ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Пояснювальна записка

до магістерської роботи
на тему:

«ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ WEB-ДОДАТКУ НА БАЗІ WEBSOCKET»

Виконав студент 6 курсу, групи БСДМ-62
спеціальності 125 Кібербезпека
освітньо-професійної програми «Інформаційна та кібернетична безпека»
(шифр і назва спеціальності)
Кисельов О.В.
(прізвище та ініціали)
Керівник Гайдур Г.І.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)
Нормоконтролер Чумак Н.С.
(прізвище та ініціали)

КИЇВ - 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Інститут ННІЗІ
Кафедра Інформаційної та кібернетичної безпеки
Ступінь вищої освіти Магістр
Спеціальність 125 «Кібербезпека»
Освітньо-професійна програма Інформаційна та кібернетичка безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
_____ Гайдур Г.І.
«__» _____ 2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Кисельову Олексію Володимировичу
(прізвище, ім'я, по батькові)

1.Тема магістерської роботи: «Технологія забезпечення безпеки Web-додатку на базі WebSocket»

керівник магістерської роботи Гайдур Галина Іванівна, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від « 12 » жовтня 2022 року № 122 .

2. Строк подання студентом магістерської роботи 15.12.2022 р.

3. Вихідні дані до магістерської роботи
корпоративна інформаційна система;
програмний комплекс розробки захищеного Web-додатку;
технологія передачі даних в режимі реального часу WebRTC;
бібліотека з відкритим кодом Socket.io
науково-технічна література, експлуатаційна документація, нормативні документи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз проблеми безпеки систем комунікації в реальному часі.
2. Дослідження методів та засобів захисту систем комунікації в реальному часі.
3. Розроблення рекомендацій щодо удосконалення безпеки Web-додатку на базі WebSocket від сучасних загроз.

5. Перелік графічного матеріалу.

1. Об’єкт, предмет, мета та наукові завдання дослідження.
2. Аналіз проблеми безпеки систем комунікації в реальному часі.
3. Аналіз методів та засобів захисту систем комунікації в реальному часі.
4. Розгортання робочого середовища та продукту Socket.io
5. Дослідження та аналіз процесу розробки real-time програмного забезпечення.
6. Програмна реалізація та тестування захищеного Web-додатку на базі WebSocket.
7. Рекомендації щодо захисту інформації в процесі проектування Web-додатку.
8. Висновки за результатами роботи.

6.Дата видачі завдання 13.10.2022р.

КАЛЕНДАРНИЙ ПЛАН

№ зп	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1.	Аналіз проблеми безпеки систем комунікації в реальному часі.	15.10.2022 р.	
2.	Дослідження методів та засобів захисту систем комунікації в реальному часі.	20.10.2022 р.	
3.	Проектування захищеного Web-додатку на базі WebSocket.	30.10.2022 р.	
4.	Розроблення рекомендацій щодо реалізації технології забезпечення безпеки Web-додатка на базі WebSocket.	10.11.2022 р.	
5.	Оформлення результатів дослідження.	25.11.2022 р.	
6.	Підготовка доповіді до захисту.	28.11.2022 р.	

Студент Кисельов О.В.
(підпис) прізвище та ініціали

Керівник магістерської роботи Гайдур Г.І.
(підпис) прізвище та ініціали

РЕФЕРАТ

Текстова частина магістерської роботи: 66 сторінки, 37 рисунки, 32 джерел, 1 таблиця.

Об'єкт дослідження - процес захисту інформації на етапах проектування Web-додатку на базі WebSocket.

Предмет дослідження - технологія забезпечення безпеки Web-додатка на базі WebSocket.

Мета роботи - розробити варіант захисту інформації Web-додатку на базі WebSocket від сучасних загроз та надати рекомендації щодо удосконалення їх безпеки під час проектування.

Методи дослідження - опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів, аналіз існуючих рішень, їх порівняння та проведення експерименту.

В роботі проведено аналіз безпеки Web-додатка на базі WebSocket.

Досліджено методи та засоби захисту Web-додатка на базі WebSocket.

На основі досліджень проведених в роботі розроблено рекомендації щодо застосування методів та засобів проектування захищеного Web-додатка на базі WebSocket.

Галузь використання – технологія забезпечення безпеки Web-додатків.

WEBSOCKET, ІНФОРМАЦІЙНА СИСТЕМА ОРГАНІЗАЦІЇ, КІБЕРБЕЗПЕКА, REACT, NODEJS, SOCKET.IO, OWASP.

ABSTRACT

Master`s thesis: 66 pages, 33 figures, 20 sources.

Object of research - the process of information security at the stages of developing a Web application based on WebSocket.

Subject of research - Web application security technology based on WebSocket.

The aim of research - to develop an option for protecting WebSocket-based Web application information from modern threats and provide recommendations for improving their security during design.

Research methods - study of the literature on this topic, analysis of operational documentation, international standards, analysis of existing solutions, their comparison and conducting an experiment.

The paper analyzes the security of Web-application based on WebSocket.

The methods and means of protection of Web-application based on WebSocket are investigated.

Based on the research conducted in the work developed recommendations for the use of methods and tools for designing secure Web-application based on WebSocket.

Scope - technology to ensure the security of Web-applications.

WEBSOCKET, ORGANIZATION INFORMATION SYSTEM, CYBER SECURITY, REACT, NODEJS, SOCKET.IO, OWASP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX (Asynchronous JavaScript and XML) – технологія звернення до сервера без перезавантаження сторінки

W3C (World Wide Web Consortium) – організація, яка розробляє технологічні стандарти. Прикладами стандартів консорціуму є HTML, XML, CSS, SVG, RSS та WCAG.

WASC (Web Application Security Consortium) – консорціум безпеки Web-додатків

CVSS(Common Vulnerability Scoring System) – загальна система оцінки вразливостей

NIST (National Institute of Standards and Technology) – національний інститут стандартів та технологій

OWASP (Open Web Application Security Project) – відкритий проект забезпечення безпеки Web-додатків

REST (Representational state transfer) – являє собою узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи

SOAP (Simple Object Access Protocol) – мова Web-сервісів і доступу до них

SAST (Static Application Security Testing) – статичний аналіз коду на наявність відомих вразливостей

ВСТУП

Актуальність проблеми захисту інформації при використанні систем комунікації в реальному часі. Сучасний світ – це світ інформаційних технологій, що базується на щоденному використанні приладів для комунікації та обміну інформацією. Людям стало набагато легше знаходити потрібну інформацію, спілкуватися між собою, купувати собі речі і все це, – не виходячи з дому.

Ми все більше використовуємо онлайн-технології для зв'язку. За останній час різко виріс попит на засоби комунікації в реальному часі. Декілька років тому високий запит на системи для спілкування був спричинений у зв'язку з пандемією Covid-19. Люди вимушені були бути на самоізоляції та працювати вдома. Різні бізнес-процеси у компаніях, де важливо бачити співрозмовника, спілкування із співробітниками, навчання у школах та вищих навчальних закладах – вимушено було перейти в режим «онлайн». Схоже відбувається і зараз, під час повномасштабного вторгнення з боку держави-агресора. Цілі родини, розділені на відстані тисячі кілометрів, компанії та бізнеси стали вимушеними переселенцями, але все одно продовжують підтримувати зв'язок та працювати віддалено за допомогою різних систем комунікації в реальному часі. До найвідоміших програм для спілкування можна віднести такі, як: Google Meet, Skype, Zoom. Для ефективного ділового спілкування найчастіше використовують MS Teams, Slack. Їх використання спрощує спілкування між співробітниками, дозволяє відправляти текстові, голосові повідомлення та забезпечує відеозв'язок. Під час комунікації в реальному часі важливо, щоб зберігалась висока пропускна здатність, а також надійний захист інформації при передачі даних. Це стосується як переписок в чатах, де може фігурувати «чутлива» інформація, так і відеоконференцій.

Вищезгаданий додаток Zoom став особливо популярним під час пандемії за рахунок своєї простоти та зручного інтерфейсу. Через великий попит почастішили атаки з боку хакерів, які можуть зненацька перервати відеоконференцію або скинути в чат посилання, перехід на яке може бути потенційно шкідливим для комп'ютера. Для таких атак навіть придумали термін – Zoom Bombing. Британське видання The Guardian опубліковувало матеріали дослідження, де вказується про

витік персональних даних та порушення конфіденційності при використанні додатку Zoom. Цю тему підхопили й інші відомі видання і надрукували новину про злив персональних даних додатком Zoom в компанію Facebook. При чому, навіть, тих людей, які не зареєстровані в цій соцмережі.

Подібні системи комунікації здебільшого працюють на базі протоколу WebSocket або використовують технологію WebRTC.

Вищесказане визначає актуальність теми даної магістерської роботи, основний зміст якої становлять дослідження щодо захисту інформації Web-додатків на базі websocket.

Об'єкт дослідження - процес забезпечення управління ідентифікацією та доступом користувачів до інформаційної системи організації на базі рішення Gluu

Предмет дослідження - технологія управління ідентифікацією та доступом користувачів до інформаційної системи організації

Мета роботи - розробити порядок застосування технології управління ідентифікацією та доступом користувачів до інформаційної системи організації та рекомендації щодо його реалізації.

Наукові завдання:

дослідити сутність проблеми безпеки Web-додатків на базі WebSocket;

проаналізувати існуючі системи комунікації в реальному часі;

проаналізувати методи та засоби захисту Web-додатків на базі WebSocket;

розкрити порядок проектування технології забезпечення безпеки Web-додатка на базі WebSocket;

Методи дослідження - опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів, їх порівняння та проведення експерименту.

Практичне значення одержаних результатів: рекомендації щодо методів та засобів проектування захищеного Web-додатку можуть бути використані у сфері забезпечення кібербезпеки та при розробці сучасних Web-додатків на базі WebSocket.

Апробація результатів. Основні наукові результати роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції «Актуальні проблеми кібербезпеки», Навчально-науковий інститут захисту інформації, 27 жовтня 2022 р., 163-166 стр.

1 АНАЛІЗ ПРОБЛЕМИ БЕЗПЕКИ СИСТЕМ КОМУНІКАЦІЇ В РЕАЛЬНОМУ ЧАСІ

1.1. Аналіз роботи сучасних Web-додатків на базі архітектури клієнт-сервер

Система клієнт-сервер є різновидом клієнт-серверних обчислень. Клієнти та сервери можуть працювати між різними комп'ютерами, об'єднаними в мережу. Зазвичай клієнтом є персональний комп'ютер, який запитує і використовує послугу, а сервером - комп'ютер, який може бути ПК або міні-мейнфреймом, що надає послугу. Таким чином, основна архітектура системи клієнт-сервер - це клієнт, сервер і третій рівень, тобто модель, яка має два типи в своїй системі, яка називається дворівневими системами. Це між клієнтом і сервером, який обмінюється даними. Існує також трирівнева система, яка має три типи обміну та втручання матеріалів між серверами і зараз використовується багаторівнева.

