Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Кваліфікаційна робота

Освітній ступінь – бакалавр

На тему: «**АНАЛІЗ БЕЗПЕКОВИХ ЗАГРОЗ ДЛЯ КЛІЄНТ-СЕРВЕРНИХ ЗАСТОСУНКІВ І МЕТОДИ ЗАХИСТУ ВІД НИХ**»

Виконав студент
4-го року навчання
122 «Комп’ютерні науки»
Цабут Денис Владиславович

Керівник курсової роботи
доцент Олецький О.В.

Рецензент _____

Кваліфікаційна робота захищена

з оцінкою _____

Секретар ЕК _____

“___” _____ 2023 р.

Київ 2023

Міністерство освіти і науки України НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ» Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
доц., к.ф.-м.н.
_____ С. С. Гороховський
“__” _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту _____Цабуту Денису_____
___4_____ курсу факультету інформатики
ТЕМА: Аналіз загроз клієнт-серверним застосункам та методи захисту від загроз
Вихідні дані: тренувальний модуль до загроз клієнт-серверним застосункам із відповідними методами їхнього вирішення
Зміст ТЧ до кваліфікаційної роботи:

Вступ
1. Характеристика безпекових загроз клієнт-серверної архітектури ПЗ
2. Методи боротьби з безпековими загрозами клієнт-сервер застосунків
3. Створення тест модуля з вразливостей клієнт-серверної архітектури
Висновки
Список джерел
Дата видачі “__” _____ 202_ р.

Керівник _____ Завдання отримано _____

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проєкту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	10.2022	
2	Дослідження вразливостей клієнт-серверних застосунків	01.2023	
3	Характеристика вразливостей клієнт-серверних застосунків	03.2023	
4	Дослідження характеристик фреймворків програмування	04.2023	
5	Опис технічного завдання розробки додатку	04.2023	
6	Створення тренувального модуля	05.2023	
7	Консультація з науковим керівником	14.05.2023	
8	Остаточне оформлення пояснювальної роботи.	19.05.2023	
9	Захист кваліфікаційної роботи.	30.05.2023	

Студент _____Цабут Д.В._____ _____

Керівник _____Олецький О.В._____ _____

“_____” _____ _____ р.

Зміст

Анотація

У цій роботі розглядається безпекова складова застосунків, заснованих на клієнт-серверній архітектурі: основні типи загроз, що можуть впливають на таке програмне забезпечення, аналіз методів захисту від загроз, що безпосередньо спіткають клієнт-серверні застосунки.

Вступ

У сучасних реаліях інформаційного суспільства вимоги до легкого та швидкого доступу до інформації через мережу Інтернет завжди залишаються актуальними як для підприємств різного типу, так і неприбуткових організацій. Такі вимоги задовольняються, зазвичай, за допомогою програмного забезпечення, що є більш загальновідомим як веб-сайт. Це програмне забезпечення в свою чергу переважно відповідає конкретному типу архітектури – а саме клієнт-серверній архітектурі ПЗ.

Мета кваліфікаційної роботи – створити тренувальний модуль з безпеки клієнт-серверних застосунків з попереднім аналізом та характеристикою клієнт-серверної архітектури, огляду та характеристики вразливостей, що притаманні такому типу архітектури, компонентів, що найбільш на них впливають. Огляд типів загроз, що є притаманними такому програмному забезпеченню, співставлення та дослідження методів захисту від них у контексті сучасних викликів спілкування в мережі.

Текстова частина курсової роботи включає 3 розділи.

Перший розділ розглядає клієнт-серверну архітектуру як загальний підхід до архітектури ПЗ та аналізує уразливості, що йому зазвичай

притаманні. Другий розділ розглядає сучасні методи захисту даних та засоби досягнення безпеки у проєктуванні та використанні клієнт-серверних застосунків. Третій розділ визначає набір найбільш поширених вразливостей у вигляді тренувального модуля, що покликаний продемонструвати обраний набір вразливостей та запропонувати набір методів щодо захисту додатків клієнт-серверної архітектури, що вже були розглянуті та охарактеризовані у попередніх розділах.

РОЗДІЛ 1: Характеристика безпекових загроз для клієнт-серверної архітектури програмного забезпечення

Клієнт-серверна архітектура – це поширений підхід до побудови сучасного програмного забезпечення, у якому основні функції розподілені між двома компонентами: клієнтами та серверами. Узагальнено, функції між компонентами розділені наступним чином: клієнти надсилають запити на сервер для обробки конкретної операції ініційованої клієнтом, сервер оброблює запит та виконує необхідні дії з формування відповіді та надсилає дані назад клієнту.

Клієнт-серверні застосунки будуються, спираючись на принцип розподілення програмних функцій та логіки між компонентами клієнтів та серверів. Найбільш поширеною сферою відповідальності сервера є бізнес логіка застосунку, зберігання оброблюваних даних та інші функції, тоді як клієнт забезпечує безпосередню можливість користувачів взаємодіяти із системою через інтерфейс користувача.

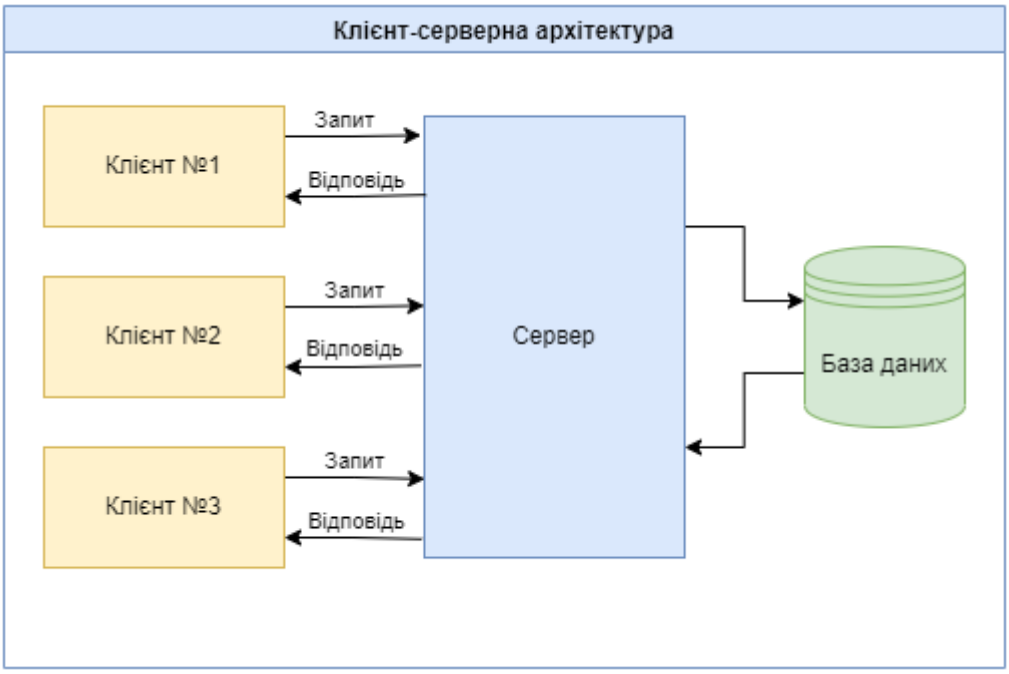


Рисунок 1.1 Структура програмних застосунків заснованих на клієнт-серверній архітектурі

Архітектура клієнт-сервер розроблена з метою забезпечити можливість кільком клієнтам отримувати доступ до одного сервера одночасно. З точки зору імплементації та безпосереднього використання можна виділити наступні тези як переваги клієнт серверної архітектури:

- Легкість у керуванні – керувати файлами досить легко, оскільки всі вони зберігаються на одному сервері. Мережі клієнт-сервер мають найкраще керування для відстеження та пошуку записів необхідних файлів.
- Доступність – клієнти можуть входити в систему незалежно від свого місцезнаходження чи вибраної платформи, що дозволяє кожному співробітнику отримувати доступ до своєї корпоративної інформації без використання режиму терміналу чи процесора.

- Масштабованість – клієнт-серверна мережа має високу масштабованість. Щоразу, коли користувач потребує, він може додати більше ресурсів, таких як клієнти та сервери, таким чином збільшуючи розмір сервера без будь-яких перерв. Через те, що сервер є централізованим, дозвіл на доступ до мережевих ресурсів не є проблемою, тому для конфігурацій потрібно дуже мало співробітників, навіть якщо розмір збільшується.
- Централізоване керування – мережі клієнт-сервер мають перевагу централізованого керування, оскільки вся інформація зберігається в одному місці. Оскільки адміністратор мережі має повний контроль над керуванням і адмініструванням, це особливо корисно. У результаті будь-які проблеми, які виникають у всій мережі, можна вирішувати з одного централізованого місця. У результаті оновлювати дані та ресурси стало набагато легше.

