отримати пароль від акаунту адміністратора бази даних або від акаунту shopdb2. Якщо зловмисник знає пароль, він зможе виконати команду *SELECT * FROM users*. Як можна бачити із рисунку 3.4, зловмисник може отримати інформацію про поштові скриньки і ролі користувачів веб-додатку, але не їх паролі, що хешовані за допомогою алгоритму bcrypt.

```
diploma@ubuntu:~$ mysql -u shopdb2 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.25-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use shopdb2
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from users;
+----+---------------+--------------------------------------------------------------+-------+---------------------+---------------------+
| id | email         | password                                                     | role  | create_date         | last_update         |
+----+---------------+--------------------------------------------------------------+-------+---------------------+---------------------+
|  1 | a@shop.com    | $2a$10$HafCXdlh30LkgUNU3j53quDDXSNLA96Pl71avDmtYxQ9Tacg3A4oO | USER  | 2021-05-26 09:17:21 | 2021-05-26 09:17:21 |
|  2 | admin@shop.com | $2a$10$MwWc3T/NE5yBETEymHOAdeb5qxNApZE2z42aswu0DKo982O.ewyrm | ADMIN | 2021-05-27 03:58:27 | 2021-05-27 03:59:35 |
|  3 | a2@shop.com   | $2a$10$PLY3rNHfMbnS0jQd4UBmme8hU/OSx9w6fUmxNi9zbeKPYYBI7pmDa | USER  | 2021-05-27 04:00:53 | 2021-05-27 04:00:53 |
+----+---------------+--------------------------------------------------------------+-------+---------------------+---------------------+
3 rows in set (0.00 sec)

mysql>
```

Рисунок 3.4 – Вміст таблиці *users*, отриманий за допомогою команди *SELECT * FROM users*

3.3 Захист від SQL-ін’єкцій

Потенційно вразливими до SQL-ін’єкцій місцями побудованого веб-додатку є форми реєстрації та входу й перехід на сторінку категорії. В усіх згаданих місцях відбувається взаємодія із базою даних з використанням користувацького вводу. Допустимо, що зловмисник проводить ін’єкцію на сторінці входу, вставляючи зловмисний код у поле пошти. Як можна бачити із рисунку 3.5, форма вимагає введення рядку, що відповідає пошті, тому просто так вставити видалити таблицю користувачів за допомогою ін’єкції не вийде.

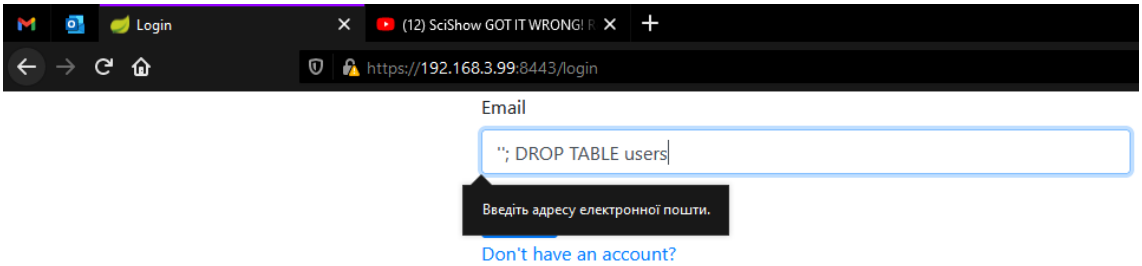


Рисунок 3.5 – Ін’єкція SQL-коду у форму входу

Зловмисник може спробувати провести ін’єкцію у рядок вводу URL при перегляді категорій товару, розміщених у веб-додатку. На рисунку 3.6 зображений результат спроби ін’єкції у рядок вводу браузера.

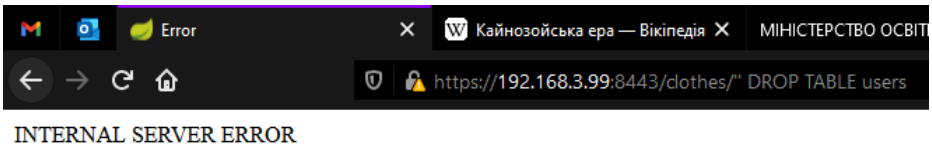


Рисунок 3.6 – Ін’єкція SQL-коду у рядок вводу браузера

Як можна бачити, обробка запиту видала помилку 500, що означає внутрішню помилку сервера. Додатково можна відмітити те, що на сторінці не відображаються деталі щодо помилки, що є ще одною мірою захисту. Зловмисник не зможе точно дізнатися які саме дії призвели до помилки і деталі самої помилки.

3.4 Захист від ескалації привілеїв

Веб-додаток містить сторінку адміністративного керування за посиланням *https://{address}/admin*. У кожного із користувачів може бути одна із наступних ролей: USER, ADMIN, BLOCKED. BLOCKED може переглядати товари. USER може переглядати товари і здійснювати покупки. ADMIN може переглядати товари, здійснювати покупки, змінювати статус покупок користувачів, переглядати список користувачів, блокувати/розблоковувати користувачів. Очевидно, що роль ADMIN має найбільші привілеї, тому отримання повноважень адміністратора є одною із можливих цілей зловмисника.