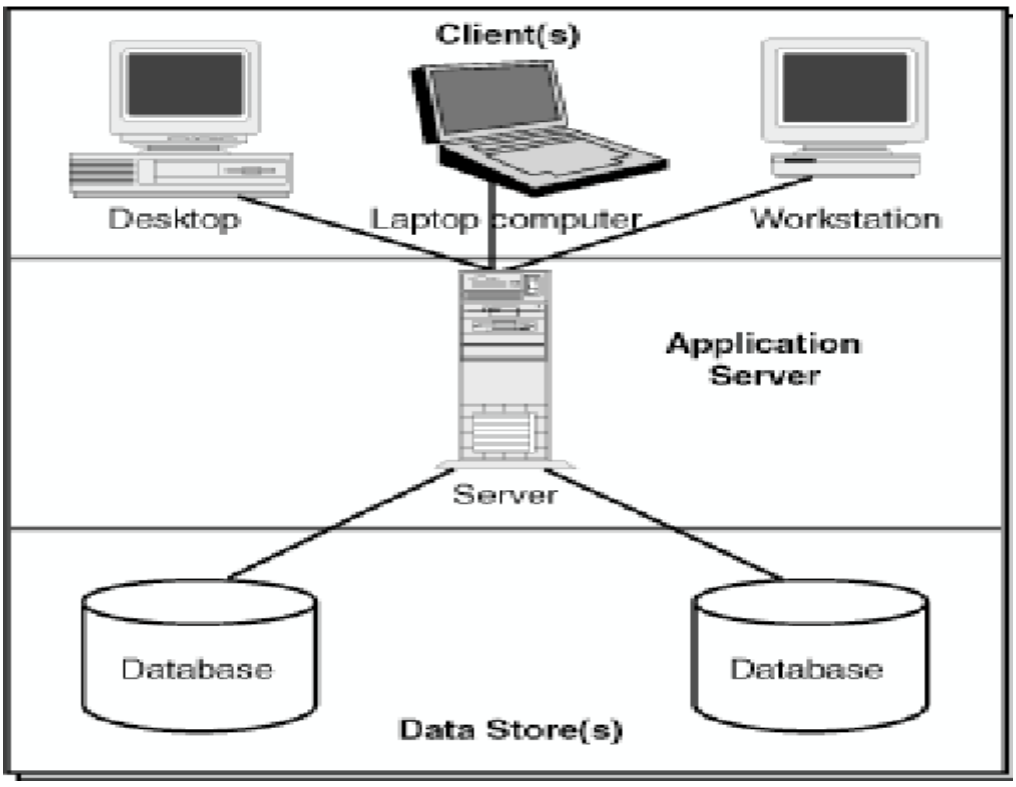


Рис. 1.1. Клієнт-серверна архітектура

Web-додаток – це додаток, побудований на архітектурі клієнт-сервер, основна частина контенту якого, зберігається на віддаленому сервері, а користувацький інтерфейс (UI) відображається в браузері у вигляді Web-сторінок. Web-додаток завантажуються на будь-якому пристрої з допомогою клієнтського застосунку – браузера та доступом до Інтернету.

Web-сайт – це група взаємопов'язаних, структурованих Web-сторінок на одному домені. Головна мета Web-сайту – показати інформацію користувачам, наприклад, блог. Головна відмінність між Web-додатком і Web-сторінкою – різні взаємодії зі сторінкою. У той час, як Web-сайти містять тексти і візуальний контент, з яким користувач не може взаємодіяти, Web додатки надають користувачеві можливість не тільки читати, а й маніпулю,вати інформацією на сторінці. Наприклад, інтернет-магазин, який дозволяє користувачам купляти товари, здійснювати пошук через каталог та оперувати даними являється Web-додатком. Зазвичай, він має меню пошуку по сайту (мапа сайту), каталог товарів, які завантажуються динамічно з серверу, форму авторизації, різні віджети для пошуку в соціальних мережах тощо, API для оплати і т.д.

Бувають декілька видів Web-додатків: CRM та ERP. CRM – система управління проектами, спрямована на автоматизацію обробки повного спектру інформації про клієнтів і товарів. Подібні рішення – це комплексний продукт, що поєднує функції баз даних, пошти, календаря, обліку фінансів та інші. У них можуть бути інтегровані, в залежності від потреб, різні модулі. CRM незамінні, коли потрібно налаштувати проектну роботу з чітким поділом за ролями і зонам відповідальності, взаємодія між відділами, роботу з клієнтами. Це актуально для банків, агентств маркетингових комунікацій, компаній-розробників IT, онлайн-магазинів товарів і послуг. Більш заточений під потреби конкретного 13 бізнесу варіант – це ERP. Це web-додатки, розроблені для автоматизації процесів управління внутрішньогосподарської діяльністю великих підприємств з розвиненою філіальною мережею, різними напрямками діяльності, складнопідрядних структурою. Включає модулі виробничого, фінансового управління, закупівлі і тд.

Серед переваг Web-додатків можна відмітити: масштабованість, доступ з будь-якого пристрою, економія. Серед недоліків те, що повна безпека не гарантована, тому Web-додаток може бути вразливим для несанкціонованого доступу.

Зазвичай, щоб створити Web-додаток, збирається команда, яка складається з дизайнерів (створюють макет майбутнього Web-додатку, як він буде виглядати), розробників (втілюють макет, реалізують його програмно, розробляють Web-додаток за його створеним макетом), тестувальники(тестують код).

Web-застосунок складається зі структури сторінок, стилів та функціоналу. Щоб створити простий Web-додаток, потрібно знати стек: HTML, CSS та мову програмування JavaScript. Це сторона фронтенду – те, що бачить в браузері користувач, все, що браузер може читати, виводити на екран або запускати. Бекенд, в свою чергу, – це те, що працює на сервері, а не в браузері. Для бекенду ви можете використовувати будь-які інструменти, доступні на вашому сервері (який, по суті, є просто комп'ютером, налаштованим для відповідей на повідомлення). Це означає, що ви можете використовувати будь-який універсальний мову програмування: Ruby, PHP, Python, Java, JavaScript / Node, bash. Це також означає, що ви можете використовувати системи управління базами даних, такі як MySQL, PostgreSQL, MongoDB, Cassandra, Redis, Memcached. На даний момент існує кілька основних архітектур, що визначають, як будуть взаємодіяти ваші бекенд і фронтенд: з допомогою серверних додатків – в цьому випадку HTTP-запити відправляються безпосередньо на сервер додатки, а сервер відповідає HTML-сторінкою. Коли сторінка завантажена в браузері, HTML визначає, що буде показано, CSS - як це буде виглядати, а JS – всякі особливі взаємодії. Інший тип архітектури використовує для зв'язку AJAX (Asynchronous JavaScript and XML). Це означає, що JavaScript, завантажений в браузері, відправляє HTTP-запит (XHR, XML HTTP Request) зсередини сторінки і (так склалося історично) отримує XML-відповідь. Зараз для відповідей також можна використовувати формат JSON. Це означає, що у вашого сервера повинна бути кінцева точка, яка відповідає на запити JSON або XML-кодом.

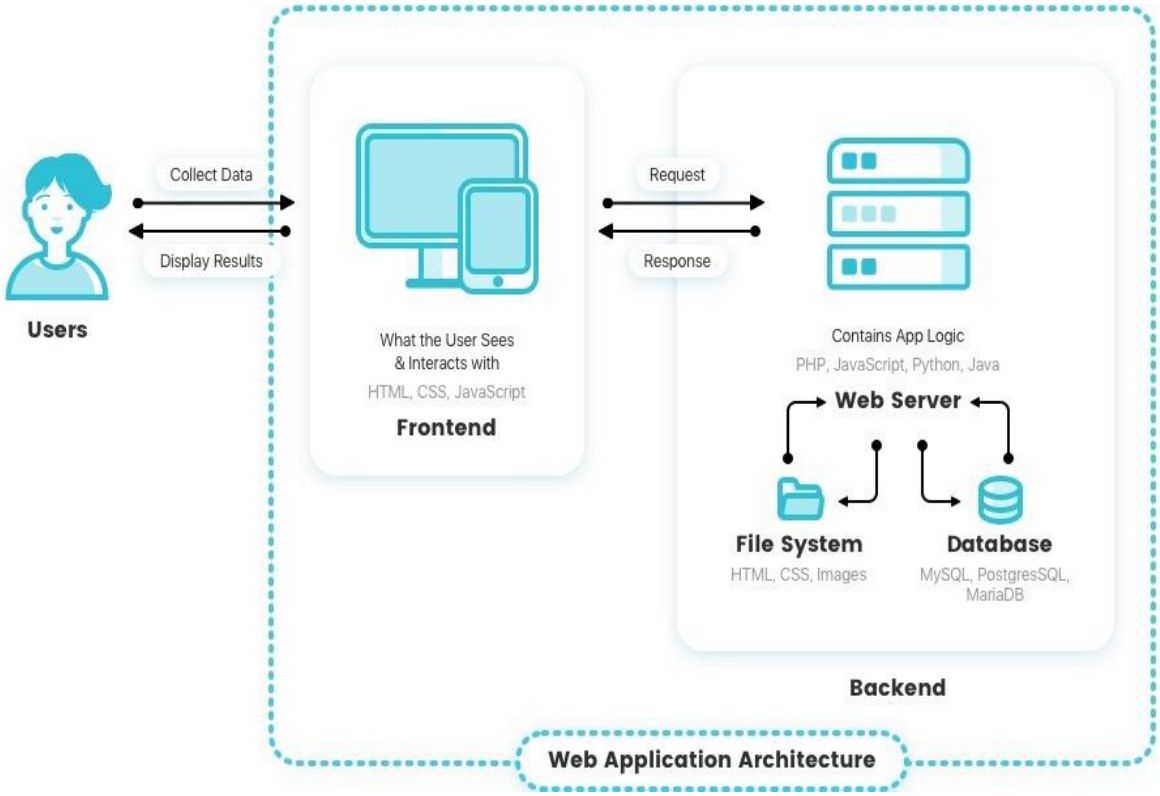


Рис. 1.2. Взаємодія між фронтендом та бекендом

Традиційний контроль за безпекою даних в клієнт-серверній архітектурі полягає у встановленні паролів для різних рівнів доступу до даних. Контроль даних зосереджений головним чином на центральному комп'ютері та доступі до нього. Зі зростанням клієнт-серверної системи потрібно кілька контрольних точок і кілька рівнів контролю.