Однак серед недоліків можна виділити наступні позиції:

- Необхідність регулярного технічного обслуговування – сервери працюватимуть у мережі безперервно. Це означає, що їх потрібно належним чином обслуговувати. Якщо є якісь проблеми, їх потрібно негайно усувати. Менеджер мережі повинен бути призначений для обслуговування сервера.
- Вартість – у мережі клієнт-сервер вартість налаштування та обслуговування сервера зазвичай дуже висока, так само, як і мережеві операції. Оскільки мережі потужні, їх придбання може бути дорогим. Таким чином, не всі користувачі зможуть ними скористатися.
- Можливість перевантаженості мережі. Якщо сервер отримуватиме занадто багато запитів від багатьох клієнтів – це може спричинити

несправності та уповільнення з’єднання для користувачів. Доступ до даних у таких умовах буде суттєво ускладнений.

- Безпека. Оскільки клієнти та сервери спілкуються через мережу, вони вразливі до різних типів атак, включаючи крадіжку даних, атаки на відмову в обслуговуванні та інші види зловмисної діяльності. Для захисту від цих загроз безпеці програми клієнт-сервер вимагають надійних заходів безпеки для забезпечення конфіденційності, цілісності та доступності даних і системи в цілому.

У рамках цієї роботи буде детально розглянуто безпекову складову клієнт-серверної архітектури.

1.1 Характеристика типів безпекових загроз клієнт-серверних застосунків

Спираючись на вже розглянуті базові компоненти клієнт-серверних застосунків можна виділити 3 наступні групи безпекових загроз:

- Мережеві
- Програмні
- Фізичні

Ці загрози виникають на основі тих чи інших залежностей робочих компонентів застосунку: робота по мережі, вразливості у коді програми тощо.

1. Мережеві загрози: Мережеві загрози стосуються атак, спрямованих на мережеву інфраструктуру, яка є основою клієнт-серверних програм. Ці загрози включають атаки на відмову в обслуговуванні (DoS), атаки на розподілену відмову в обслуговуванні (DDoS), атаки типу "людина посередині" (MITM) і перехоплення пакетів.

a. Атаки на відмову в обслуговуванні (DoS): DoS-атаки включають заповнення сервера запитами, доки він більше не зможе обробляти запити, що призводить до того, що сервер стає недоступним для користувачів.
b. Розподілені атаки на відмову в обслуговуванні (DDoS): DDoS-атаки схожі на DoS-атаки, але включають кілька комп'ютерів, що значно ускладнює зупинку атаки.
c. Атаки "людина посередині" (MITM): Атаки MITM передбачають перехоплення зловмисником зв'язку між клієнтом і сервером. Тоді зловмисник може прочитати, змінити або вставити повідомлення в комунікацію, не усвідомлюючи жодної сторони.
d. Сніфінг пакетів: Принцип перехоплення пакетів полягає в тому, що зловмисник перехоплює та читає пакети даних, що передаються через мережу.

2. Програмні загрози: Загрози на основі програми стосуються атак, які використовують вразливі місця в коді програми, що може призвести до несанкціонованого доступу до даних, впровадження коду тощо. Ці загрози включають атаки впровадження SQL, атаки міжсайтового скриптингу (XSS) і атаки переповнення буфера.
a. Атаки SQL ін'єкції: Атаки SQL-ін'єкції передбачають впровадження шкідливого коду в запити SQL для отримання несанкціонованого доступу до даних або зміни бази даних.
b. Атаки міжсайтового скриптингу (XSS): XSS-атаки передбачають впровадження шкідливого коду на веб-сторінки, які переглядають інші користувачі, що може призвести до крадіжки конфіденційної інформації.

c. Атаки переповнення буфера: Атаки переповнення буфера передбачають використання вразливостей у коді програми для перезапису даних у пам’яті та виконання шкідливого коду.

3. Фізичні загрози: Фізичні загрози стосуються загроз, які включають фізичний доступ до сервера, що може призвести до крадіжки даних, пошкодження апаратного забезпечення тощо.

1.2 Аналіз безпекових загроз для клієнт-серверних застосунків у мережевому середовищі

Оскільки мережа є невід’ємним елементом архітектури клієнт-серверних застосунків та програмного забезпечення, осмислення мережевих загроз під час визначення структури ПО є невід’ємною складовою його успішної побудови. Таким чином, необхідно розглянути весь спектр потенційних загроз. Як вже будо зазначено у попередньому розділі, мережеві загрози – це такі загрози, що здебільшого притаманні мережевій інфраструктурі, але для того щоб продовжити більш глибокий аналіз необхідно, для початку, дати більш конкретне визначення мережевим загрозам.

Безпекова загроза – це подія, умова або обставина, результатом якої є спричинення втрати або пошкодження даних та/або мережевих ресурсів. Серед видів та форм, яких можуть набувати такі втрати, здебільшого виражаються у наступних формах:

- Зміна структури або цілісності даних: знищення, розголошення або модифікація даних
- Відмова надання послуг (DoS)
- Шахрайство та вимагання грошей

Розглянемо основні типи загроз, з якими безпосередньо мають справу системи клієнт-серверної архітектури.

Як вже було зазначено у попередньому розділі, можна виділити наступні позиції:

1. DoS/DDoS атаки
2. MITM
3. Сніфінг

Атака на відмову в обслуговуванні (DoS) переповнює сервер трафіком, роблячи веб-сайт або ресурс недоступними. Розподілена атака типу «відмова в обслуговуванні» (DDoS) – це атака DoS, яка використовує кілька комп'ютерів або машин для затоплення цільового ресурсу. Обидва типи атак перевантажують сервер або веб-програму з метою переривання служб.

Оскільки сервер заповнюється більшою кількістю пакетів протоколу керування передачею/протоколу дейтаграм користувача (TCP/UDP), ніж він може обробити, він може вийти з ладу, дані можуть бути пошкоджені, а ресурси можуть бути неправильно спрямовані або навіть вичерпані до паралізуючої системи.

Принципова відмінність між DoS-атакою та DDoS-атакою полягає в тому, що перша – це атака «система на систему», тоді як остання передбачає атаку декількох систем на одну систему.

Нижче наведено деякі типові форми таких атак:

- Краплеподібна атака — це DoS-атака, яка надсилає в мережу велику кількість фрагментів даних Інтернет-протоколу (IP). Коли мережа намагається зібрати фрагменти в оригінальні пакети, вона не може це

зробити. Наприклад, зловмисник може взяти дуже великі пакети даних і розбити їх на кілька фрагментів, щоб цільова система могла знову зібратися. Однак зловмисник змінює спосіб розбирання пакета, щоб заплутати цільову систему, яка потім не може повторно зібрати фрагменти в оригінальні пакети.

- Атака flooding — це DoS-атака, яка надсилає кілька запитів на сервер, але потім не відповідає для завершення процесу. Наприклад, зловмисник може надсилати різні запити на підключення як клієнт, але коли сервер намагається зв'язатися, щоб перевірити з'єднання, зловмисник не відповідає. Після повторення процесу стає завантаженим запитами, що очікують на розгляд, що реальні клієнти не можуть підключитися, і сервер стає «зайнятим» або виходить з ладу.
- Атака фрагментації IP – це тип DoS-атаки, яка доставляє змінені мережеві пакети, які приймаюча мережа не може повторно зібрати. Мережа загрузає громіздкими нерозібраними пакетами, використовуючи всі її ресурси.
- Об'ємна атака – це тип DDoS-атаки, який використовується для цільових ресурсів пропускної здатності. Наприклад, зловмисник використовує ботнет, щоб надсилати велику кількість пакетів запитів у мережу, переповнюючи її пропускну здатність ехо-запитами протоколу керуючих повідомлень Інтернету (ICMP). Це сповільнює або припиняє роботу систем.
- Атака на протокол — це тип DDoS-атаки, яка використовує слабкі місця на рівнях 3 і 4 моделі OSI. Використовується послідовність підключення TCP, надсилаючи запити, але не відповідаючи повноцінно, або відповідаючи іншим запитом, використовуючи

підроблену IP-адресу джерела. Запити без відповіді використовують ресурси мережі, поки вона не стає недоступною.

- Атака на основі програми – це тип DDoS-атаки, націлений на рівень 7 моделі OSI. Атака Slowloris, під час якої зловмисник надсилає часткові запити HTTP, але не виконує їх. HTTP-заголовки періодично надсилаються для кожного запиту, в результаті чого мережеві ресурси стають зв’язаними. Під час атаки, поки сервер не зможе встановити нові з’єднання. Цей тип атаки дуже важко виявити, тому що замість надсилання пошкоджених пакетів вони надсилають часткові пакети та не використовують пропускну здатність.

Іншою широко розповсюдженою атакою на клієнт-серверні застосунки є MITM (Man in the middle). Розглянемо принцип роботи даної атаки.

Атака MITM – це форма кібератаки, що являє собою переривання передачі даних, призначених для іншого отримувача, які взагалі не повинні надсилатися, без розпізнавання будь-якого учасника, поки не стане надто пізно.