Розподіл доступу до сторінок та ресурсів залежно від акаунтів і їх привілеїв відбувається за допомогою програмного каркасу Spring Security. Налаштування програмних параметрів безпеки веб-додатку наведено у наступному фрагменті коду (див. додаток):

```
protected void configure(HttpSecurity http) throws Exception {
    http
```

```
            .csrf().ignoringAntMatchers("/login",  "/logout")
            .and()
            .authorizeRequests()
            .antMatchers("/categories/**").permitAll()
            .antMatchers("/admin/**").hasAuthority("ADMIN")
            .antMatchers("/cart/**").hasAnyAuthority("USER",  "ADMIN")
            .antMatchers("login").anonymous()
            .antMatchers("logout").authenticated()
            .and()
            .formLogin().loginPage("/login").defaultSuccessUrl("/").usernameParameter("email")
            .and()
            .logout().logoutUrl("/logout").logoutSuccessUrl("/")
            .logoutSuccessHandler(userService).invalidateHttpSession(true)
            .clearAuthentication(true);
    }
```

За допомогою об’єкту класу HttpSecurity виконується розподіл доступу до різних сторінок веб-додатку. Сторінки з адміністративним функціоналом доступні лише користувачам, що мають роль ADMIN. Сторінки категорій доступні усім, у тому числі неавторизованим користувачам. Сторінки, пов’язані із функціоналом корзини, доступні користувачам із ролями USER чи ADMIN. Сторінка входу доступна лише анонімним (неавторизованим) користувачам. Сторінка виходу доступна лише аутентифікованим користувачам.

HttpSecurity визначає URL для сторінок входу/виходу та подальші дії у випадку успішного входу/виходу. Так, при успішному вході у систему чи виході із системи відбувається перехід за посиланням *https://{address}/*. При виході також відбувається очищення даних аутентифікації та інвалідація сесії HTTP.

При переході за посиланням *https://{address}/cart* відбувається перенаправлення на сторінку входу, якщо вхід не був здійснений раніше. Анонімний користувач не може переглядати корзину, бо кожна покупка прив’язана до зареєстрованого користувача.

Логіка отримання і зміни інформації про покупки окремого користувача (ресурси користувача) прив’язана до класу SecurityContextHolder.

SecurityContextHolder надає механізм взаємодії із контекстом безпеки веб-додатку, де зберігаються дані про аутентифікованого користувача. Приклад використання SecurityContextHolder наведено у наступному фрагменті коду (див. додаток):

```
public Page<Product> getClothesInCart(int page, String sortField, boolean descending) {
    User currentUser = (User)
    SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    List<Order> orders = orderRepository.findAllByUserIdAndStatus(currentUser.getId(),
    Status.CREATED);
     return descending
          ? productRepository.findAllByOrders(orders, PageRequest.of(page,
    Constants.PRODUCTS_PER_PAGE, Sort.by(sortField).descending()))
          : productRepository.findAllByOrders(orders, PageRequest.of(page,
    Constants.PRODUCTS_PER_PAGE, Sort.by(sortField).ascending()));
}
```

Рядок User currentUser = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal(); повертає екземпляр класу User для користувача, що є аутентифікованим у браузері на момент виконання програми. Посилання для отримання ресурсів для окремого користувача не використовується, тому горизонтальна ескалація повноважень за допомогою URL є неможливою.

Розглянемо спробу отримати доступ до адміністративного функціоналу через вертикальну ескалацію привілеїв. Для цього, будучи аутентифікованим як користувач з роллю USER, спробуємо зайти на *https://{address}/admin*. Результат операції зображено на рисунку 3.7.

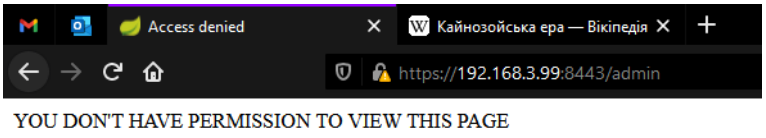


Рисунок 3.7 – Спроба увійти на сторінку адміністрування без необхідних привілеїв

Зловмисник не може отримати доступ до сторінки адміністрування. Якщо зловмисник спробує зайти на адміністративну сторінку, будучи неавторизованим у системі, веб-додаток перенаправить його на сторінку входу. Це не є недоліком. Якщо користувач є гостем, тобто неавторизованим у системі, веб-додаток не володіє жодною особистою інформацією про нього і не може її перенаправити.