Серед основних проблем безпеки в клієнт-серверній архітектурі варто виділити такі:

- «Зараження» серверу хробаком, вірусом або троянською програмою спровокує зараження користувачів, оскільки мережа складається з пов’язаних між собою клієнтів і серверів
- Сервер вразливий до атак типу «відмова в обслуговуванні» (DoS)
- Пакети даних можуть бути підроблені або модифіковані під час передачі
- Схильність до фішингу та атак типу «людина посередині» (MITM)

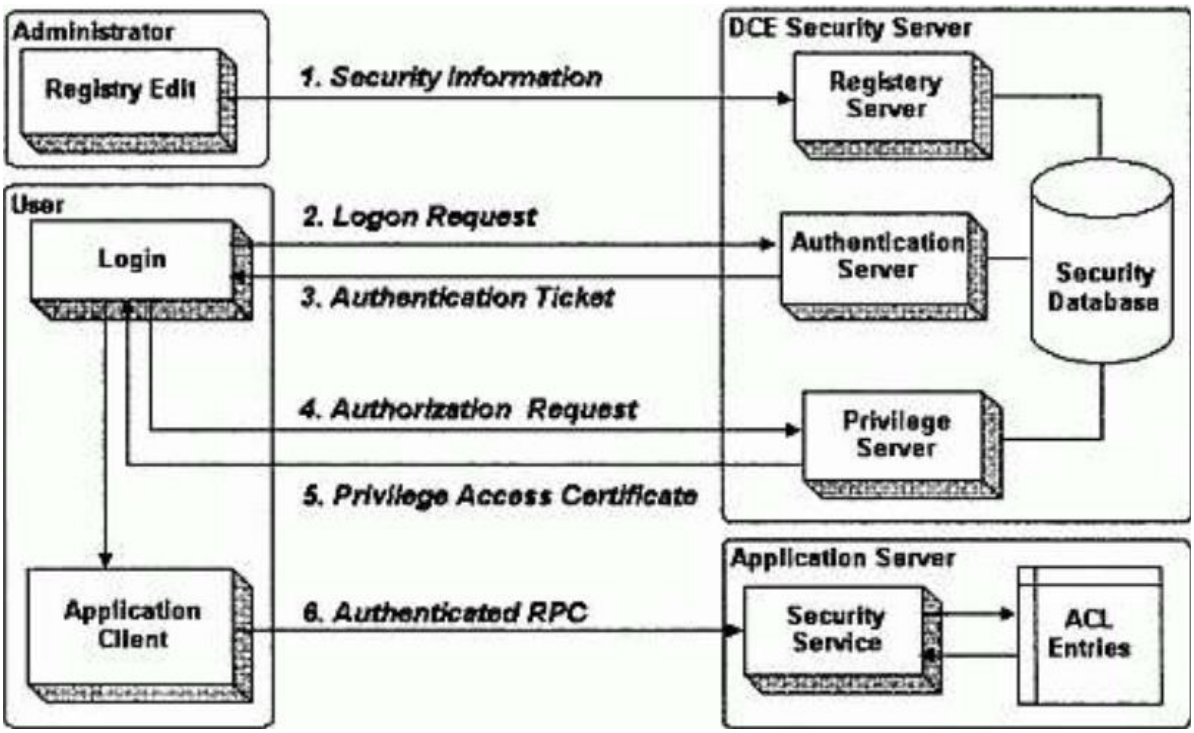


Рис. 1.3. Безпека клієнт-серверної архітектури

Основна особливість клієнт-серверної архітектури в розподілених обов'язках між учасниками. Клієнт може самостійно повідомляти про зміну необхідних даних, просто регулярно надсилаючи запити на їх отримання. При цьому дані можуть не змінюватись. Цей тип комунікації називається «технологією опитування» (pull technology). Даний підхід є не оптимальним в системах комунікації в реальному часі, оскільки з'являються великі затримки між отриманням даних, так як сервер надсилає дані не тоді, коли вони з'явились, а, коли прийшов відповідний запит.

Досконалішим рішенням цих проблем є використання веб-сокетів, за рахунок створення двонапрямленного каналу зв'язку.

1.2. Використання технологій під час проектування захищених real-time Web-додатків

Функціональність в режимі реального часу є критичною вимогою у сучасних веб-технологіях. Подібні технології використовуються у багатьох ресурсах в мережі, таких, як: чати для спілкування, відеоконференції, пошта, соціальні мережі

тощо.

У всіх нас є обліковий запис електронної пошти. Після того, як ви увійшли в систему, ви можете побачити список електронних листів, які ви отримали до цього часу. Як тільки на нашу поштову скриньку надходить новий електронний лист, список автоматично оновлюється. Нам не потрібно кожного разу перезавантажуватись, щоб побачити оновлення. Поштовий сервер ідентифікує новий лист і автоматично оновлює список. Це реалізація функції в режимі реального часу реалізована у вашому поштовому клієнті.

У розділі коментарів будь-якої соціальної мережі, якщо хтось додає коментар, він швидко з'являється в розділі. Не потрібно щоразу перезавантажувати сторінку, щоб побачити нові коментарі. Це ще один приклад використання функціональності реального часу в повсякденних додатках.

Якщо ви працювали з такими веб-документами, як Google Docs, Office 360, One, зверніть увагу, що зміни, які вносяться іншими користувачами, негайно відображаються в наших документах. Це відбувається в режимі реального часу.

Історично склалося так, що створення веб-додатків, які потребують двостороннього зв'язку між клієнтом і сервером (наприклад, обмін миттєвими повідомленнями та ігрові додатки) вимагало зловживання HTTP для опитування сервера для отримання оновлень при одночасній відправці вихідних повідомлень у вигляді окремих HTTP-викликів [RFC6202].

Це призводить до різноманітних проблем:

- Сервер змушений використовувати ряд різних базових TCP
- Дротовий протокол має високі накладні витрати, оскільки кожне повідомлення між клієнтом і сервером має HTTP-заголовок.
- Скрипт на стороні клієнта змушений підтримувати відображення від вихідних з'єднань до вхідних з'єднань для відстеження відповідей.

Рішення, які використовуються в real-time Web-додатках:

- Ajax - це не те, що безпосередньо пов'язано з Web-додатками в реальному часі. Ajax - це технологія відправки запитів з браузера на сервер за допомогою JavaScript. І такі технології, як polling, використовують для відправки запитів на сервер за допомогою ajax. А дані відповіді можуть бути додані до HTML-сторінки без перезавантаження всієї сторінки. Варіант використання Ajax - оновлення DOM без перезавантаження всієї сторінки. Або, можна сказати, відправка запитів у фоновому режимі.
- Polling - це спосіб створення веб-функціональності в режимі реального часу. Що він робить, так це періодично запитує сервер, чи є оновлення. На основі часового інтервалу. Сервер відповідає "Не знайдено", якщо оновлення немає, або з оновленим вмістом, якщо воно є. Це не є ефективним методом. Тому що періодично на сервер відправляється величезна кількість Get запитів.
- Long polling - браузер відправляє початковий запит, але сервер не завершує запит в той же час. Він тримає отриманий запит відкритим до тих пір, поки на сервері не з'явиться оновлення. Як тільки оновлення з'являється, сервер відправляє відповідь з оновленими даними. Якщо протягом деякого часу оновлення не з'являється, запит отримує таймаут.

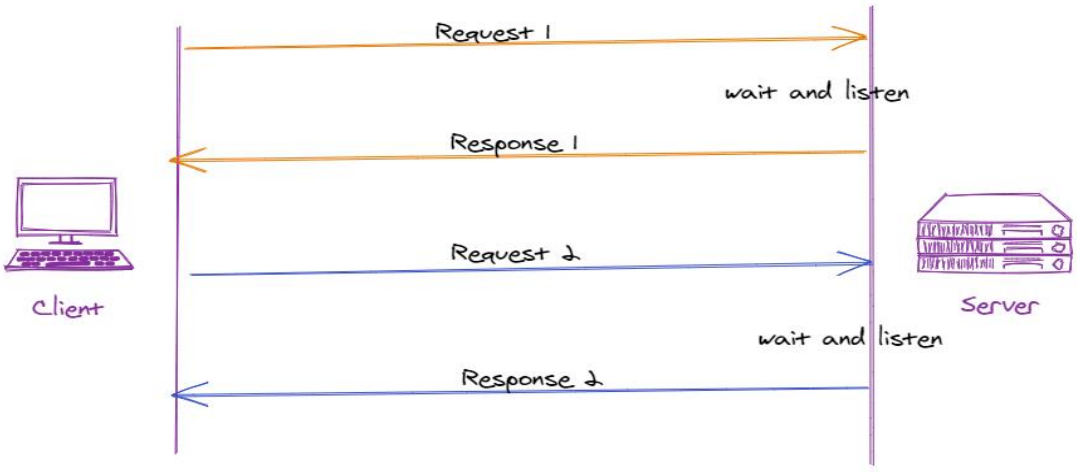


Рис. 1.4. Long Polling

- Server-Sent Events - це функція HTML5. Тут сервер створює HTTP-з'єднання з браузером. І браузер активно прослуховує з'єднання у вигляді потоку до тих пір, поки з'єднання не буде закрито. А клієнт має джерело подій, яке має метод для обробки вхідних повідомлень від прослуховування з'єднання. Він також має методи для обробки помилок, які можуть виникнути.