Для цієї атаки можна виділити 2 основні етапи:

1. Перехоплення: Щоб отримати доступ до мережі, зловмисники зазвичай використовують відкриті або неналежним чином захищені маршрутизатори Wi-Fi. Вони також можуть маніпулювати DNS-серверами. Їхня мета – знайти слабкі паролі, але вони також можуть скористатися підробкою IP-адреси або отруєнням кешу. Коли доступ отримано, дані жертви будуть зібрані за допомогою інструментів збору даних.
2. Розшифровка: Під час цієї фази перехоплені дані декодуються

Перехоплення (sniffing) – це перехоплення та моніторинг трафіку в мережі. Відбувається за допомогою ПЗ, яке фіксує всі пакети даних, що проходять через даний мережевий інтерфейс, або за допомогою апаратних пристроїв, явно призначених для цієї мети. Атака з перехопленням відбувається, коли використовується сніфер пакетів для перехоплення та читання тих чи іншихданих, що проходять мережею. Зазвичай, цілями цих атак є незашифровані повідомлення електронної пошти, облікові дані для входу та фінансова інформація.

У деяких випадках зловмисники також можуть використовувати інструменти атаки з перехопленням і аналізатори пакетів, щоб вставити шкідливий код у нешкідливі пакети даних у спробі зламати комп'ютер або інші пристрої цілі.

Існує два основних типи атаки: пасивна та активна:

- Під час атаки пасивного перехоплення хакер відстежує трафік, що проходить через мережу, без втручання. Цей тип атаки може бути корисним для збору інформації про цілі в мережі та типи даних (наприклад, облікові дані для входу, повідомлення електронної пошти), які вони передають. Оскільки це не передбачає жодного втручання в цільові системи, воно також має меншу ймовірність викликати підозру, ніж інші типи атак.
- Активне перехоплення – це тип атаки, який передбачає надсилання створених пакетів до однієї або кількох цілей у мережі для вилучення конфіденційних даних. Використовуючи спеціально створені пакети, зловмисники часто можуть обійти заходи безпеки, які в іншому випадку захистили б дані від перехоплення. Активне перехоплення може також передбачати впровадження шкідливого коду в цільові

системи, що дозволяє взяти під контроль або викрасти конфіденційну інформацію.

Таким чином, для убезпечення від таких загроз розробники програмного забезпечення повинні приділяти особливу увагу тим рішенням з безпеки, що дозволяють уникнути таких проблем. Серед таких способів захиститися можна виділити наступні:

- Ідентифікація та автентифікація користувачів
- Шифрування трафіку від програми до користувача
- Контроль доступу до інформації.

Конкретні методи реалізації цих способів буде розглянуто в рамках наступних розділів.

1.3 Аналіз програмних безпекових загроз для клієнт-серверних застосунків

У цьому розділі досліджуються різні типи програмних загроз, що є притаманними клієнт-серверним застосункам. Розглядаються методи, які використовують зловмисники для використання вразливостей на прикладному рівні, що порушує конфіденційність, цілісність і доступність системи. Розуміючи ці загрози, розробники та системні адміністратори можуть запровадити ефективні контрзаходи для захисту програм клієнт-сервер.

Нижче наведено характеристику вразливостей, що є безпосередньо викликаними недоліками програмного коду:

1. Порушений контроль доступу на стороні клієнта

Недостатній контроль доступу JavaScript до клієнтських елементів (даних і коду), компрометація конфіденційних даних або маніпулювання DOM із зловмисною метою.

2. XSS + CSRF

Міжсайтовий скриптинг (XSS) – це такий тип вразливостей, під час яких шкідливі скрипти додаються до оброблюваних системою даних тим чи іншим чином. XSS-атаки відбуваються, коли зловмисник використовує веб-програму для надсилання шкідливого коду, як правило, у формі скрипта на стороні клієнта (наприклад браузера), іншому кінцевому користувачеві. Недоліки, які дозволяють цим атакам бути успішними, досить широко поширені та виникають у будь-якому місці, де веб-додаток використовує вхідні дані від користувача в межах вихідних даних, які він генерує, без їх перевірки чи кодування.

Підсумуємо, які загальні характеристики є притаманними цьому типу атаки:

- Включає веб-сайти, які покладаються на ідентифікацію користувача
- Використовує довіру веб-сайту до цієї особи
- Обманом веб-браузер користувача надсилає HTTP-запити на цільовий сайт
- Включає запити HTTP, які мають негативні наслідки

3. Ін’єкція HTML – ще один тип міжсайтових скриптових атак, ін’єкція HTML передбачає ін’єкцію HTML-коду через уразливі розділи веб-сайту. Зазвичай метою впровадження HTML є зміна дизайну веб-сайту або інформації, що відображається на веб-сайті.

4. Перенаправлення URL-адреси на стороні клієнта або відкрите перенаправлення – у цьому типі атаки програма приймає ненадійні дані, які містять значення URL-адреси, що змушує веб-програму перенаправляти користувача на іншу, ймовірно, шкідливу сторінку, контрольовану зловмисником.
5. Впровадження каскадних таблиць стилів (CSS) – зловмисники вставляють довільний код CSS на веб-сайт, який потім відображається в браузері кінцевого користувача. Залежно від типу корисного навантаження CSS атака може призвести до міжсайтового скрипта, модифікації інтерфейсу користувача або викрадання конфіденційної інформації, наприклад даних кредитної картки.

Окрему групу вразливостей варто виділити, як набір сталих порад та принципів, яких необхідно дотримуватися під час розробки ПЗ:

1. Включення конфіденційної інформації на стороні клієнта

Наявність конфіденційної бізнес-логіки, коментарів розробників, власних алгоритмів або системної інформації, що міститься в клієнтському коді або збережених даних.

2. Конфіденційні дані, що зберігаються на стороні клієнта

Зберігання конфіденційних даних, таких як паролі, криптосекрети, маркери API або дані ідентифікаційної інформації, у постійному сховищі на стороні клієнта, як-от LocalStorage, кеш-пам’ять браузера, або тимчасовому сховищі, як-от змінні JavaScript на рівні даних.

3. Невикористання стандартних елементів безпеки браузера

Невикористання звичайних засобів контролю безпеки на основі стандартів, вбудованих у браузери, таких як пісочниці iframe, і заголовків безпеки, як-от Content Security Policy (CSP), цілісності підресурсу та багатьох інших стандартних функцій безпеки.

4. Вразливі та застарілі компоненти

Відсутність виявлення та оновлення бібліотек JavaScript, які застаріли або містять відомі вразливості. Вразливі та застарілі компоненти, які зосереджено в бібліотеках на стороні клієнта становлять безпосередню небезпеку цілісності та працеспроможності додатку.

Таким чином, програмні вразливості клієнт-серверних застосунків – є такими вразливостями, що спричиняються безпосередніми техніками та підходами до програмування, що були використані або не використані під час створення програми. Такі

РОЗДІЛ 2: Методи боротьби з безпековими загрозами клієнт-серверних застосунків

У даному розділі буде детально та ґрунтовно розглянуто сучасні загально визнані методи щодо боротьби з безпековими загрозами, що притаманні застосункам побудованим на основі клієнт-серверної архітектури програмного забезпечення.

2.1 Огляд методів автентифікації та контролю доступу

Цей підрозділ має на меті забезпечити повне розуміння того, як забезпечити автентифікацію та контроль доступу в програмах клієнт-сервер. Впроваджуючи найкращі методи автентифікації та контролю доступу,

розробники можуть зменшити ризик неавторизованого доступу та забезпечити конфіденційність, цілісність і доступність своїх програм та співставити їх вирішуваними загрозами.

Автентифікація – це процес підтвердження особи користувача або системи. Це передбачає підтвердження того, що користувач або система є тими, за кого себе видають, на основі їхніх облікових даних, таких як ім’я користувача та пароль, біометричні дані або цифрові сертифікати.

Автентифікація є важливим аспектом безпеки в різних контекстах, включаючи комп’ютерні системи, мережі та онлайн-сервіси. Це гарантує, що лише авторизовані особи або системи можуть отримати доступ до конфіденційних даних або ресурсів, запобігаючи несанкціонованому доступу та потенційним порушенням безпеки.

Існує кілька типів методів автентифікації, зокрема:

1. Автентифікація на основі пароля: користувач надає ім’я користувача та пароль, які перевіряються системою.
2. Біометрична автентифікація: користувач надає біометричні дані, такі як відбитки пальців, розпізнавання обличчя або сканування райдужної оболонки ока для перевірки.
3. Аутентифікація на основі сертифіката: користувач надає цифровий сертифікат, виданий довіреною третьою стороною, щоб підтвердити свою особу.
4. Багатофакторна автентифікація: користувач надає кілька форм автентифікації, таких як пароль і сканування відбитків пальців, для підвищення безпеки.

Автентифікація часто поєднується з авторизацією, яка визначає, до яких ресурсів або дій автентифікований користувач або система може отримати доступ або виконати їх. Разом автентифікація та авторизація складають основу механізмів контролю доступу, які регулюють доступ до конфіденційних даних або ресурсів.