3.5 Надання практичних рекомендацій

Згідно із розглянутими вразливостями й загрозами і побудованого веб-додатку із упровадженими засобами та механізмами захисту від загроз порушеної аутентифікації, можна надати наступні практичні рекомендації щодо підвищення безпеки веб-додатків:

- Для захищення мережі від перехоплення чутливої інформації та токенів безпеки варто використовувати захищені протоколи, такі як HTTPS. Для цього необхідно мати цифровий сертифікат, що буде використаний для налаштування криптографічно захищеного обміну даними.
- Для захисту від атак, що стосуються керування обліковими даними, необхідно впроваджувати політику створення сильних паролів. Зберігати паролі у базі даних необхідно не у відкритому вигляді, а з використанням надійних алгоритмів хешування і солі. Хешування гарантує унікальність згенерованих даних для одного користувача. Сіль гарантує те, що для різних користувачів згенеровані дані будуть різними.
- Для захисту від SQL-ін’єкцій необхідно строго обмежувати типи даних, що можуть бути введені користувачем, використовувати ескейпування символів, програмні каркаси. Використання перевірених проектів із вбудованими механізмами взаємодії із базою даних може спростити завдання захисту від SQL-ін’єкцій.
- Для захисту від ескалації привілеїв варто уважно виконувати налаштування програми, використовувати вбудовані у мову програмування або програмний каркас інструменти для обмеження доступу до ресурсів для

користувачів веб-додатку. Якщо це можливо, розподіл доступу не має базуватися на посиланнях.

Висновки за розділом 3

У розділі 3 було побудовано веб-додаток із захистом від загроз порушеної аутентифікації, SQL-ін'єкцій і ескалації привілеїв. Технологічний стек веб-додатку має наступний вигляд: Apache Tomcat у якості веб-серверу, Java у якості мови програмування веб-додатку, MySQL у якості бази даних, Linux Ubuntu 20.04.2.0 LTS у якості операційної системи сервера.

Захист від вразливостей, пов'язаних із керуванням обліковими даними стосується двох типів атак: перехоплення сесії і використання облікових даних інших користувачів. Для захисту від перехоплення сесії використовується протокол HTTPS, що містить криптографічний захист даних, які передаються. Зловмисник не зможе перехопити ідентифікатори сесії чи іншу зашифровану інформацію, якщо не буде володіти даними про використовувані ключі. Для уникнення використання слабких паролів даних веб-додаток має строгу парольну політику. Кожен пароль користувача має містити щонайменше літеру великого регістру, літеру малого регістру, цифру і бути не меншим за 8 символів. Таким чином зменшуються шанси того, що користувач використає слабкий пароль для облікового запису. Усі паролі у базі даних хешуються із використанням солі за допомогою алгоритму bcrypt. Таким чином, зловмисник не зможе розкрити паролі користувачів, якщо отримає доступ до бази даних, бо кожному паролю буде відповідати свій унікальний хеш.

Для захисту від ескалації привілеїв використовується Spring Security. За допомогою Spring Security був обмежений доступ до сторінок адміністративного функціоналу лише тим користувачам, що мають роль ADMIN; до сторінок користувацького функціоналу лише тим користувачам, що мають роль ADMIN чи USER; усі інші сторінки може переглядати будь-хто. Розподіл доступу до ресурсів окремого користувача базується на отриманні даних про користувача під час виконання коду і не базується на посиланнях.

ВИСНОВКИ

У дипломній роботі було побудовано веб-додаток із засобами та механізмами захисту від трьох загроз: загрози порушеної аутентифікації, SQL-ін’єкції, ескалація повноважень. Захист від загроз порушеної аутентифікації включав у себе захист від перехоплення сесії і захист від атак, що стосуються керування обліковими даними.

У першій частині дипломної роботи було проведено аналіз принципів роботи веб-додатків. Аналіз включав у себе перегляд нормативно-правової бази в галузі інформаційної діяльності і безпеки інформації, визначення структури веб-додатків та можливих архітектур, розгляд мережевого протоколу прикладного рівня SOAP та архітектурного стилю побудови REST. У першій частині було поставлено завдання на виконання у подальших частинах дипломної роботи згідно із проведеним аналізом.

У другій частині дипломної роботи було розглянуто найбільш поширені вразливості веб-додатків згідно із джерелами [21, 22]. В результаті аналізу джерела [23] було подано узагальнену статистику вразливостей веб-додатків. У другій частині розглядалися вразливості, пов’язані із наступними загрозами:

- Порушена аутентифікація
- SQL-ін’єкції
- Недоліки у конфігураціях безпеки
- Міжсайтовий скриптинг (XSS)
- Порушений контроль доступу
- Міжсайтова підробка запиту (CSRF)

У третій частині дипломної роботи на основі проаналізованих у першій та другій частині даних був побудований веб-додаток із засобами та механізмами захисту від загроз порушеної аутентифікації, SQL-ін’єкцій, ескалації привілеїв.

Захист від загроз порушеної аутентифікації включає в себе захист від перехоплення сесії і захист від атак, що стосуються керування обліковими даними. Для захисту від перехоплення сесії використовувався протокол захищеного обміну

даними HTTPS. Для його налаштування був згенерований цифровий сертифікат, поміщений у папку ресурсів проекту. За допомогою нього відбувається налаштування криптографічного захисту. Для захисту від атак, що стосуються керування обліковими даними, використовується надійний алгоритм хешування для секретної інформації.