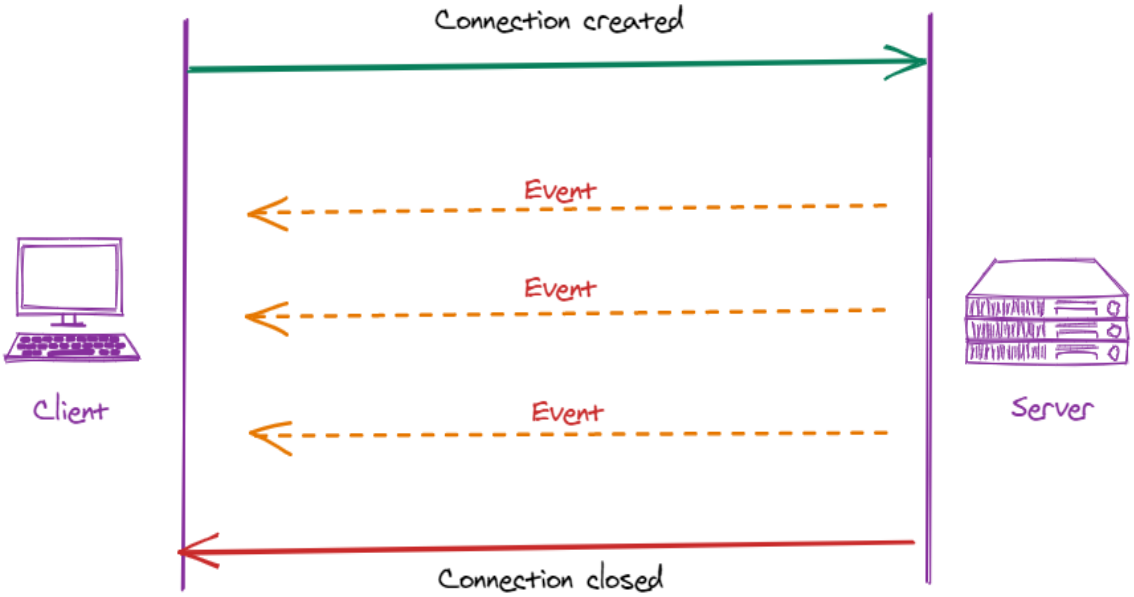


Рис. 1.5. Server-Sent Events

Більш простим рішенням було б використання одного TCP-з'єднання для трафіку в обох напрямках. Це те, що забезпечує протокол WebSocket. У поєднанні з WebSocket API [WSAPI] він забезпечує альтернативу HTTP-опитуванню для двостороннього зв'язку з веб-сторінки до віддаленого сервера.

- WebSocket - протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу. Протокол повністю асинхронний та симетричний. Він застосовується для організації чатів, онлайн-табло тощо і створює постійне з'єднання між клієнтом та сервером, яке обидві сторони можуть використовувати для надсилання даних.

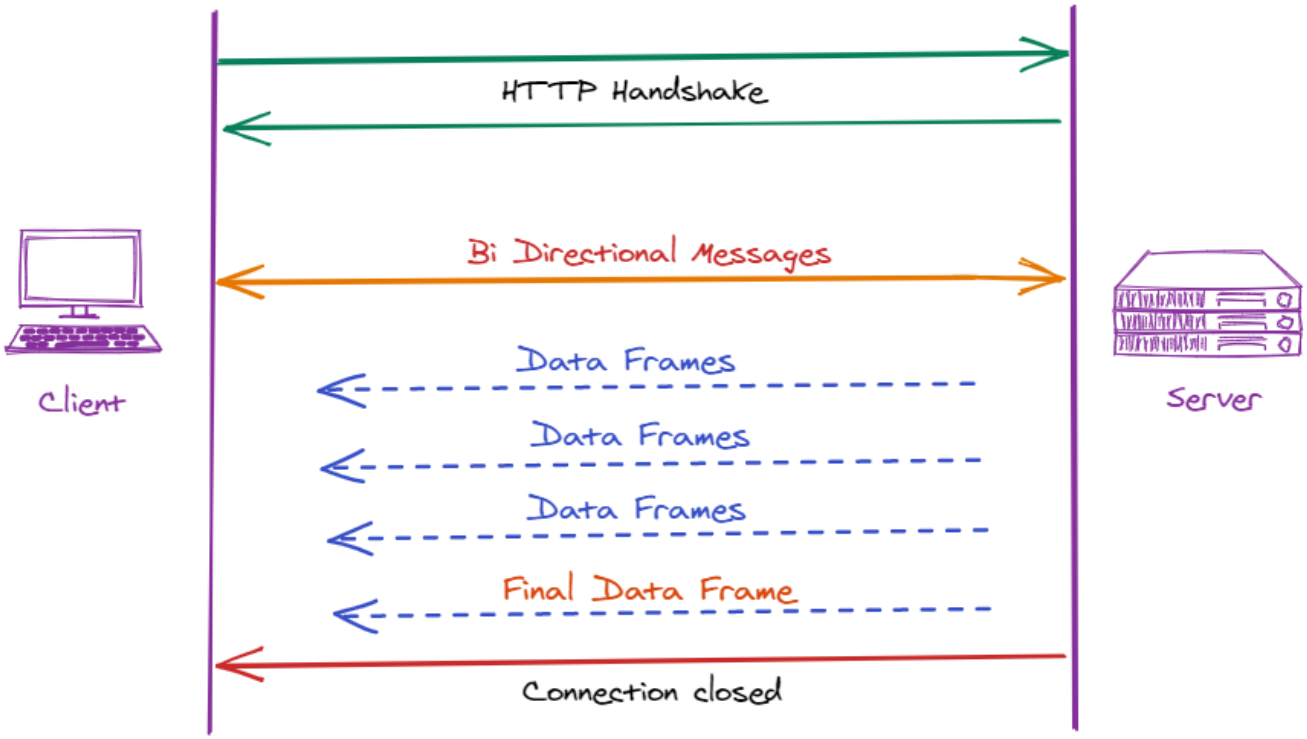


Рис.1.6. WebSocket

WebSocket можуть створювати до 50 з'єднань, уникаючи обмежень на максимум 6 з'єднань, які мають події, що надсилаються сервером. Вони можуть передавати повідомлення в текстовому або двійковому форматі. Таким чином, їх можна використовувати для передачі текстових повідомлень, відео та аудіо.

WebSocket дозволяють як серверу, так і клієнту передавати повідомлення в будь-який час без будь-якого відношення до попереднього запиту. Протокол є серйозним розширенням HTTP, яке дає змогу веб-додаткам підтримувати багатокористувацьку взаємодію в режимі реального часу, що є хорошою перевагою в порівнянні з протоколом HTTP не тільки в плані функціональності, а й з боку забезпечення належного рівня безпеки переданих даних. Однією з помітних переваг використання сокетів вирішує кілька проблем з HTTP:

Двонаправлений протокол - будь-який клієнт/сервер може надіслати повідомлення іншій стороні (у HTTP запит завжди ініціюється клієнтом, а відповідь обробляється сервером, що робить HTTP однонаправленим протоколом)

Повнодуплексний зв’язок - клієнт і сервер можуть одночасно спілкуватись один з одним.

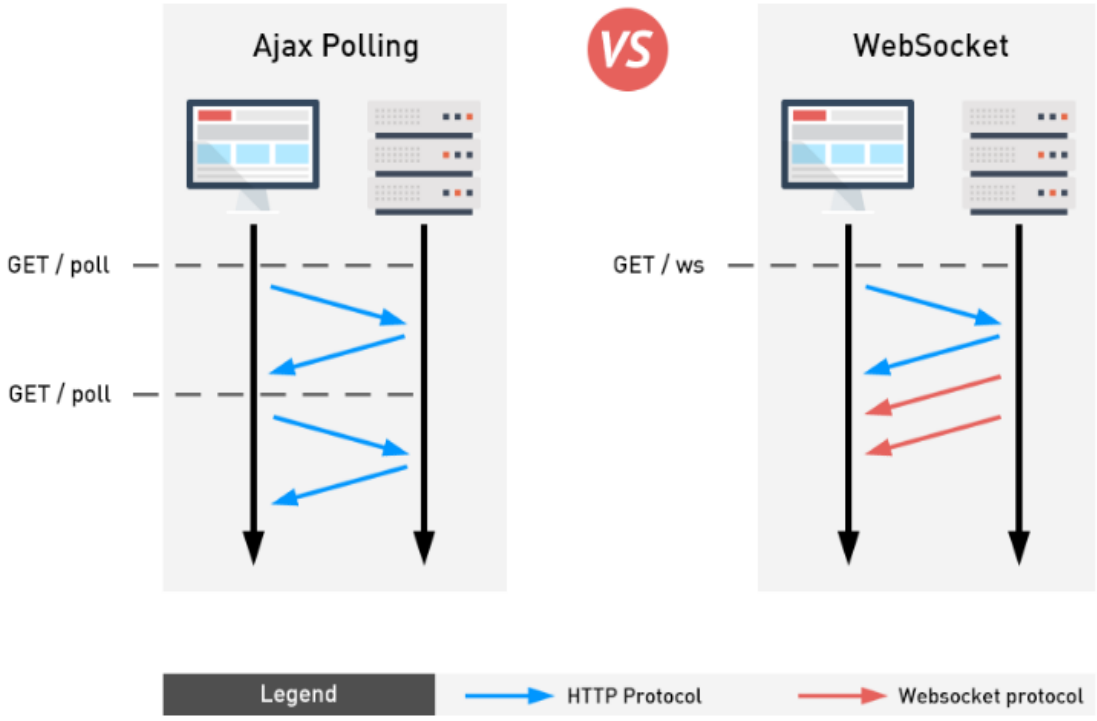


Рис.1.7. З’єднання через WebSocket

Єдине TCP-з’єднання - на початку клієнт і сервер спілкуються через TCP-з’єднання, потім протягом усього життєвого циклу - через з’єднання WebSocket.

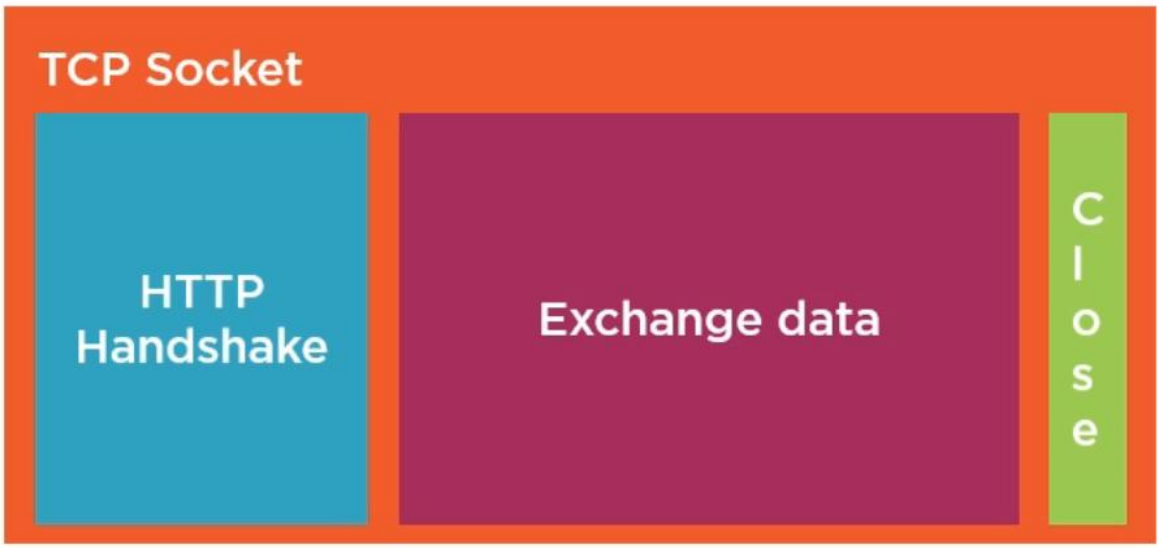


Рис.1.8. TCP-сокет

Все, що відбувається під час з’єднання, відбувається всередині TCP-сокета. Спочатку на сервер відправляється звичайний HTTP-запит, щоб оновити доступний TCP-сокет на веб-сокет. Вхідні повідомлення можуть проходити через

створений сокет до тих пір, поки він не буде закритий.