Фреймворки авторизації – це інструменти та методології, які використовуються для регулювання доступу до ресурсів або дій у системі. Ці структури допомагають забезпечити доступ до конфіденційних даних або ресурсів тільки авторизованим користувачам або системам. Існує кілька типів платформ авторизації, кожна з яких має власний підхід до контролю доступу. Деякі з найпопулярніших фреймворків авторизації:

1. Контроль доступу на основі ролей (RBAC): RBAC – це широко використовувана структура авторизації, яка призначає дозволи доступу на основі ролі користувача в організації. Користувачам призначаються ролі відповідно до їхніх посадових обов’язків, а права доступу надаються відповідно до ролі.
2. Контроль доступу на основі атрибутів (ABAC): ABAC – це структура авторизації, яка використовує такі атрибути, як ідентифікація користувача, посада, місцезнаходження та інші фактори для визначення прав доступу. Цей підхід забезпечує більшу гнучкість, ніж RBAC, оскільки дозволи на доступ можуть базуватися на кількох атрибутах, а не на ролі користувача.
3. Контроль доступу на основі правил (RBAC): RBAC – це структура авторизації, яка використовує набір правил для визначення прав

доступу. Ці правила можуть базуватися на таких даних, як час доби, місцезнаходження та поведінка користувача.

4. Обов'язковий контроль доступу (MAC): MAC – це структура авторизації, яка забезпечує суворий контроль доступу на основі класифікації безпеки. Права доступу визначаються центральною політикою безпеки, яка визначає, які користувачі або системи можуть отримувати доступ до певних ресурсів або дій.
5. Шифрування на основі атрибутів (ABE): ABE – це структура авторизації, яка шифрує дані за допомогою таких атрибутів, як ідентифікатор або роль користувача. Користувачі можуть отримати доступ до даних, лише якщо вони мають відповідні атрибути для розшифровки даних.
6. OAuth: OAuth – це структура авторизації, яка дозволяє користувачам надавати стороннім програмам доступ до своїх ресурсів, не повідомляючи свої облікові дані, наприклад імена користувачів і паролі. Він надає користувачам стандартизований спосіб авторизувати програми сторонніх розробників для доступу до їхніх ресурсів на різних платформах або службах.

Вибір правильної системи авторизації для системи залежить від різних факторів, включаючи характер ресурсів, що захищаються, розмір і складність системи та необхідний рівень безпеки.

2.2 Автентифікація за допомогою JSON Web Token

Як вже зазначалося у попередньому розділі, алгоритм автентифікації та передачі даних JSON Web Token ґрунтується на використані так званого токена, що містить у собі певний набір інформації про відправника,

шифрування, інші дані тощо. Для продовження імплементації наведемо більш точні визначення використовуваним термінам.

Токен – можна визначити як закодований підпис, що використовується для автентифікації та/або авторизації користувача з метою отримання доступу до певного набору ресурсів у системи або мережі.

Токен, за визначенням, генерується у формі **OTP** (одноразового пароля), що й має на меті продемонструвати, що токен можна використати лише один раз, адже генерація відбувається випадковим чином для кожної виконуваної транзакції.

Автентифікація, що заснована на основі використання токенів з набором параметрів, дозволяє користувачам підтверджувати свою унікальну особу з подальшим отримання унікального токену, що має на меті забезпечувати доступ до набору визначених ресурсів протягом відведеного проміжку часу, що задається логікою системи. Також користувачі можуть отримати доступ до веб-сайту чи мережі, для якої видано токен, і їм не потрібно знову і знову вводити облікові дані, доки термін дії токена не закінчиться.

Токени широко використовуються для регулярних онлайн-транзакцій для підвищення загальної безпеки та точності.

Щоразу, коли виконується транзакція, користувачу необхідно ввести облікові дані. Після надання цих даних, система надсилає OTP на визначений , здебільшого мобільний, пристрій за допомогою текстового повідомлення або електронної пошти.

Генератор токенів генерує ці випадкові одноразові паролі, і користувач проходить автентифікацію, коли вони представлені веб-сайту або додатку.

Користувачеві надсилається випадковий рядок, який зберігається в постійному сховищі, наприклад у веб-сховищі, і з кожним запитом користувача рядок надсилається для автоматичної автентифікації користувача кілька разів протягом терміну служби маркера.

Термін життя токена невеликий. Крім того, задіяна таблиця БД, що містить усі маркери сеансу, зіставлена з ідентифікатором користувача та містить інші відомості, зокрема термін дії, тип пристрою тощо.

Щоразу, коли ви виконуєте транзакцію онлайн, вам потрібно ввести облікові дані. Після того, як ви надаєте облікові дані, система надсилає OTP на мобільний пристрій за допомогою текстового повідомлення або електронної пошти.

Генератор токенів генерує випадкові одноразові паролі, і користувач проходить автентифікацію, коли вони представлені веб-сайту або додатку.

Користувачеві надсилається випадковий рядок, який зберігається в постійному сховищі, наприклад у веб-сховищі, і з кожним запитом користувача рядок надсилається для автоматичної автентифікації користувача кілька разів протягом терміну служби маркера.

Термін життя токена невеликий. Крім того, задіяна таблиця БД, що містить усі токени сеансу, зіставлена з ідентифікатором користувача та містить інші відомості, зокрема термін дії, тип пристрою тощо.

Веб-токен JSON (JWT) – це відкритий стандарт (RFC 7519), який визначає повноцінний спосіб безпечної передачі інформації між сторонами у вигляді об’єкту JSON. Передавана інформація у такому підході має цифровий підпис. JWT можна підтвердити/підписати за допомогою секрету (з

алгоритмом HMAC) або пари з публічних та приватних ключів за допомогою криптографічного алгоритму RSA або ECDSA.

Даний підхід реалізовано як частину тренувального модуля, описаного у розділі 3.

2.3 Обмеження доступу до веб ресурсів

Обмеження доступу (Rate Limiting) – це важливий підхід, яка використовується для контролю та обмеження кількості запитів або дій, дозволених протягом певного періоду часу. Він відіграє важливу роль у захисті систем і ресурсів від зловживань, надмірного використання та потенційних зловмисних дій. У цьому розділі розглядаються різні методи обмеження швидкості та пояснюється, як вони працюють для підтримки стабільності, продуктивності та безпеки системи.

1. Фіксоване обмеження:

Обмеження фіксованої частоти вікон – це простий метод, який встановлює попередньо визначене обмеження на кількість запитів або дій, дозволених у фіксованому часовому вікні – будь-які додаткові запити понед встановлене обмеження будуть відхилені або відкладені до початку наступного часового проміжку. Цей метод гарантує, що швидкість запитів розподіляється рівномірно, проте має недоліки під час піків трафіку.

2. Обмеження доступу (sliding window)

Обмеження швидкості ковзного вікна забезпечує більшу гнучкість порівняно з обмеженням швидкості фіксованого вікна. Замість використання фіксованого часового вікна він розглядає рухоме або ковзаюче часове вікно. По мірі надходження запитів ковзне вікно переміщується, відстежуючи

кількість запитів протягом заданого інтервалу часу. Якщо кількість запитів перевищує допустимий ліміт, подальші запити відхиляються або відкладаються, поки рівень не впаде нижче порогового значення. Обмеження швидкості ковзного вікна забезпечує кращу обробку швидкісного трафіку та пропонує більш плавні переходи обмеження швидкості.

3. Відро токенів:

Алгоритм відра токенів є популярним методом обмеження швидкості, який дозволяє пакети запитів у межах визначеного ліміту. Він працює, підтримуючи маркерне відро, яке поповнюється з постійною швидкістю. Кожен запит споживає певну кількість токенів із відра. Якщо токени доступні, запит обробляється, а токени вираховуються. Однак, якщо відро порожнє, запит або відхиляється, або відкладається, доки не стане доступною достатня кількість токенів. Обмеження швидкості сегментів токенів забезпечує гнучкість обробки як постійного, так і різкого трафіку, якщо загальна швидкість знаходиться в межах попередньо визначеного ліміту.

4. Діряве відро (Leaky Bucket)

Алгоритм leaky bucket — це метод обмеження швидкості, який контролює швидкість вихідних запитів, обробляючи їх із постійною швидкістю. Він забезпечує максимальну швидкість, розглядаючи запити як воду, що ллється у відро. Якщо відро переповнюється, зайві запити або відкидаються, або відкладаються. Цей метод допомагає підтримувати постійну швидкість виведення, запобігаючи раптовим сплескам запитів, які можуть перевантажити систему. Алгоритм витоку відра зазвичай використовується для формування трафіку та реалізації якості обслуговування (QoS).

5. Адаптивне обмеження швидкості

Адаптивне обмеження швидкості динамічно регулює обмеження швидкості на основі різних факторів, таких як навантаження системи, моделі трафіку та доступність ресурсів. Цей метод використовує алгоритми та евристику для аналізу вхідних запитів і відповідного налаштування обмежень швидкості. Це дозволяє точно налаштувати обмеження швидкості на основі умов реального часу, забезпечуючи оптимальне використання ресурсів і оперативність.

6. Білий і чорний список:

Окрім традиційних методів обмеження швидкості, білий і чорний списки можна використовувати для контролю доступу та запобігання зловживанням. Білий список дозволяє певним IP-адресам, користувачам або клієнтам обходити обмеження швидкості, забезпечуючи безперебійний доступ для довірених об’єктів. Чорний список, з іншого боку, блокує або обмежує доступ до певних IP-адрес, користувачів або клієнтів, які демонструють підозрілу або зловмисну поведінку, ще більше посилюючи безпеку та захист від зловживань.