Захист від SQL-ін’єкцій забезпечується ескейпуванням та підготовленими запитами, що генеруються за допомогою програмного каркасу Spring.

Захист від ескалації привілеїв забезпечується налаштуванням доступу до сторінок веб-додатку за допомогою інструменту Spring Security. За допомогою Spring Security було налаштовано доступ до сторінок загального доступу, сторінок користувачів та сторінок адміністрування. Доступ до ресурсів окремого користувача відбувається не через прив’язку посилання до даних користувача, а через програмний код, що у реальному часі визначає аутентифікованого користувача.

Виходячи із поставленої мети дипломної роботи були виконані наступні завдання:

- визначено найбільш поширені вразливості, властиві сучасним веб-додаткам
- проаналізовано можливі засоби та механізми захисту від загроз, що стосуються визначених вразливостей
- створено веб-застосунок і впровадити деякі із зазначених засобів та механізмів захисту
- надано відповідні рекомендації щодо упровадження засобів та механізмів захисту веб-додатків на основі створеного захищеного веб-застосунку

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thousands of Zoom video calls left exposed on open Web [Electronic resource]: The Washington Post. – Access: https://www.washingtonpost.com/technology/2020/04/03/thousands-zoom-video-calls-left-exposed-open-web.

2. CS:GO, Valve Source games vulnerable to hacking using Steam invites [Electronic resource]: Bleeping Computer. – Access: https://www.bleepingcomputer.com/news/security/cs-go-valve-source-games-vulnerable-to-hacking-using-steam-invites.

3. Parkhomenko, I., Makarenko, A. Web Application Security / Ivan Parkhomenko, Anton Makarenko // PCSITS. – Kyiv, 2021.

4. Конституція України, редакція від 01.01.2020 [Електронний ресурс]: Закон України № 254к/96-ВР від 01.01.2020. – Режим доступу: https://zakon.rada.gov.ua/laws/show/254%D0%BA/96-%D0%B2%D1%80#Text.

5. Про інформацію [Електронний ресурс]: Закон України № 2657-XII від 16.06.2020. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2657-12#Text.

6. Про захист персональних даних [Електронний ресурс]: Закон України № 2297-VI від 23.04.2021. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17#Text.

7. Про затвердження Положення про організацію заходів із забезпечення інформаційної безпеки в банківській системі України [Електронний ресурс]: Постанова Правління Національного банку України №95 від 28.09.2017. – Режим доступу: https://zakon.rada.gov.ua/laws/show/v0095500-17#Text.

8. NIST Special Publication 800-63B. Digital Identity Guidelines. Authentication and Lifecycle Management.

9. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1.

10. ISO/IEC 27000:2018. Information technology – Security techniques – Information security management systems – Overview and vocabulary.

11. Hoffman, A. Web Application Security. Exploitation and Countermeasures for Modern Web Applications / Andrew Hoffman. – O'Reilly Media, Sebastopol, 2020.

12. Web application architecture: Components, models and types [Electronic resource]: ScienceSoft. – Access: https://www.scnsoft.com/blog/web-application-architecture.

13. Gos, K., Zabierowski, W. The Comparison of Microservice and Monolithic Architecture / Konrad Gos, Wojciech Zabierowski // Memstech. – Lviv, 2020.

14. Monolithic architecture vs microservices [Electronic resource]: Divante. – Access: https://divante.com/blog/monolithic-architecture-vs-microservices.

15. Shklar, L., Rosen, R. Web Application Architecture. Principles, Protocols and Practices / Leon Shklar, Richard Rosen. – John Wiley & Sons Ltd, Chichester, 2003.

16. XML WSDL [Electronic resource]: w3schools. – Access: https://www.w3schools.com/xml/xml_wsdl.asp.

17. SOAP Version 1.2 Part 0: Primer (Second Edition) [Electronic resource]: World Wide Web Consortium. – Access: https://www.w3.org/TR/soap12-part0.

18. Различия REST и SOAP [Электронный ресурс]: Хабр. – Режим доступа: https://habr.com/ru/post/483204.

19. XML Soap [Electronic resource]: w3schools. – Access: https://www.w3schools.com/xml/xml_soap.asp.

20. Fielding, R. Architectural Styles and the Design of Network-based Software Architectures / Roy Thomas Fielding // University of California – Irvine, 2000. – Access: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

21. OWASP Top Ten [Electronic resource]: OWASP. – Access: https://owasp.org/www-project-top-ten.

22. 2020 CWE Top 25 Most Dangerous Software Weaknesses [Electronic resource]: Common Weakness Enumeration. – Access: https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html.

23. Web application vulnerabilities and threats: statistics for 2019. – Positive Technologies.

24. What Is Broken Authentication? [Electronic resource]: auth0. – Access: https://auth0.com/blog/what-is-broken-authentication.