Для моніторингу трафіку WebSocket зручно використовувати інструменти розробника, доступні, наприклад, в Chrome.

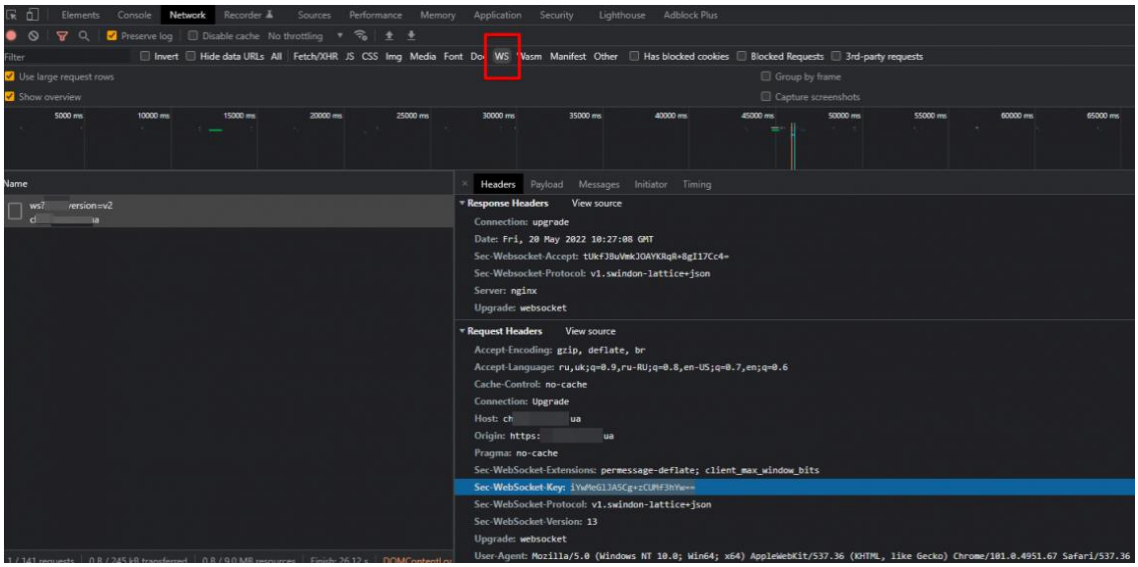


Рис.1.9. WebSocket в інструментах розробника браузера

Дані протоколу WebSocket передаються як послідовність кадрів. Фрейм має заголовок, у якому міститься така інформація:

- чи фрагментоване повідомлення;
- тип даних — all code;
- чи піддавалися повідомлення маскування - прапор маски;
- розмір даних;
- ключ маски (32 біти);
- інші керуючі дані (ping, pong...).

Усі повідомлення, надіслані клієнтом, мають шифруватися. Шифрування проводиться звичайним XOR із ключем маски. Клієнт повинен змінювати ключ для кожного переданого кадру. Сервер не повинен шифрувати повідомлення. Шифрування повідомлень, що передаються, некриптостійке, щоб забезпечити конфіденційність, для WebSocket слід використовувати протокол TLS і схему WSS.

Також для створення систем комунікації в реальному часі, окрім протоколу WebSocket, можна використовувати WebRTC.

WebRTC (Web Real Time Communication) – технологія, що дозволяє пряме з’єднання між різними клієнтськими браузерами, замість того, щоб проходити через сервер.

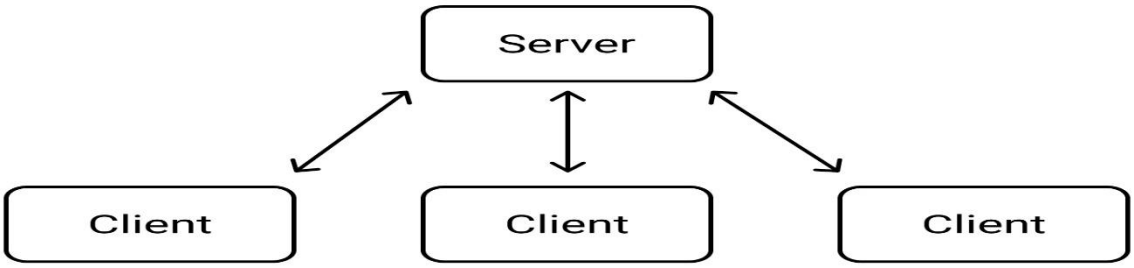


Рис.1.10. Комунікація за допомогою протоколу WebSocket

За допомогою WebRTC ви можете додати можливості зв'язку в режимі реального часу до вашого додатку, який працює на основі відкритого стандарту. Він підтримує передачу відео, голосу та типових даних між одноранговими пристроями, дозволяючи розробникам створювати потужні рішення для голосового та відеозв'язку. Технологія доступна у всіх сучасних браузерах, а також у нативних клієнтах для всіх основних платформ. Технології, що лежать в основі WebRTC, реалізовані як відкритий веб-стандарт і доступні як звичайні JavaScript API у всіх основних браузерах. Для нативних клієнтів, таких як додатки для Android та iOS, доступна бібліотека, яка забезпечує таку ж функціональність. Проєкт WebRTC має відкритий вихідний код і підтримується Apple, Google, Microsoft, Mozilla та серед інших.

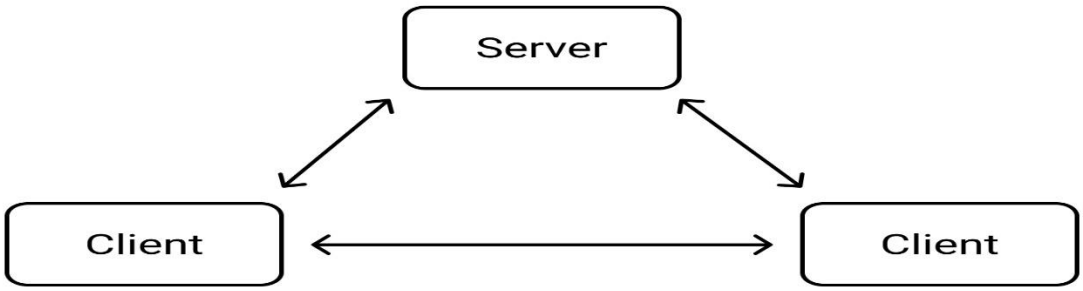


Рис.1.11. Комунікація за допомогою технології WebRTC

Під час з’єднання за допомогою WebRTC існує пряме з'єднання між усіма унікальними парами сервер і клієнт. Клієнтські браузери можуть безпосередньо спілкуватися один з одним, замість того, щоб проходити через сервер. Замість TCP-з’єднання під час роботи з WebSocket, WebRTC відкриває з’єднання поверх протоколу UDP, що забезпечує швидку передачу даних. Роль сервера тут - допомагти координувати з'єднання. Наприклад, звідки клієнт А знає, що існує клієнт Б, і навпаки.

	WebSockets	**WebRTC**
Definition	WebSocket is a computer communications protocol, which provides communication channels over a single TCP connection	WebRTC is a open-soruce project that provides browsers and mobile apps with Real-Time communications capabilities via simple APIs
Latency	Low Latency	Near Real-time
Data Transfer	TCP	UDP
Uses	- Chat rooms, Location-based Apps, File Transfer, etc	- Streaming, Video calls, Multiplayer games, etc

Рис.1.12. Порівняльна характеристика між WebSocket та WebRTC

Загалом заходи безпеки з WebRTC можна розділити на три основні групи: Клієнт, Сервер і Мережевий трафік.

- Безпека сервера - оскільки WebRTC розроблено як одноранговий інтерфейс, обов'язком сервера є посередництво між клієнтами. Таким чином, ролі, що виконуються сервером реалізуються за допомогою програмних компонентів до WebRTC, таких як STURN (Session Traversal Utilities for NAT) та IdP (Identity Provider). Ніяких додаткових призначених заходів безпеки не створюється.
- Клієнт(контроль доступу) - оскільки WebRTC в першу чергу призначений для полегшення P2P аудіо та відео чатів P2P, браузер, що сприяє цьому, повинен мати доступ до мікрофона та відеокамери. Для цього потрібна згода користувача.
- Мережевий трафік

1.3. Аналіз існуючих систем комунікації в реальному часі

Існує безліч систем комунікації в реальному часі. Більшість з них використовуються для спілкування: особистого чи робочого, соціальних мереж, форумів, стрічок новин тощо. Серед найвідоміших слід відмітити такі, як: Skype, Zoom, Google Meet, Slack, Microsoft Teams, Telegram, Viber та багато інших.

Під час пандемії Covid-19 стрімко зріс попит на Web-додатки онлайн спілкування, бо люди вимушені були працювати віддалено. Завдяки багатьом сервісам комунікації в реальному часі, людям вдалось швидко адаптуватись до дистанційної роботи. Паралельно із великим попитом з боку звичайних споживачів зріс інтерес до подібних сервісів і з боку зловмисників.

Оскільки пандемія застала нас зненацька, ми були швидко змушені працювати онлайн та по телефону. Zoom був популярним рішенням на той час, пропонуючи безкоштовну альтернативу більш складним бізнес-рішенням, таким як

Google Meets. Природно, що Zoom швидко став популярним рішенням, оскільки був безкоштовним і простим у використанні. Однак його вибух як універсального рішення також виявив багато недоліків. Zoom не був підготовлений.

Першою проблемою був сам інсталятор. Інсталятор Zoom був надзвичайно схематичним, і хакери легко використовували доступ на системному рівні, щоб встановити бекдори на комп'ютерах користувачів. Ці інсталятори встановлювали Zoom як зазвичай, але також відкривали довільний порт, який надавав хакерам постійний адміністративний доступ до комп'ютерів користувачів. Вони могли бачити все і чути все.

Одна з найбільших проблем, яка викликала багато заголовків, пов'язаних з питаннями безпеки Zoom. Історично склалося так, що будь-хто, маючи посилання на зустріч у Zoom або ідентифікатор, міг увійти до кімнати для нарад і підслуховувати всю розмову. Були випадки, коли непрохані гості переривали цифрові зустрічі, демонструючи неприйнятний контент.