Методи обмеження швидкості є важливими інструментами для підтримки стабільності, продуктивності та безпеки системи шляхом контролю швидкості вхідних запитів або дій. Впроваджуючи відповідні методи обмеження швидкості, такі як фіксоване вікно, ковзне вікно, відро токенів, діряве відро, адаптивне обмеження швидкості та включаючи механізми білого та чорного списків, організації можуть захистити свої системи від зловживань, запобігти виснаженню ресурсів, забезпечити справедливе використання та

підтримувати надійне та безпечне середовище. Необхідно ретельно розглянути метод обмеження швидкості та його параметри

У рамках тренувального модуля буду розглянуто перший зазначений підхід до Rate Limiting.

РОЗДІЛ 3: Створення тренувального модуля з вразливостей клієнт-серверної архітектури

За мету цього розділу взято створення програмного додатку, як тренувального модуля, що має на меті продемонструвати вразливості тих чи інших компонентів веб застосунків (клієнтів та серверів), які можуть бути створенні програмістом під час розробки системи або таких, що є притаманними клієнт-серверним застосункам, як вже було розглянуто у попередніх розділах. У контексті визначеного списку вразливостей, що будуть розглядатися, також будуть розглянуті властивості конкретних засобів розробки запропонованої реалізації, їхні переваги у забезпеченні безпеки застосунку та визначені загальні принципи щодо вирішення тих чи інших безпекових проблем клієнт-серверного ПЗ.

3.1 Технологічний стек для імплементації

Spring Boot – це фреймворк програмування із відкритим вихідним кодом, який полегшує розробникам створення автономних цифрових продуктів і готових до виробництва Spring додатків, включаючи програми Java і веб-сервіси. Він використовує мікрофреймворк, що робить його найбільш корисним для створення мікросервісів для веб-сайтів і мобільних додатків.

Spring Boot існує, щоб надати розробникам надзвичайно швидкий спосіб створювати та розгортати програми та служби. Це дві головні переваги Spring Boot.

Spring Security – це структура, яка дозволяє програмісту використовувати компоненти JEE для встановлення обмежень безпеки для веб-додатків на основі Spring-framework. У двох словах, це бібліотека, яку можна використовувати та налаштовувати відповідно до вимог програміста. Оскільки він є частиною того самого сімейства Spring, що й Spring Web MVC, він добре працює разом. Його основною функцією є керування автентифікацією та авторизацією як на рівні веб-запиту, так і на рівні виклику методу. Можливо. Найсуттєвіша перевага цього фреймворку полягає в тому, що він одночасно міцний і дуже адаптивний. Незважаючи на те, що він дотримується умов налаштувань Spring, програмісти можуть вибирати між положеннями за замовчуванням і змінювати їх відповідно до своїх конкретних вимог. Spring Security працює на основі наступних чотирьох основних концепцій:

- Аутентифікація
- Авторизація
- Зберігання паролів
- Фільтри сервлетів

Переваги Spring Security. Це деякі з основних переваг безпеки Spring

- Захист від таких атак, як фіксація сесії та клікджекінг.
- Підтримка конфігурації Java.
- Портативний
- Інтеграція Servlet API

- Захист від атак грубої сили.

1. Авторизація

Spring Security надає гнучку структуру автентифікації, яка підтримує широкий спектр механізмів автентифікації, включаючи автентифікацію на основі форми, автентифікацію на основі маркерів і автентифікацію через зовнішні постачальники ідентифікаційних даних, такі як OAuth2 і OpenID Connect. Також Spring Security надає надійну та настроювану структуру авторизації, яка дозволяє розробникам визначати точні політики контролю доступу для своїх програм. Це включає підтримку контролю доступу на основі ролей, контролю доступу на основі дозволів та інших більш складних механізмів контролю доступу.

2. Локалізація ПЗ

Ця можливість дозволяє нам створювати інтерфейси користувача для додатків будь-якою мовою.

3. Запам'ятай мене

За допомогою файлів cookie HTTP Spring Security надає цю можливість. Він запам’ятовує користувача та не дозволяє йому входити з однієї робочої станції, доки він не вийде.

4. LDAP (полегшений протокол доступу до каталогу)

Це відкритий прикладний протокол для керування та взаємодії з розсіяними інформаційними службами каталогів через Інтернет-протокол.

5. JAAS (служба автентифікації та авторизації Java) LoginModule

Це Pluggable Authentication Module на основі Java. Він підтримується процедурою автентифікації Spring Security.

6. Аутентифікація через веб-форму

Під час цієї процедури веб-форми збирають і автентифікують облікові дані користувача з веб-браузера. Хоча ми хочемо створити автентифікацію веб-форм, Spring Security підтримує це.

7. Автентифікація дайджест-доступу:

Ми можемо зробити процедуру автентифікації більш безпечною за допомогою цієї функції, ніж за допомогою автентифікації базового доступу. Перш ніж передати конфіденційні дані через мережу, він запитує, щоб браузер підтвердив особу користувача.

8. Авторизація HTTP:

Використовуючи шляхи Apache Ant або регулярні вирази, Spring надає цю функцію для авторизації HTTP URL-адрес веб-запитів.

9. Базова автентифікація доступу:

Spring Security підтримує автентифікацію базового доступу, яка використовується для надання імені користувача та пароля під час виконання мережевих запитів.

10. Захист CSRF

Spring Security забезпечує готовий захист від атак міжсайтової підробки запитів (CSRF), які є поширеним вектором атак для веб-додатків.

3.2 Імплементація захисту від визначених вразливостей

У рамках даного розділу будуть розглянутий набір визначених видів вразливостей, що притаманні клієнт-серверним застосункам у вигляді

тренувального модуля. За підґрунтя реалізації демонстраційного застосунку взято раніше описаний фреймворк Spring Boot.

Першою тренувальною вразливістю серед визначеного списку є IDOR.

IDOR (Insecure Direct Object Reference) – це вразливість, що являє собою надання програмою посилання на внутрішній об’єкт реалізації (тієї чи іншої сутності тощо). Використовуючи цей метод, IDOR розкриває справжній ідентифікатор і формат або шаблон, що запрограмований на використання елементом у БД. Найпоширенішим прикладом є ідентифікатор сутності в базі даних або файловій системі.

IDOR не є атакою чи прямою загрозою безпеці додатку, так як результатом використання цієї потенційної вразливості є формат ідентифікатора, який використовується для доступу до об’єкта. Тим не менш, для певних форматів ідентифікатора IDOR створює потенційну можливість здійснити атаку перерахуванням (Enumeration attack), що полягає у багатьох спробах перевірки ідентифікаторів доступу до пов’язаних з ними об’єктів/сутностей.

Атака на перерахування в свою чергу є способом, утворити колекцію дійсних ідентифікаторів, з використанням виявленого формату або шаблону і перевірки на валідність за допомогою конкретного API.

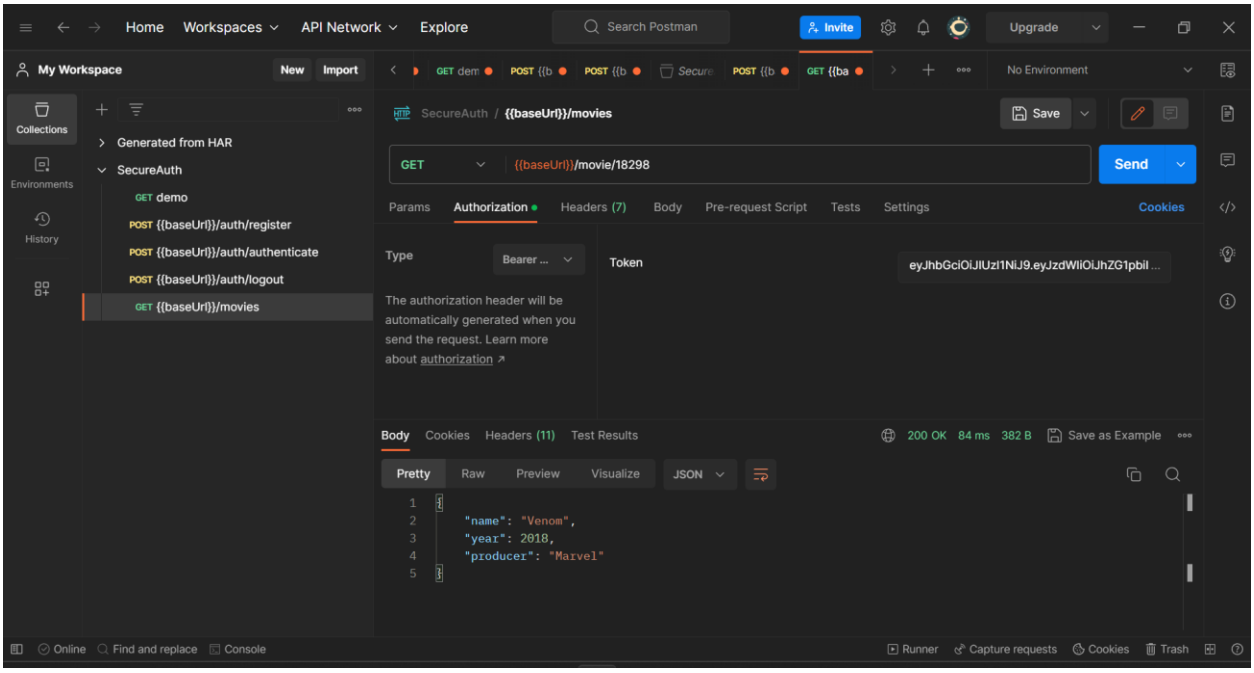


Рисунок 3.1 Запит на отримання сутності за внутрішнім ідентифікатором

Для боротьби з цією вразливістю можна виділити наступний підхід: використовувати альтернативний ідентифікатор для запитів від клієнта або сторонньої системи – хеш ідентифікатора.