25. Hassan, M., Nipa, S., Akter, A. and others. Broken Authentication and Session Management Vulnerability: A Case Study of Web Application / Md. Maruf Hassan, Shamima Sultana Nipa, Marjan Akter and others // International Journal of Simulation: Systems, Science & Technology. – 2018.

26. Baitha, A., Vinod, S. Session Hijacking and Prevention Technique / Anuj Kumar Baitha, Smitha Vinod // International Journal of Engineering & Technology. – 2018.

27. Louis, J., Feng, X. Detection of Session Hijacking / Jerry Louis, Xiaohua Feng // University of Bedfordshire. – 2011.

28. RFC 793: Transmission Control Protocol.

29. Whitaker, A., Newman, D. Penetration Testing and Network Defense / Andrew Whitaker, Daniel P. Newman. – Cisco Press, Indianapolis, 2006.

30. Jain, V., Sahu, D., Tomar, D. Session Hijacking: Threat Analysis and Countermeasures / Vineeta Jain, Divya Rishi Sahu, Deepak Singh Tomar // International Conference on Futuristic Trends in Computational analysis and Knowledge management. – Greater Noida, 2015.

31. 2020 Data Breach Investigations Report – Verizon.

32. Most hacked passwords revealed as UK cyber survey exposes gaps in online security [Electronic resource]: The National Cyber Security Centre. – Access: https://www.ncsc.gov.uk/news/most-hacked-passwords-revealed-as-uk-cyber-survey-exposes-gaps-in-online-security.

33. Know Your Attackers: 2020 CrowdStrike Services Report Key Findings [Electronic resource]: CrowdStrike. – Access: https://www.crowdstrike.com/blog/2019-services-report-key-findings-part-1.

34. Auth0 Reveals 50,000 Unique IP Addresses Make Credential Stuffing Attempts on Daily Basis [Electronic resource]: auth0. – Access: https://auth0.com/blog/auth0-reveals-50000-credential-stuffing-attempts.

35. Pauli, J. The Basics of Web Hacking. Tools and Techniques to Attack the Web / Josh Pauli. – Elsevier, Waltham, 2013.

36. Abdoulaye, D., Pathan, A. A Survey on SQL Injection: Vulnerabilities, Attacks, and Prevention Techniques / Diallo Abdoulaye Kindy, Al-Sakib Khan Pathan // 2011 IEEE 15th International Symposium on Consumer Electronics (ISCE). – Singapore, 2011.

37. A1:2017-Injection [Electronic resource]: OWASP. – Access: https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.

38. SQL Injection [Electronic resource]: w3schools. – Access: https://www.w3schools.com/sql/sql_injection.asp.

39. Hwang, D. SQL Injection / Drew Hwang // California State Polytechnic University, Pomona. – Access: http://hwang.cisdept.cpp.edu/swanew/Text/SQL-Injection.htm.

40. Clarke, J. SQL Injection Attacks and Defense. Second Edition / Justin Clarke. – Elsevier, Waltham, 2012.

41. A6:2017-Security Misconfiguration [Electronic resource]: OWASP. – Access: https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.

42. Shalini, S., Usha, S. Prevention of Cross-Site Scripting Attacks (XSS) On Web Applications in The Client Side / S. Shalini, S. Usha // International Journal of Computer Science Issues. – 2011.

43. Kocher, P., Horn, J., Fogh, A. and others. Spectre Attacks: Exploiting Speculative Execution / Paul Kocher, Jann Horn, Anders Fogh and others.

44. Kocher, P., Horn, J., Fogh, A. and others. Meltdown: Reading Kernel Memory from User Space / Paul Kocher, Jann Horn, Anders Fogh, and others.

45. Firefox представил новую архитектуру безопасности браузера с изоляцией сайтов [Электронный ресурс]: Хабр. – Режим доступа: https://habr.com/ru/company/cloud4y/blog/558236.

46. Cross Site Scripting Prevention Cheat Sheet [Electronic resource]: OWASP. – Access:

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

47. DOM based XSS Prevention Cheat Sheet [Electronic resource]: OWASP. – Access: https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html.

48. Content Security Policy Cheat Sheet [Electronic resource]: OWASP. – Access: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

49. Access control vulnerabilities and privilege escalation [Electronic Resource]: PortSwigger. – Access: https://portswigger.net/web-security/access-control.

50. Cross Site Request Forgery (CSRF) [Electronic resource]: OWASP. – Access: https://owasp.org/www-community/attacks/csrf.

51. Cross-site request forgery (CSRF) [Electronic resource]: PortSwigger. – Access: https://portswigger.net/web-security/csrf.

52. CSRF tokens [Electronic resource]: PortSwigger. – Access: https://portswigger.net/web-security/csrf/tokens.

53. temzgief-project-spring [Electronic resource]: GitHub. – Access: https://github.com/WaifuAnton/temzgief-project-spring/tree/security.