Під час "Zoom-bombing" організатори практично не контролюють ситуацію. Вони не можуть вирішити, хто може увійти в нараду, призупинити діяльність користувача або поставити нараду на паузу.

Якщо порівнювати Zoom та Google Meet, якими вони є сьогодні, то на щастя, компанія Google зробила своє рішення для бізнесу (Google Meet, раніше Hangouts) повністю безкоштовним для всіх, хто має обліковий запис електронної пошти Google. Google, як набагато досвідченіший і надійніший постачальник, намагався задовольнити попит, який Zoom не зміг задовольнити.

Zoom, безумовно, посилив свою безпеку, як і очікувалося. Найбільшою проблемою, здається, є той факт, що для повної функціональності потрібно буде встановити клієнт/додаток, і саме цей клієнт, здається, є джерелом багатьох проблем з безпекою. Google Meet, з іншого боку, надає все через браузер. Немає необхідності в патчах, і немає необхідності в установці, але вони надають цю можливість незалежно від цього.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ЗАХИСТУ СИСТЕМ КОМУНІКАЦІЇ В РЕАЛЬНОМУ ЧАСІ

2.1. Безпека real-time Web-додатків

В звіті Global Internet Security Threat Report (ISTR) компанія Symantec вказує, що зловмисники при отриманні несанкціонованого доступу зазвичай використовують вразливості Web-додатків, або експлуатують деякі вразливості операційної системи серверів, на якій працюють ці додатки.

У розроблюваних інженерами застосунків є кілька точок, де найчастіше ховаються вразливості:

Ін'єкції - SQL, LDAP, XPath;

Слабка криптографія;

Небезпечна передача інформації;

Некоректна обробка помилок;

Розкриття інформації;

Cross-Site Scripting (XSS);

Cross Site Request Forgery (CSRF);

Помилки в контролі доступів;

Некоректна автентифікація та управління сеансами.

При розробці будь-якого продукту головною вимогою є дотримання усіх етапів безпечного циклу створення додатків.

З приводу масштабних зламів Web-додатків, розробники та ентузіасти в області інформаційної безпеки створили некомерційний проект OWASP – Open Web Application Security Project. Це відкритий проект забезпечення безпеки Web-додатків. Основною причиною більшості взломів в Web-додатках є написаний розробниками програмний код. Розробники можуть допускати помилки при написанні коду або не усвідомлювати всю важливість використання прийомів безпечного програмування – все це призводить до появи вразливостей в додатках. Вразливість, або слабке місце Web-додатку, яке може призвести до його

експлуатації в зловмисних цілях залишається одним з найбільш поширених недоліків забезпечення захисті інформації. Крім того, серед інших проблем, які часто зустрічаються, є низька обізнаність розробників у питаннях інформаційної 24 безпеки, слабкі паролі, недолуга політика безпеки в компанії, недоліки в процесах управління, використання застарілого програмного забезпечення, використання небезпечних конфігурацій тощо. Проблема захищеності Web-додатків часто виникає, що при їх розробці часто не враховується питання, пов'язане з безпекою цих систем від внутрішніх та зовнішніх загроз. Недооцінка ризиків інформаційної безпеки призводить до того, що приходиться виправляти вразливості вже в готовому Web-додатку, що стає дорожче для компанії, ніж при його розробці. Ризики потрібно контролювати. Для цього є спеціальні метрики матриці ризиків CVSS 3.0.

CVSS складається з трьох метричних груп: Base, Temporal і Environmental. Базова метрика описує рівень складності експлуатації вразливості і потенційний збиток. Тимчасова метрика вносить в загальну оцінку поправку на повноту інформації про вразливість та зрілість експлуатуючого коду (якщо він є). За допомогою контекстних метрик спеціалісти з інформаційної безпеки можуть вносити результуючу оцінку, відповідно характеристикам інформаційної середи. З новою версією CVSSv3, додаються нові поняття: - вразливий компонент – компонент інформаційної середи, схильний до експлуатації - компонент, який атакують - компонент, конфіденційність, цілісність або доступність якого може постраждати при успішній реалізації атаки.

Рейтинг вразливостей Таблиця 2.1

Немає	0.0
Низький	0.1 - 3.9
Середній	4.0 - 6.9
Високий	7.0 - 8.9
Критичний	9.0 - 10.0

Відповідно від оцінки вразливості, їй виставляється рейтинг. Таким чином система CVSS надає можливість оцінити критичність вразливості Web-додатку. При проектуванні real-time Web-додатку слід використовувати оцінку CVSS3.0. Відповідно до даних WASC (Web Application Security Consortium), більше 13% сайтів можуть бути скомпрометовані повністю автоматично, 80-96% схильні до загроз у високому ступені, 86% – середній ступінь вразливості, 37% –низький. Згідно з методологією NIST Special Publications 800-115 Technical Guide to Information Security Testing and Assessment, суб'єкт надання послуг повинен розробити політику оцінки інформаційної безпеки, що повинна визначати вимоги до оцінки безпеки та притягувати до відповідальності осіб, відповідальних за забезпечення відповідності оцінок вимогам. В розділі документу "Техніки оцінки вразливостей" описується важливість тестів на проникнення. Відповідно документації, Pentest, крім його основних функцій, можна застосовувати для визначення: - здатність системи переносити існуючі моделі атак - рівень складності, який потрібно подолати зловмиснику в процесі атаки - здатність системи до самозахисту - додаткових заходів протидії, які могли б послабити загрози на адресу системи. При розробці додатку треба дотримуватись певних вимог, сталих практик та всіх етапів безпечного життєвого циклу розробки.

Безпечний життєвий цикл розробки, або S-SDLC, являє собою певні практики безпеки на кожному етапі розробки додатків. Дотримання S-SDLC допомагає нашим фахівцям будувати якісні та безпечні процеси розробки. Такий підхід допомагає передбачити вразливості, які можуть виникнути на всіх етапах розробки, і захистити від них додаток.

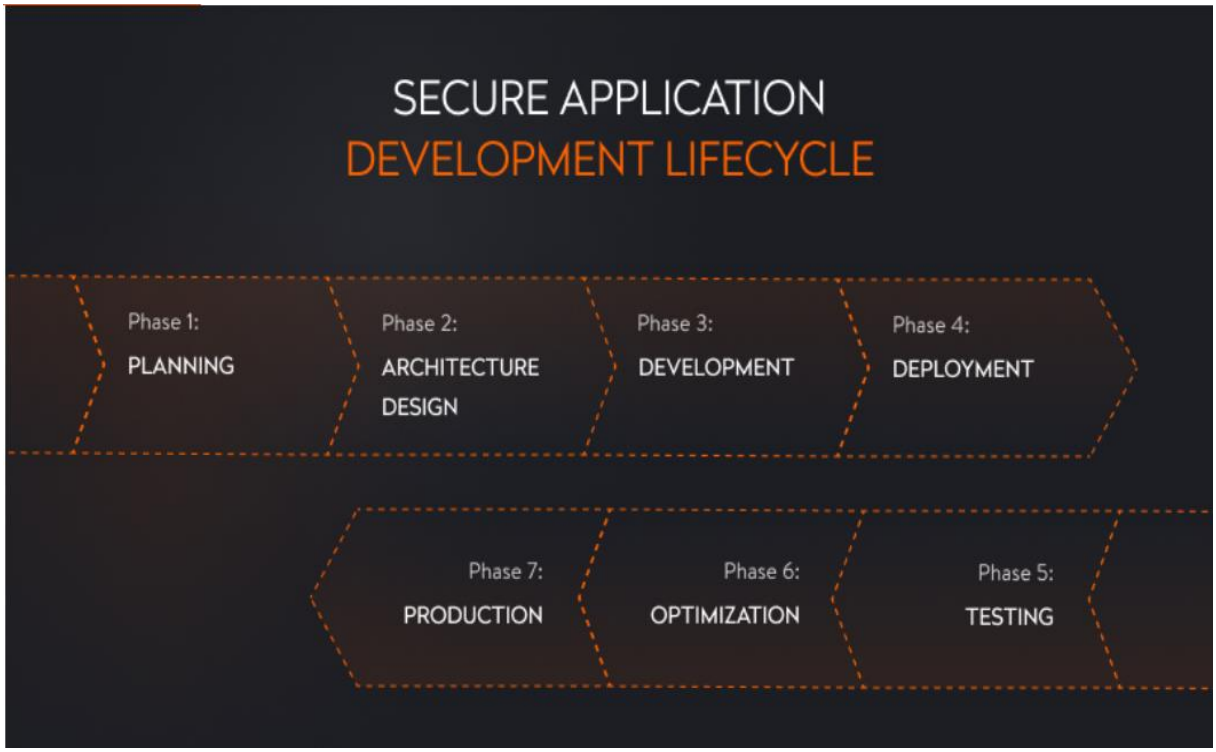


Рис.2.1. S-SDLC

Безпека додатків з'явилася, розвинулася, дозріла і прийнята на етапах програмування і тестування життєвого циклу додатків, а не на етапі їх експлуатації. Технології захисту додатків на етапі експлуатації прийняті в меншій мірі, та й то з деякими застереженнями. Це можна пояснити. Впровадження технологій оцінки додатків/виявлення вразливостей є менш ризикованим, ніж впровадження технологій захисту додатків.

Такі технології, як статичне тестування безпеки додатків (SAST), динамічне тестування безпеки додатків (DAST) та аналіз складу програмного забезпечення (SCA) слугують гарним прикладом технологій оцінювання. Вони аналізують джерело додатку, байт або двійковий код під час оцінки або аналізують наявність сторонніх компонентів у додатку. На основі цього аналізу фахівці з розробки та безпеки додатків можуть вжити заходів щодо усунення недоліків, таких як виправлення вразливостей у коді програми. Ці дії впливають лише на невиробничі додатки, таким чином, як правило, створюючи незначний ризик для зручності використання додатків або взагалі не створюючи такого ризику.

В процесі розробки Web-додатку необхідно бути впевненим, що код працює

коректно і на виході буде отримано результат, який очікується. Щоб в цьому бути впевненим, потрібно проводити тестування функцій в коді. Для тестування JavaScript-компонентів підійде бібліотека Jest. Функціонал даної бібліотеки дуже широкий і надає можливість перевірити багато факторів: який тип даних приходить в компоненті, чи працює компонента так, як від неї вимагається тощо.