Використання хешу дозволяє подолати наступні проблеми:

- Дозволяє уникнути створення таблиці відповідних внутрішніх та зовнішніх ідентифікаторів у кеші сеансу користувача або на рівні програми.
- Максимально знижує можливість успіхи enumeration attack на основі алгоритму хешування з використанням геш солі.

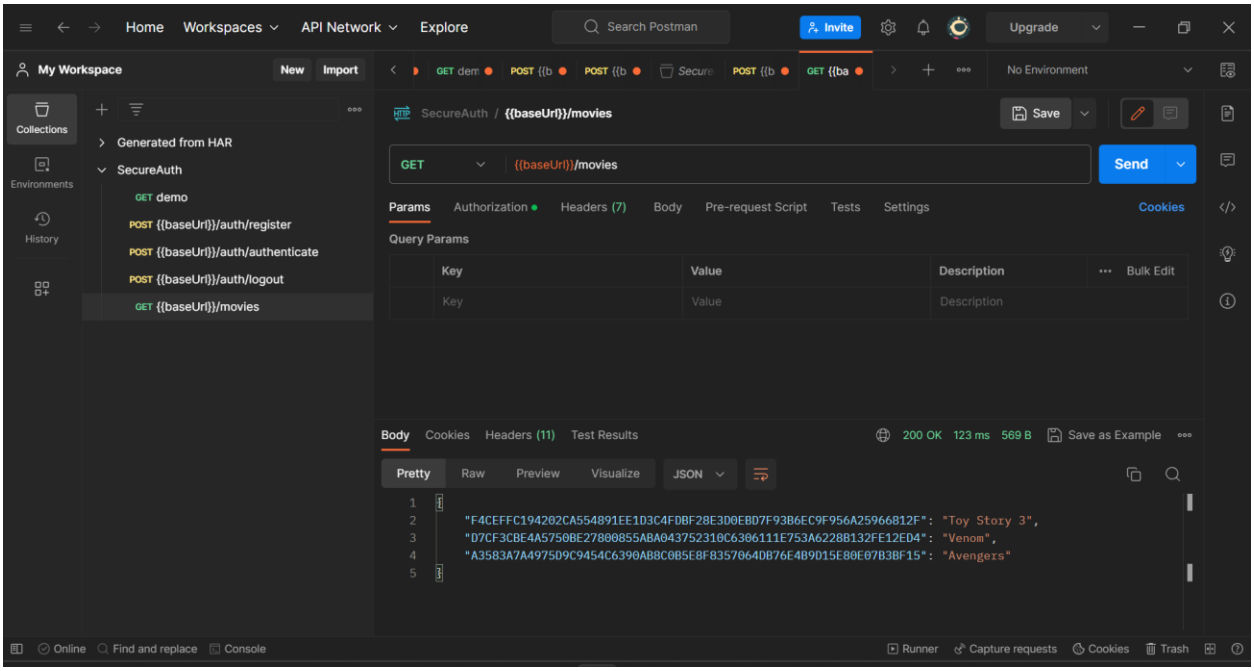


Рисунок 3.1 Запит на отримання всіх сутностей

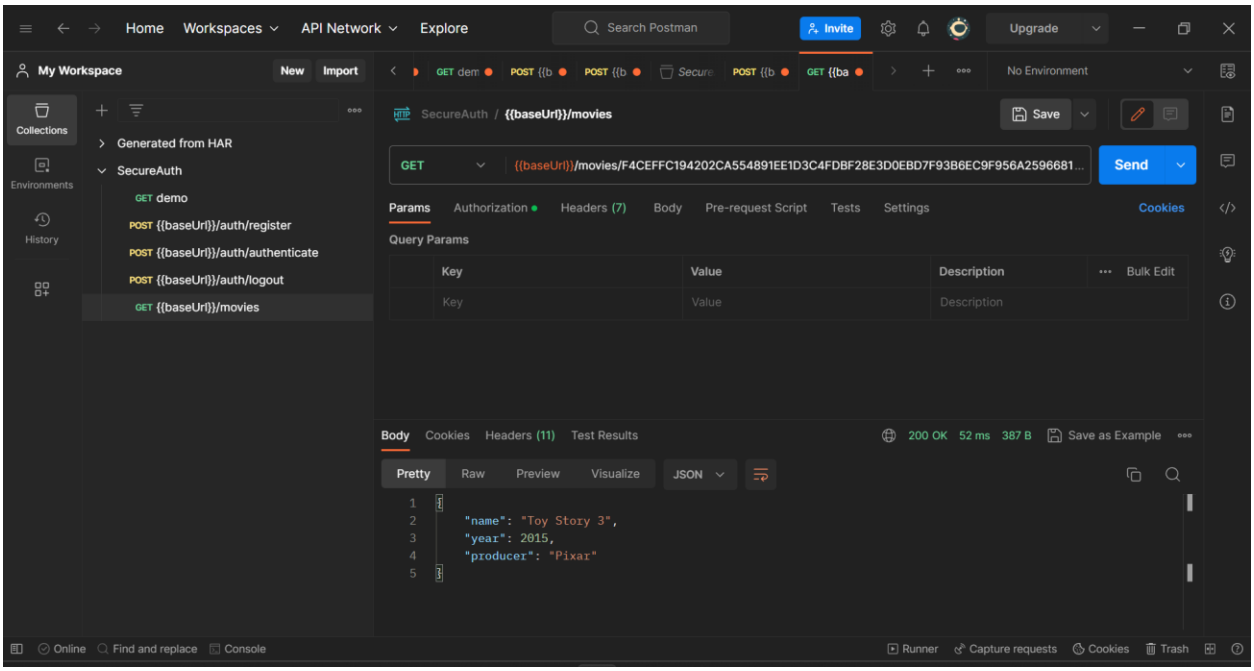


Рисунок 3.2 Запит на отримання конкретної сутності за зовнішнім ідентифікатором

Таким чином, унеможливлюється підбір ідентифікатора й атака на перерахування загалом.

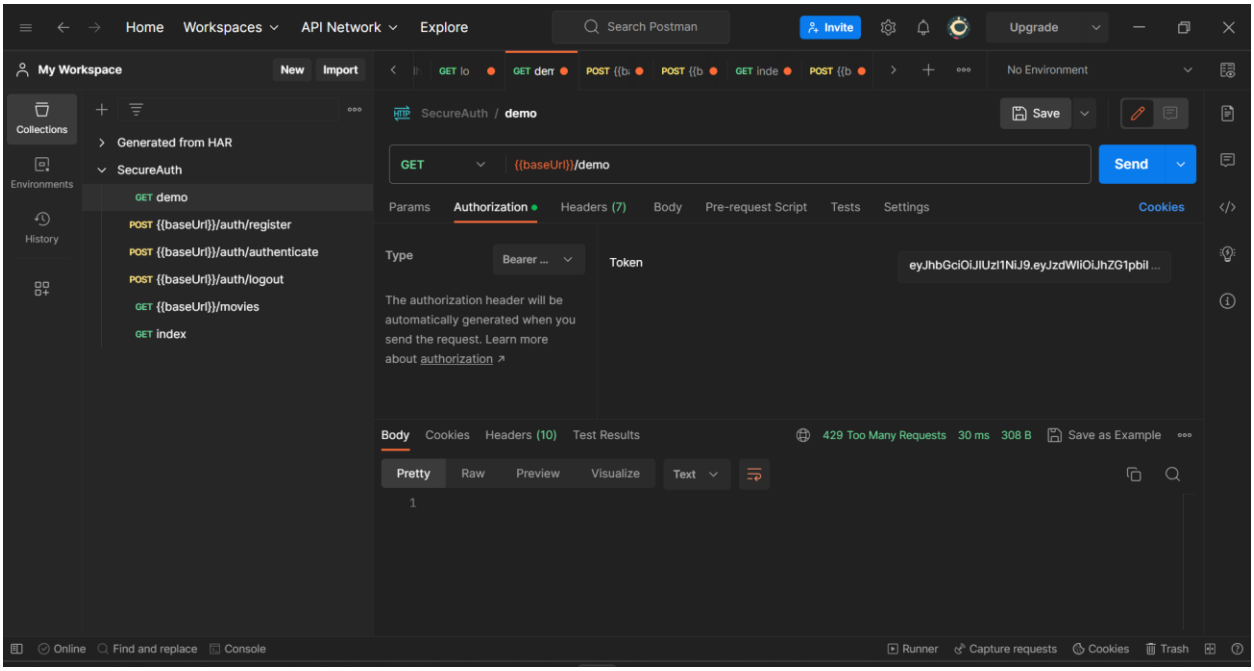
Наступна група атак, що притаманна клієнт-серверному ПЗ – це атаки ін’єкціями. Як вже розглядалося у попередніх розділах атаки.