54. Supported Platforms: MySQL Database [Electronic resource]: MySQL. – Access: https://www.mysql.com/support/supportedplatforms/database.html.

55. Анализ SSL/TLS трафика в Wireshark [Электронный ресурс]: Хабр. – Режим доступа: https://habr.com/ru/company/billing/blog/261301.

ДОДАТКИ

ДОДАТОК А

SQL-скрипт для створення бази даних shopdb2

```
-- MySQL Workbench Forward Engineering


SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZE
RO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';


-- -----------------------------------------------------

-- Schema shopdb2

-- -----------------------------------------------------

DROP USER IF EXISTS `shopdb2`@`localhost` ;


DROP SCHEMA IF EXISTS `shopdb2` ;

-- -----------------------------------------------------

-- Schema shopdb2

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `shopdb2` DEFAULT CHARACTER SET utf8 COLLATE
utf8_bin ;


CREATE USER IF NOT EXISTS `shopdb2`@`localhost` IDENTIFIED BY
'Super1_Secret2_Password3_db' ;


GRANT ALL PRIVILEGES ON `shopdb2`.* TO `shopdb2`@`localhost` ;


USE `shopdb2` ;


-- -----------------------------------------------------

-- Table `shopdb2`.`categories`
```

```
-- -----------------------------------------------------

DROP TABLE IF EXISTS `shopdb2`.`categories` ;


CREATE TABLE IF NOT EXISTS `shopdb2`.`categories` (
  `id` BIGINT(10) NOT NULL,
  `name` VARCHAR(256) NOT NULL,
  `picture` VARCHAR(128) NOT NULL,
  `parent_id` BIGINT(10) NULL DEFAULT NULL,
  `create_date` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_categories_categories1`
    FOREIGN KEY (`parent_id`)
    REFERENCES `shopdb2`.`categories` (`id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 7
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_bin;


CREATE UNIQUE INDEX `name_UNIQUE` ON `shopdb2`.`categories` (`name` ASC) INVISIBLE;


CREATE INDEX `fk_categories_categories1_idx` ON `shopdb2`.`categories` (`parent_id` ASC)
VISIBLE;



-- -----------------------------------------------------
-- Table `shopdb2`.`users`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `shopdb2`.`users` ;
```

```
CREATE TABLE IF NOT EXISTS `shopdb2`.`users` (
  `id` BIGINT(10) NOT NULL,
  `email` VARCHAR(255) NOT NULL,
  `password` VARCHAR(63) NOT NULL,
  `role` VARCHAR(15) NOT NULL,
  `create_date` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_bin;

CREATE UNIQUE INDEX `email_UNIQUE` ON `shopdb2`.`users` (`email` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `shopdb2`.`orders`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `shopdb2`.`orders` ;

CREATE TABLE IF NOT EXISTS `shopdb2`.`orders` (
  `id` BIGINT(10) NOT NULL,
  `user_id` BIGINT(10) NOT NULL,
  `total` DECIMAL(10,2) UNSIGNED NOT NULL DEFAULT 0,
  `status` VARCHAR(15) NOT NULL DEFAULT 'REGISTERED',
  `create_date` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_orders_users1`
    FOREIGN KEY (`user_id`)
```

```
  REFERENCES `shopdb2`.`users` (`id`)
  ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_bin;

CREATE INDEX `fk_orders_users1_idx` ON `shopdb2`.`orders` (`user_id` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `shopdb2`.`products`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `shopdb2`.`products` ;

CREATE TABLE IF NOT EXISTS `shopdb2`.`products` (
 `id` BIGINT(10) NOT NULL,
 `name` VARCHAR(255) NOT NULL,
 `picture` VARCHAR(127) NOT NULL,
 `color` VARCHAR(15) NOT NULL,
 `manufacture_date` DATE NOT NULL,
 `description` VARCHAR(4096) NULL DEFAULT NULL,
 `price` DECIMAL(10,2) UNSIGNED NOT NULL,
 `amount` INT(10) UNSIGNED NOT NULL DEFAULT '0',
 `category_id` BIGINT(10) NOT NULL,
 `create_date` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
 PRIMARY KEY (`id`),
 CONSTRAINT `fk_products_categories1`
  FOREIGN KEY (`category_id`)
  REFERENCES `shopdb2`.`categories` (`id`)
  ON DELETE CASCADE)
```

```
ENGINE = InnoDB
AUTO_INCREMENT = 5
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_bin;

CREATE INDEX `fk_products_categories1_idx` ON `shopdb2`.`products` (`category_id` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `shopdb2`.`orders_products`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `shopdb2`.`orders_products` ;

CREATE TABLE IF NOT EXISTS `shopdb2`.`orders_products` (
  `order_id` BIGINT(10) NOT NULL,
  `product_id` BIGINT(10) NOT NULL,
  CONSTRAINT `fk_cart_has_products_cart1`
    FOREIGN KEY (`order_id`)
    REFERENCES `shopdb2`.`orders` (`id`),
  CONSTRAINT `fk_cart_has_products_products1`
    FOREIGN KEY (`product_id`)
    REFERENCES `shopdb2`.`products` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_bin;