Наприклад, можна провести простий тест на перевірку чи видає помилку та чи інша функція:

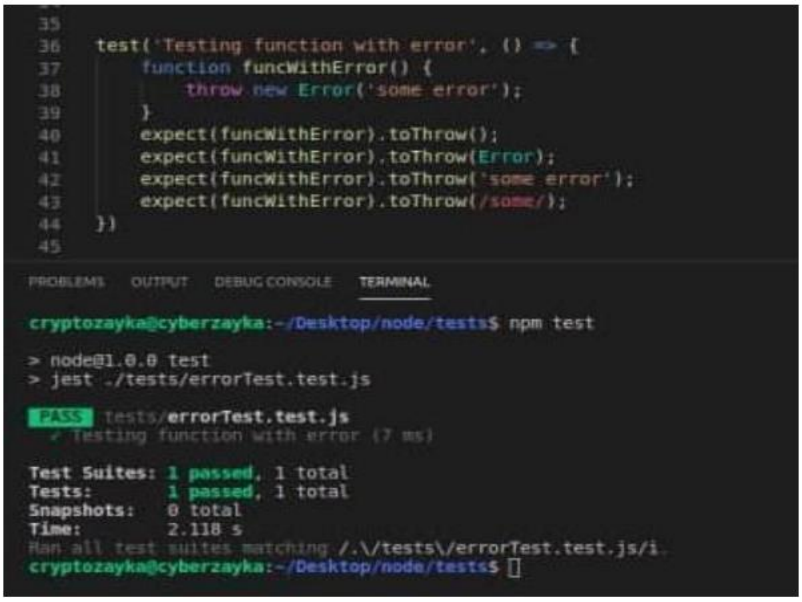


Рис.2.2. Тестування

Метод toThrow() використовується, коли потрібно перевірити виключення або просто наявність помилки. Тест пройдено успішно, отже, функція funcWithError() повертає помилку. Таким чином можна тестувати більш складніші компоненти коду. При масштабуванні Web-додатку виникає шанс появи вразливостей в коді, тому тестування - кращий метод перевірки на наявність помилок у коді. При перевірці стану захищеності коду підійде тестування методом SAST (Static 51 Application Security Testing). Даний тип тестування дозволяє розробникам знаходити вразливості безпеки у вихідному коді застосунку вже на ранніх етапах життєвого циклу розробки ПЗ. SAST також забезпечує відповідність стандартам написання коду без фактичного виконання базового коду. Серед

готових рішень, які пропонують SAST-тестування, консорціум OWASP рекомендує: плагін eslint-plugin-security, NodeJsScan та CodeWarrior. Останній, після завантаження його з репозиторію та запуску виконуваного файлу, запустить локальний сервер, на якому локально можна будет протестувати Web-додаток на наявність відомих вразливостей.

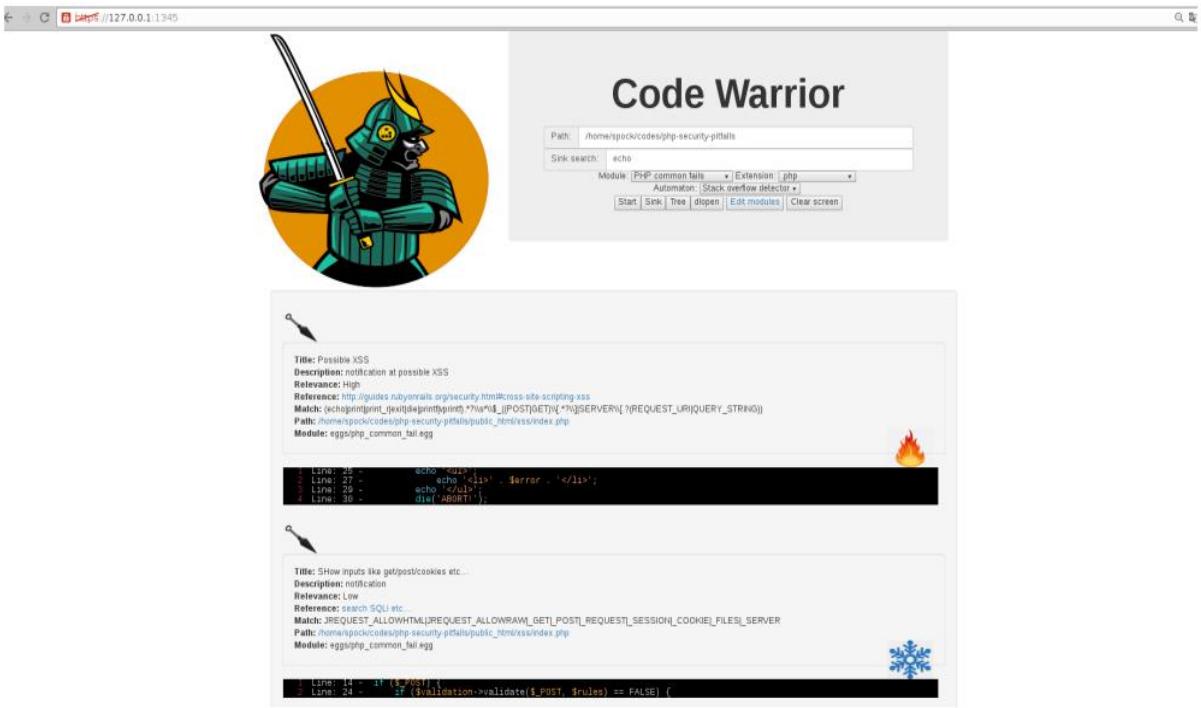


Рис.2.3. SAST-тестування утилітою CodeWarrior

Для повної впевненості в безпеці даних Web-додатку недостатньо захистити тільки застосунок. Необхідно детально вивчити середу роботи, сервер, на якому хоститься додаток, щоб бути впевненим, що ніщо не стане точкою входу для атаки, яка спроможна буде потім скомпрометувати Web-додаток і його дані. Для подібного роду аналізу використовуються сканери вразливостей. Сканер вразливостей – це програмне або апаратне комплексне рішення для сканування 52 інформаційної структури в реальному часі. Сканер використовується для виявлення вразливостей та помилок в мережі, операційній системі, базах даних, Web-додатках тощо. Головна задача – оцінювати безпеку, виявляти вразливості та

виводити докладний звіт стосовно дослідження. Існує багато сканерів, серед яких варто відзначити: Nessus Tenable, OpenVas та Qualys, які користуються зараз популярністю та мають широкий та зручний функціонал.

Nessus Tenable Nessus являється одним з багатьох сканерів вразливостей, які використовуються під час їх оцінки та тестування на проникнення, включаючи вредоносні атаки. Програма виконує сканування, використовуючи плагіни – це окремі фрагменти коду, які Nessus використовує для проведення окремих типів сканування. Nessus дає можливість налаштувати сканування на основі різноманітних шаблонів сканування і політики.

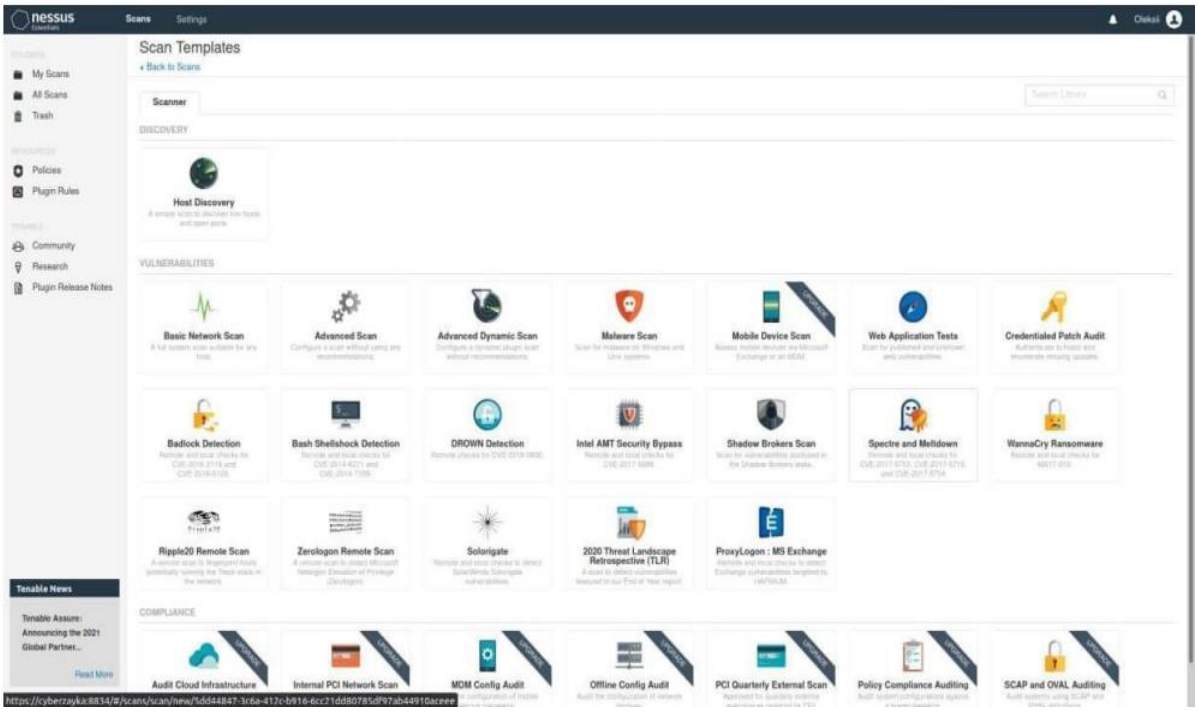


Рис.2.4. Головна сторінка Nessus

Впровадження технологій захисту, які діють на етапі виробництва, є набагато більш ризикованою пропозицією, оскільки це може порушити виконання виробничих додатків, спричинити збій або несправність. Таким чином, впровадження цих технологій відставало від технологій виявлення/тестування, і ринок мирився з цим до недавнього часу, поки ситуація в сфері безпеки не змінилася.

За останні кілька років трансформувався вектор і характер атак. Прикладний рівень все частіше стає основною мішенню атак. Останні атаки спрямовані на використання слабких місць в додатках для отримання несанкціонованого доступу до грошей, даних і, можливо, контролю над (наприклад, електромережами або системами водопостачання). Ці атаки також стали більш цілеспрямованими і часто підтримуються урядами або терористичними організаціями.

Зміна ситуації б'є на сполох і викликає попит на технології з нульовою затримкою, здатні в режимі реального часу виявляти атаки, спрямовані на працюючі додатки, і захищати від них.