Наступний тип атак, що розглядається – це атаки на відмову в обслуговуванні.

Як вже розглядалося у попередніх розділах, під час так званих DoS атак зловмисник моделює виснаження ресурсів сервера за допомогою багатьох запитів на доступ. У сучасних умовах існують різні методи боротьби, проте з точки зору сервера до рішення можна підійти наступним чином:

- Визначати IP адресу відправника до БД у пам’яті та блокувати її у випадку надходження надмірної кількості запитів.

Таким чином, у запропонованому тренувальному модулі за допомогою Spring фільтру запит обробляється: зчитується IP адреса відправника та якщо ліміт запитів переважає запрограмований, у відповідь надійде помилка 429 Too Many Requests.



3.3 Реалізація безпечного завантаження файлів

Однією з широкопоширених функцій клієнт-серверних додатків є опція завантаження файлів, таких документи, зображення або мультимедійні файли. Така функція є фундаментальною для певних видів додатків, проте вона має потенціал створювати безпекові ризики. У цьому розділі розглядаються загрози безпеці, пов’язані з веб-завантажувачами файлів, і надається уявлення про ефективне подолання цих загроз.

1. Завантаження шкідливих файлів: завантаження файлів, що містять шкідливе ПЗ – ті чи інші віруси, які можуть скомпрометувати сервер, забезпечити несанкціонований доступ тощо. Зловмисне завантаження файлів може призвести до витоку даних і компрометації системи.
2. Маніпуляції з форматом файлів: змінюючи розширення файлу або використовуючи кілька розширень, можливо змусити систему розглядати потенційно небезпечний файл як нешкідливий.

3. Завантаження великих файлів: завантаження надмірно великих файлів може бути використане як основа для DOS атаки. Навмисно завантажуючи великі файли, ресурси сервера перевантажуються та спричиняють проблеми з продуктивністю або збої на сервері. Запровадження обмежень щодо розміру файлу зменшує ризик DoS атак.

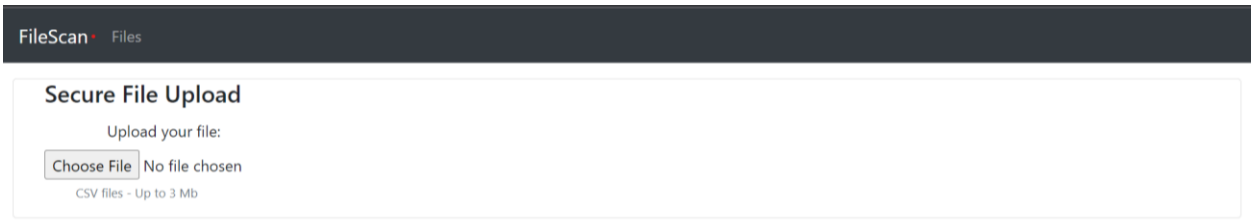


Рисунок 3.3 Стартова сторінка завантаження файлів на сервер

Як частина тренувального модуля імплементований наступний набір методів запобігання вище переліченим загрозам:

1. Перевірка розширення та фільтрація: Застосування суворих механізмів перевірки та фільтрації до завантажених файлів перевіряє тип файлу, розмір і цілісність вмісту, щоб запобігти зловмисному завантаженню, на основі бібліотек Tika.

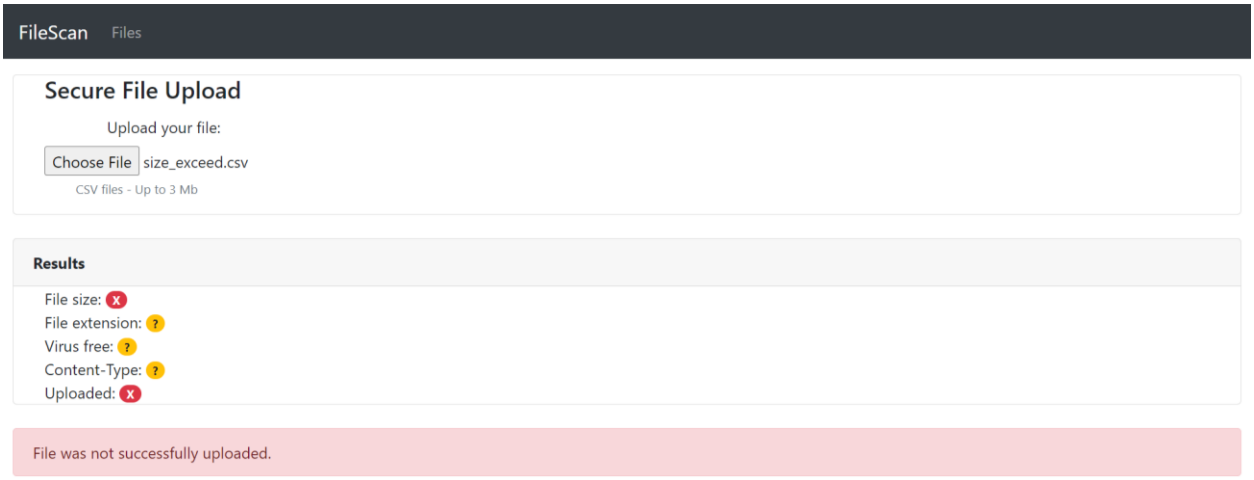


Рисунок 3.3 Спроба завантаження завеликого файлу

2. Безпечне зберігання файлів: під час проходження перевірки завантажені файли зберігаються поза кореневим каталогом програми, у тимчасовій директорії. Це запобігає безпосередньому виконанню завантажених файлів і обмежує потенційну шкоду у разі атаки.
3. Антивірусне Сканування: файли у тимчасовій директорії скануються на віруси за допомогою Cloudmersive public API, що є обов'язковим елементом сканування та допомагає ідентифікувати й заблокувати шкідливі файли.

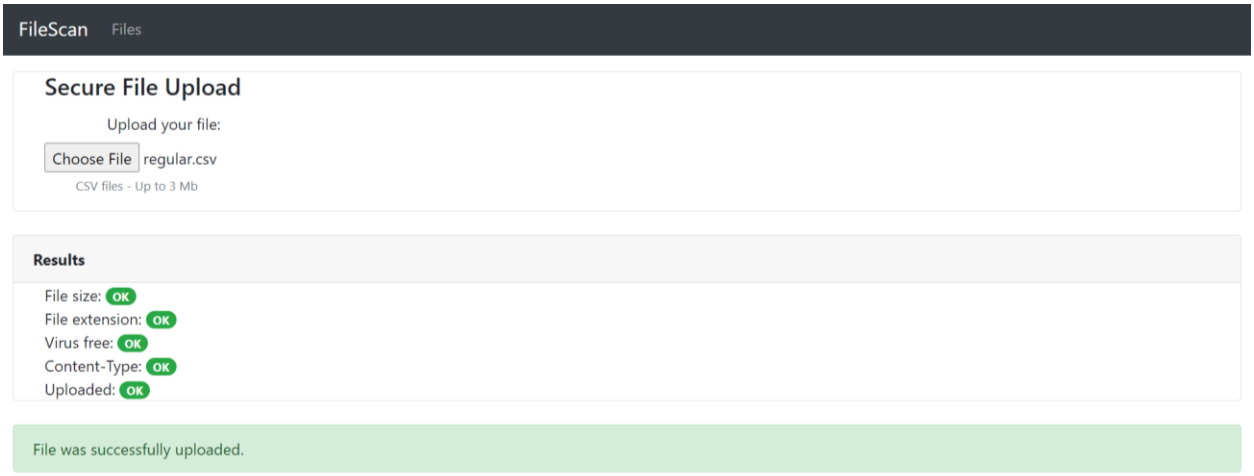


Рисунок 3.3 Сторінка успішно перевіреного та збереженого файлу

Програми для завантаження веб-файлів, незважаючи на те, що вони важливі для взаємодії з користувачем, створюють ризики безпеці, які потрібно ретельно розглядати. Розуміючи та впроваджуючи рекомендовані стратегії пом’якшення, розглянуті в цьому розділі, організації можуть підвищити безпеку своїх функцій завантаження веб-файлів, захищаючи свої системи та інші дані.