CREATE INDEX `fk_cart_has_products_products1_idx` ON `shopdb2`.`orders_products` (`product_id` ASC) VISIBLE;

CREATE INDEX `fk_cart_has_products_cart1_idx` ON `shopdb2`.`orders_products` (`order_id` ASC) VISIBLE;
```

```
-- -----------------------------------------------
-- Table `shopdb2`.`clothes`
-- -----------------------------------------------
DROP TABLE IF EXISTS `shopdb2`.`clothes` ;

CREATE TABLE IF NOT EXISTS `shopdb2`.`clothes` (
  `id` BIGINT(10) NOT NULL,
  `size` VARCHAR(5) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_clothes_products1`
    FOREIGN KEY (`id`)
    REFERENCES `shopdb2`.`products` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_clothes_products1_idx` ON `shopdb2`.`clothes` (`id` ASC) VISIBLE;

USE `shopdb2`;

DELIMITER $$

USE `shopdb2`$$
DROP TRIGGER IF EXISTS `shopdb2`.`orders_products_BEFORE_INSERT` $$
USE `shopdb2`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `shopdb2`.`orders_products_BEFORE_INSERT`
BEFORE INSERT ON `shopdb2`.`orders_products`
FOR EACH ROW
BEGIN
```

```
    DECLARE product_price DOUBLE;
    SET product_price = (SELECT price FROM products WHERE id = NEW.product_id);
    UPDATE products SET amount = amount - 1 WHERE id = NEW.product_id;
    UPDATE orders SET total = total + product_price WHERE id = NEW.order_id;
END$$


USE `shopdb2`$$
DROP TRIGGER IF EXISTS `shopdb2`.`orders_products_AFTER_DELETE` $$
USE `shopdb2`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `shopdb2`.`orders_products_AFTER_DELETE`
AFTER DELETE ON `shopdb2`.`orders_products`
FOR EACH ROW
BEGIN
    DECLARE product_price DOUBLE;
        SET product_price = (SELECT price FROM products WHERE id = OLD.product_id);
        UPDATE products SET amount = amount + 1 WHERE id = OLD.product_id;
    UPDATE orders SET total = total - product_price WHERE id = OLD.order_id;
END$$


DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- ----------------------------------------------------
-- Data for table `shopdb2`.`categories`
-- ----------------------------------------------------
```

```
START TRANSACTION;

USE `shopdb2`;

INSERT INTO `shopdb2`.`categories` (`id`, `name`, `picture`, `parent_id`, `create_date`, `last_update`) VALUES (1, 'cars', 'images/cars/main.jpg', NULL, DEFAULT, DEFAULT);

INSERT INTO `shopdb2`.`categories` (`id`, `name`, `picture`, `parent_id`, `create_date`, `last_update`) VALUES (2, 'clothes', 'images/clothes/main.png', NULL, DEFAULT, DEFAULT);

INSERT INTO `shopdb2`.`categories` (`id`, `name`, `picture`, `parent_id`, `create_date`, `last_update`) VALUES (3, 'women', 'images/clothes/women.jpg', 2, DEFAULT, DEFAULT);

INSERT INTO `shopdb2`.`categories` (`id`, `name`, `picture`, `parent_id`, `create_date`, `last_update`) VALUES (4, 'men', 'images/clothes/men.jpg', 2, DEFAULT, DEFAULT);


COMMIT;



-- -----------------------------------------------------

-- Data for table `shopdb2`.`products`

-- -----------------------------------------------------

START TRANSACTION;

USE `shopdb2`;

INSERT INTO `shopdb2`.`products` (`id`, `name`, `picture`, `color`, `manufacture_date`, `description`, `price`, `amount`, `category_id`, `create_date`, `last_update`) VALUES (1, 'Mango 77055933-TO', 'images/clothes/Mango77055933-TO.jpg', 'BLUE', '2019-02-02', NULL, 499.99, 100, 3, DEFAULT, DEFAULT);

INSERT INTO `shopdb2`.`products` (`id`, `name`, `picture`, `color`, `manufacture_date`, `description`, `price`, `amount`, `category_id`, `create_date`, `last_update`) VALUES (2, 'Mango 77072888-08', 'images/clothes/Mango77072888-08.jpg', 'WHITE', '2018-01-01', NULL, 1199.49, 150, 3, DEFAULT, DEFAULT);

INSERT INTO `shopdb2`.`products` (`id`, `name`, `picture`, `color`, `manufacture_date`, `description`, `price`, `amount`, `category_id`, `create_date`, `last_update`) VALUES (3, 'Tom Tailor 19243430107', 'images/clothes/tom_tailor19243430107.jpg', 'WHITE', '2020-03-05', NULL, 899.29, 120, 3, DEFAULT, DEFAULT);

INSERT INTO `shopdb2`.`products` (`id`, `name`, `picture`, `color`, `manufacture_date`, `description`, `price`, `amount`, `category_id`, `create_date`, `last_update`) VALUES (4, 'Leo Pride 2000660001481', 'images/clothes/leo_pride_2000660001481.jpg', 'BLACK', '2017-08-01', NULL, 1999.00, 50, 3, DEFAULT, DEFAULT);


COMMIT;
```

```
-- ----------------------------------------------------
-- Data for table `shopdb2`.`clothes`
-- ----------------------------------------------------
START TRANSACTION;
USE `shopdb2`;
INSERT INTO `shopdb2`.`clothes` (`id`, `size`) VALUES (1, 'M');
INSERT INTO `shopdb2`.`clothes` (`id`, `size`) VALUES (2, 'S');
INSERT INTO `shopdb2`.`clothes` (`id`, `size`) VALUES (3, 'S');
INSERT INTO `shopdb2`.`clothes` (`id`, `size`) VALUES (4, 'M');

COMMIT;
```