3.4 Реалізація повноцінної JWT автентифікації

Як вже було зазначено в попередніх розділах, одним з найбільш затребуваних безпекових елементів клієнт-серверних додатків є контроль доступу, а кількість

Задля практичного опрацювання матеріалу, що було викладено у різних розділах даної роботи, було вирішено реалізувати додаток на основі можливостей, що надаються фреймворком для програмування Spring Boot на Java, що включає в себе 2 основні функції, що відіграють особливу,

фундаментальну роль у застосунках клієнт-серверної архітектури задля вирішення безпекових проблем, з якими найчастіше доводиться зустрічатися у реальних умовах. На основі вище зазначених критеріїв було вибрано наступні позиції:

- Описаний у розділі 3.1 підхід до автентифікації JSON Web Token
- Обмеження Brute Force Authentication

Опишемо процес проходження автентифікації на основі наступної моделі:

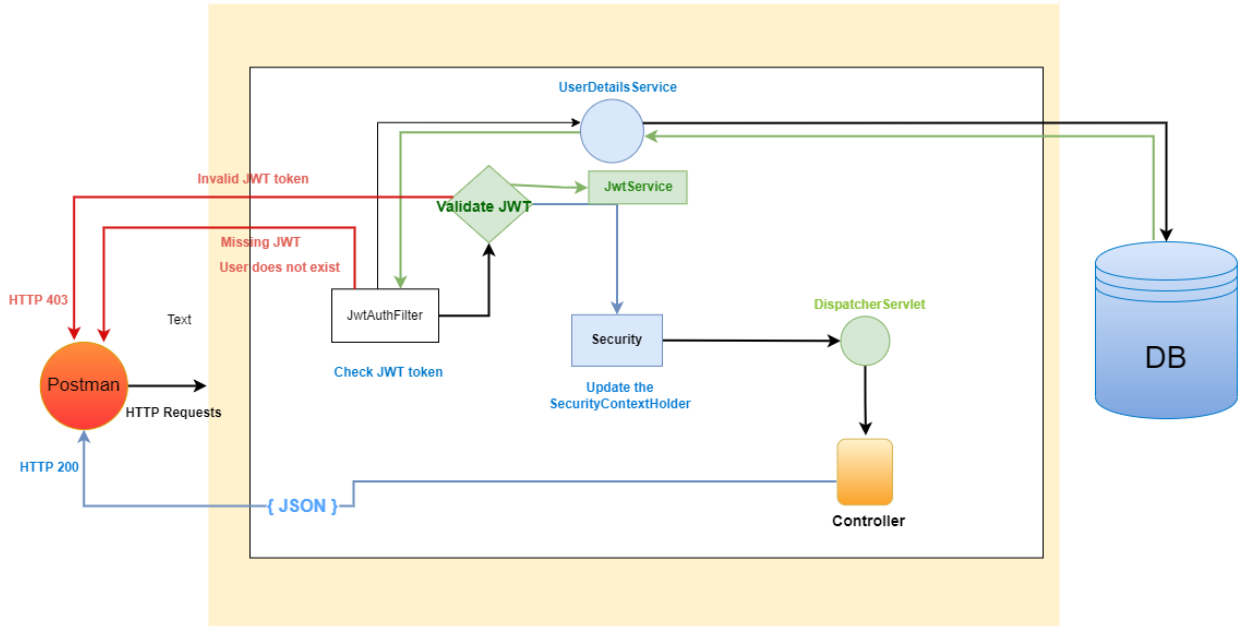


Рисунок 3.1 Структурні компоненти інформаційної

Змодельований процес має початок, коли клієнт надсилає HTTP-запит на сервер, який працює за допомогою Spring Boot container заснованому на вбудованому сервері Apache Tomcat.

Перша дія, що виконується у програмі на фреймворку Spring, це запрограмовані фільтри. Таким чином, у цьому випадку перше, що буде

виконано, це JWT фільтр – фільтр автентифікації, і це фільтр лише один раз на запит, роль якого складається з підтвердження та перевірки JWT токена, що має надійти як частина запиту на сервер

Процес обробки запиту складається з наступних кроків:

- Внутрішня перевірка на присутність JWT token
    - Якщо токен відсутній, сервер надішле клієнту у відповідь код 403
- Фільтр декодує токен аби виокремити різні частини токену, зокрема user email (credentials)
- Фільтр робить запит до UserDetailsService аби спробувати знайти дані користувача за допомогою user email, що є частиною token subject (claim)
- UserDetailsService зробить запит до бази даних аби знайти екземпляр User на основі Email
    - Якщо такого юзера знайдено не буде, сервер поверне помилку 403
- Процес або механізм JWT викликає службу JWT, передаючи на вхід раніше знайденого Юзера класу User та сам отриманий токен
- Відбувається перевірка на дійсність токена (перевірка терміну дії, токен призначений для іншого користувача і т.п.).
    - Якщо токен не дійсний/не відповідає юзеру, Сервер надішле клієнту помилку 403 у відповідь
- Система оновлює Security Context Holder знайденим раніше юзером (для якого було провалідовано токен), що встановить стан цього юзера, як
- Система оновить Authentication Manager, щоб щоразу перевіряти, чи цей користувач автентифікований для наступних запиту
    - Наступний запит буде автоматично оброблено та надіслано на Dispatcher Servlet, з якого його буде перенаправлено безпосередньо на контролер

- Сервер надішле клієнту у відповідь статус 200 разом зі JWT токеном, що має використовуватися для доступу до захищених ресурсів серверу.

Висновки

Під час роботи над дипломною роботою були досліджені та проаналізовані різні відомості про безпекові загрози, що притаманні програмним додаткам, заснованих на основі клієнт-серверної архітектури, та методи, за допомогою яких з ними можна боротися. Були засвоєні особливості їхньої структури, технологічні складові та підходи до розробки. Базуючись на цих матеріалах було створено тренувальний модуль, що демонструє безпекові вразливості та як із ними можна боротися.

Отже, закінчивши роботу над проєктом для кваліфікаційної роботи можна сказати, що мета, зазначена на початку була досягнута, інформаційна частини курсової роботи є змістовною та інформативною. Представлений підхід до програмної реалізації може бути доповненим, покращеним з точки зору розширення демонстрованих вразливостей у рамках тренувального модуля та техніки, що запобігають їм.

Список використаних джерел

1. [Електронний ресурс] Client-server architecture. Дата використання 09.04.2023 https://www.britannica.com/technology/client-server-architecture
2. [Електронний ресурс] Клієнт-серверна архітектура та ролі серверів. Дата використання 08.04.2023 https://medium.com/@IvanZmerzlyi/%D0%BA%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0-%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D1%82%D0%B0-%D1%80%D0%BE%D0%BB%D1%96-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D1%96%D0%B2-9893d8048229
3. [Електронний ресурс] Client Server Architecture – Detailed Explanation. Дата використання 09.04.2023 https://www.interviewbit.com/blog/client-server-architecture/
4. [Електронний ресурс] Client server model. Дата використання 09.04.2023 https://www.geeksforgeeks.org/client-server-model/
5. [Електронний ресурс] OWASP Top 10 Client-Side Security Risks. Дата використання 11.04.2023 https://owasp.org/www-project-top-10-client-side-security-risks/
6. [Електронний ресурс] Top 12 client-side security threats. Дата використання 12.04.2023

https://cybersecurity.att.com/blogs/security-essentials/top-12-client-side-security-threats

7. [Електронний ресурс] DoS Attack vs. DDoS Attack. Дата використання 02.05.2023 https://www.fortinet.com/resources/cyberglossary/dos-vs-ddos
8. [Електронний ресурс] What Are Sniffing Attacks, and How Can They Be Prevented? Дата використання 02.05.2023 https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/what-are-sniffing-attacks/
9. [Електронний ресурс] Packet Sniffing: Types, Methods, Examples, and Best Practices. Дата використання 02.05.2023 https://www.knowledgehut.com/blog/security/packet-sniffing
10. [Електронний ресурс] Spring Boot – Introduction. Дата використання 05.05.2023 https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm
11. [Електронний ресурс] Spring Boot – Introduction. Дата використання 05.05.2023 https://www.baeldung.com/spring-security-5-oauth2-login
12. [Електронний ресурс] Introduction to JSON Web Tokens. Дата використання 06.05.2023 https://jwt.io/introduction
13. [Електронний ресурс] Guava Tutorial. Дата використання 06.05.2023 https://www.tutorialspoint.com/guava/index.htm
14. [Електронний ресурс] What is rate limiting? | Rate limiting and bots. Дата використання 06.05.2023 https://www.cloudflare.com/en-gb/learning/bots/what-is-rate-limiting/
15. [Електронний ресурс] Virus Scan API. Дата використання 15.05.2023

https://api.cloudmersive.com/docs/virus.asp?language=net

16. [Електронний ресурс] Injection attacks. Дата використання 15.05.2023 https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html
17. [Електронний ресурс] Insecure Direct Object Reference – IDOR Vulnerability. Дата використання 13.05.2023 https://www.geeksforgeeks.org/insecure-direct-object-reference-idor-vulnerability/
18. [Електронний ресурс] Cross-Site Scripting. Дата використання 16.05.2023 https://portswigger.net/web-security/cross-site-scripting
19. [Електронний ресурс] Cross Site Request Forgery. Дата використання 15.05.2023 https://www.ibm.com/docs/en/snips/4.6.2?topic=categories-cross-site-request-forgery-csrf-attacks
20. [Електронний ресурс] Which attack is a server side attack. Дата використання 15.05.2023 https://www.alibabacloud.com/tech-news/web-server/giqt1yos1h-which-web-attack-is-a-server-side-attack

Перелік прийнятих скорочень

DBMS – Database Management System

СКБД – Система керування базами даних

БД – база даних

ПЗ – Програмне забезпечення

API – Application Programing Interface

IDOR – Insecure Direct Object Reference