ДОДАТОК Б

Програмне налаштування параметрів безпеки веб-додатку

```
package com.shop.config.security;

import com.shop.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class CustomSecurityConfigurer extends WebSecurityConfigurerAdapter {
    private UserService userService;
    private PasswordEncoder passwordEncoder;

    @Autowired
    public CustomSecurityConfigurer(UserService userService, PasswordEncoder passwordEncoder) {
        this.userService = userService;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userService).passwordEncoder(passwordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
        http
                .csrf().ignoringAntMatchers("/login", "/logout")
                .and()
                .authorizeRequests()
                .antMatchers("/categories/**").permitAll()
                .antMatchers("/admin/**").hasAuthority("ADMIN")
                .antMatchers("/cart/**").hasAnyAuthority("USER", "ADMIN")
                .antMatchers("login").anonymous()
                .antMatchers("logout").authenticated()
                .and()
                .formLogin().loginPage("/login").defaultSuccessUrl("/").usernameParameter("email")
                .and()

.logout().logoutUrl("/logout").logoutSuccessUrl("/").logoutSuccessHandler(userService).invalidateHttpSe
ssion(true).clearAuthentication(true);
    }

    @Bean
    public AuthenticationManager customAuthenticationManager() throws Exception {
        return authenticationManager();
    }
}
```

ДОДАТОК В

Сервіс функціоналу кошику користувача

```
package com.shop.service;

import com.shop.config.constant.Constants;
import com.shop.entity.Order;
import com.shop.entity.Product;
import com.shop.entity.User;
import com.shop.enumeration.Status;
import com.shop.repository.OrderRepository;
import com.shop.repository.ClothesRepository;
import com.shop.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

import java.util.List;
import java.util.Optional;

@Service
public class CartService {
    private OrderRepository orderRepository;
    private ClothesRepository clothesRepository;
    private ProductRepository productRepository;

    @Autowired
    public CartService(OrderRepository orderRepository, ClothesRepository clothesRepository,
ProductRepository productRepository) {
        this.orderRepository = orderRepository;
```

```
        this.clothesRepository = clothesRepository;
        this.productRepository = productRepository;
    }

    public void addProductToCart(long productId) {
        User currentUser = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        Order order;
        Optional<Order> optionalOrder;
        if ((optionalOrder = orderRepository.findByUserIdAndStatus(currentUser.getId(),
Status.CREATED)).isPresent())
            order = optionalOrder.get();
        else {
            order = new Order();
            order.setUser(currentUser);
            orderRepository.save(order);
        }
        order.getProducts().add(clothesRepository.findById(productId).orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, Constants.NO_PRODUCT)));
        orderRepository.save(order);
    }

    public void removeProductFromCart(long productId) {
        User currentUser = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        Order order;
        Optional<Order> optionalOrder;
        if ((optionalOrder = orderRepository.findByUserIdAndStatus(currentUser.getId(),
Status.CREATED)).isPresent()) {
            order = optionalOrder.get();
            order.getProducts().remove(clothesRepository.findById(productId).orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, Constants.NO_PRODUCT)));
            orderRepository.save(order);
        }
    }

    public void submitOrdersInCart() {
```

```
        User currentUser = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        List<Order> orders = orderRepository.findAllByUserIdAndStatus(currentUser.getId(),
Status.CREATED);
        orders.forEach((order) -> order.setStatus(Status.REGISTERED));
        orderRepository.saveAll(orders);
    }

    public Page<Product> getClothesInCart(int page, String sortField, boolean descending) {
        User currentUser = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        List<Order> orders = orderRepository.findAllByUserIdAndStatus(currentUser.getId(),
Status.CREATED);
        return descending
                ? productRepository.findAllByOrders(orders, PageRequest.of(page,
Constants.PRODUCTS_PER_PAGE, Sort.by(sortField).descending()))
                : productRepository.findAllByOrders(orders, PageRequest.of(page,
Constants.PRODUCTS_PER_PAGE, Sort.by(sortField).ascending()));
    }
}
```