Протягом багатьох років IT-індустрія замінювала спеціалізовані технології захисту додатків мережевими технологіями безпеки, в основному брандмауерами веб-додатків (WAF). Проте, вони не отримали широкого розповсюдження і не є ефективними для захисту від атак на додатки. Ці технології є аналізаторами мережевого трафіку і як такі розглядають додатки як чорні ящики. Вони не мають розуміння логічних потоків додатків, доступу до баз даних, обробки даних або конфігурацій. Через ці недоліки вони не можуть відрізнити (з необхідним ступенем впевненості) атаку від легітимного доступу, що робить їх неефективними для справжнього захисту в режимі реального часу. Покладатися лише на них часто занадто ризиковано.

Тільки RASP призначений для роботи на стадії виробництва, з додатком, що знаходиться у виробництві. RASP стає невід'ємною частиною середовища виконання, тому він пропонує унікально глибоке розуміння логіки роботи додатків, а також потоку даних і, таким чином, з унікальною точністю може виявляти атаки і захищати від них. Завдяки цій новій технології ми побачимо трансформацію в тому, як підприємства захищають свої додатки.

Runtime Application Self Protection (RASP) - це рішення безпеки, призначене для забезпечення персоналізованого захисту додатків. Воно використовує переваги розуміння внутрішніх даних і стану додатку, що дозволяє йому виявляти загрози під час виконання, які в іншому випадку могли б бути пропущені іншими рішеннями безпеки.

RASP охоплює і захищає конкретний додаток, а не загальне захисне рішення на рівні мережі або кінцевих точок. Таке більш цілеспрямоване розгортання дозволяє RASP контролювати входи, виходи і внутрішній стан програми, яку він захищає. Розгортаючи RASP, розробники можуть виявляти вразливості в своїх додатках. Крім того, рішення RASP може блокувати спроби використання існуючих вразливостей у розгорнутих додатках.

Сфокусований моніторинг RASP дозволяє виявляти широкий спектр загроз, включаючи атаки "нульового дня". Оскільки RASP має уявлення про внутрішню структуру програми, він може виявляти зміни в поведінці, які можуть бути спричинені новою атакою. Це дозволяє йому реагувати навіть на атаки "нульового дня" на основі того, як вони впливають на цільовий додаток.

RASP відрізняється від інших рішень з кібербезпеки рівнем фокусування на одному додатку. Це дозволяє йому забезпечити низку переваг у сфері безпеки:

Контекстна обізнаність: Коли рішення RASP виявляє потенційну загрозу, воно має додаткову контекстну інформацію про поточний стан програми і про те, які дані і код піддаються впливу. Цей контекст може бути безцінним для розслідування, сортування та усунення потенційних вразливостей, оскільки він вказує, де саме в коді знаходиться вразливість і як саме вона може бути використана.

Видимість атак на рівні додатків: RASP має глибоку видимість на рівні додатків, оскільки він інтегрований з конкретним додатком. Ця видимість, розуміння та знання прикладного рівня можуть допомогти виявити ширший спектр потенційних атак та вразливостей.

Захист з нульового дня: Хоча RASP може використовувати сигнатури для виявлення атак, вона не обмежується виявленням на основі сигнатур. Виявляючи і реагуючи на аномальну поведінку в захищеному додатку, RASP може виявляти і блокувати навіть атаки "нульового дня".

Зниження помилкових спрацьовувань: RASP має глибоке розуміння внутрішніх процесів програми, включаючи можливість побачити, як потенційна атака впливає на виконання програми. Це значно підвищує здатність RASP

відрізняти справжні атаки (які дійсно негативно впливають на продуктивність і безпеку додатків) від помилкових спрацьовувань (таких як спроби SQL-ін'єкцій, які ніколи не включаються в SQL-запит). Таке зменшення кількості хибних спрацьовувань зменшує навантаження на команди безпеки і дозволяє їм зосередитися на справжніх загрозах.

Зниження капітальних та експлуатаційних витрат: RASP розроблений таким чином, щоб бути простим у розгортанні, але в той же час здатний суттєво вплинути на вразливість додатків до атак та частоту хибнопозитивних сповіщень. Ця комбінація знижує як авансові витрати (CapEx), так і витрати на ефективний захист програми (OpEx) в порівнянні з ручним виправленням і брандмауерами для веб-додатків (WAF).

Простота обслуговування: RASP працює на основі розуміння програми, а не правил дорожнього руху, навчання або чорних списків. Команди SOC люблять цю надійність, а CISO цінують економію ресурсів. Додатки стають самозахищеними і залишаються захищеними, де б вони не знаходилися.

Гнучке розгортання: Хоча RASP, як правило, заснований на стандартах HTML, його API легко адаптувати для роботи з різними стандартами і архітектурами додатків. Це дозволяє захищати навіть не веб-додатки, що використовують такі стандарти, як XML і RPC.

Підтримка хмарних технологій: RASP розроблений для інтеграції та розгортання як частина програми, яку він захищає. Це дозволяє розгортати його в будь-якому місці, де можуть працювати захищені додатки, в тому числі в хмарі.

Підтримка DevSecOps: Рішення RASP розроблені для інтеграції в конвеєр безперервної інтеграції та розгортання DevOps (CI/CD). Це робить RASP простим у розгортанні та підтримує операції DevSecOps.

Гнучкість RASP означає, що розробники можуть інтегрувати його з багатьма різними додатками. Однак, певні випадки використання RASP є більш поширеними, такі як:

Захист веб-додатків: Веб-додатки та API є важливим компонентом інфраструктури організації, але можуть бути вразливими до широкого спектру

атак. Ці додатки знаходяться у відкритому доступі в Інтернеті і часто схильні до вразливостей, які можуть бути використані. Розгортаючи RASP для захисту цих додатків і API, організація може обмежити ризик кібербезпеки і поверхню атак на свою веб-інфраструктуру.

Запобігання з нульового дня: Хоча в організації можуть існувати процеси для негайного застосування патчів для критично важливих додатків і систем, патч може бути застосований тільки після того, як він розроблений і випущений. RASP може бути розгорнута для захисту критично важливих додатків в організації (які можуть включати веб-додатки та API) від вразливостей "нульового дня".

Захист хмарних додатків: Захист хмари може бути складним, оскільки додатки працюють на орендованій інфраструктурі за межами мережевого периметра організації. Інтеграція RASP в ці додатки забезпечує їм високий рівень безпеки в портативній і в значній мірі інфраструктурно-діагностичній формі.

Runtime Application Self-Protection і Web Application Firewall (WAF) є двома взаємодоповнюючими рішеннями для забезпечення безпеки додатків. У той час як WAF забезпечує першу лінію захисту, фільтруючи багато загроз для веб-додатків ще до того, як вони досягнуть цільової програми, RASP використовує контекст, що забезпечується глибокою видимістю цих додатків, для виявлення і блокування атак, які прослизають повз брандмауер веб-додатків. Ця комбінація мінімізує вплив атак, що легко виявляються, одночасно забезпечуючи захист від більш складних загроз.

Захист веб-додатків від сучасних загроз вимагає виходу за рамки використання RASP з WAF і заміни їх на сучасне рішення. Наступним поколінням WAF є автоматизований захист веб-додатків та API (WAAP). Рішення WAAP визнають той факт, що компанії все частіше виставляють програмні інтерфейси веб-додатків (API) в Інтернет та забезпечують комплексний захист як для веб-додатків, так і для API.

2.2. Аналіз можливостей методів та засобів захисту Web-додатків на базі WebSocket

У багатьох веб-технологіях сокети є універсальними. Тим не менш, більшість засобів контролю безпеки знаходяться в руках розробників. Розробники створюють уразливості через брак часу або недбалості та відсутності достатньої кількості часу. Найкращим рішенням для усунення цієї недбалості та зменшення вразливостей є оцінка безпеки, яка називається тестуванням на проникнення.

Проблеми безпеки, пов'язані з WebSocket API, мають фундаментальну основу, яка полягає в можливості встановити з'єднання між сторонами, не запитуючи дозволу в користувача; крім того, запит надсилається без сповіщення самого користувача. Атакуючому ж у такій ситуації достатньо виконати JavaScript код на стороні жертви, щоб змусити його встановити з'єднання з довільним сервером за протоколом WebSocket. З'єднання може бути використано атакуючим для обміну даними з жертвою. Таким чином, відбувається порушення вимоги безпеки "Безпечне управління сесіями" і "Контроль доступу", а також стає можливим порушення вимоги безпеки "Безпечне кешування". Оскільки не всі проксі-сервери коректно розуміють протокол WebSocket, зловмисник може змусити проксі кешувати нав'язані йому дані. Надалі ця вразливість застосовується для порушення інших вимог безпеки, шляхом нав'язування JavaScript коду клієнту жертви. Фундаментальна проблема породжує кілька загроз. Для цих загроз наводяться сценарії атаки, що показують те, як зловмисник може ними скористатися.

Проблеми безпеки WebSocket можна розділити на три частини:

Аутентифікація - це спосіб перевірки легітимного користувача в сервісу, і WebSocket не використовує конкретний метод аутентифікації. Найпоширенішим способом є заголовок set-cookie, який може залежати від розробника. Аутентифікація дозволяє кредити і ліцензії на ресурси і те, який доступ до даних може отримати людина. Авторизація сильно залежить від заголовка app-context. У

цьому випадку можуть виникнути три стани:

1. Зловмисник може бути в змозі досягти функції, яка не вимагає від веб-сервісу ліцензії більш високого рівня сервісу більш високого рівня, як було підтверджено з самого початку

2. Хакер може бачити вміст іншого користувача.

3. Хакер може отримати доступ до будь-чого без дозволу, і з цих причин безпека WebSocket є вкрай важлива

SOP(Same Origin Policy) - це метод безпеки, який зустрічається в більшості браузерах і обмеженнях. Наприклад, документи або скрипти (або інші дані), завантажені з певного джерела, обмежуються і заблоковані або не допускаються до виконання при спілкуванні.Більшість веб-додатків використовують цю політику і дотримуються її правил. Правила такі, що вони працюють на основі декількох конкретних фільтрів:

Номер порту

Різний протокол

Різні домени

url	sop	Explantion
http://store.company.com/dir2/other.html	yes	the same protocol, host and port.
http://store.company.com/dir/inner/another.html	yes	the same protocol, host and port.
https://store.company.com/secure.html	no	other protocol.
http://store.company.com:81/dir/etc.html	no	other port.
http://news.company.com/dir/other.html	no	other host.

Рис. 2.5. Same Origin Policy

Маршрути всередині авторизованого домену не є проблемою. Фільтрація більше базується на ставленні розробника і може визначати, який метод використовувати, а який не використовувати. Принцип роботи полягає в тому, що це відбувається за допомогою HTTP-заголовків.Існує заголовок, який називається

Origin, і якщо його значення дорівнює '*', то дозволені всі домени. Якщо вказані деякі домени, то приймаються дані тільки тих самих доменів.

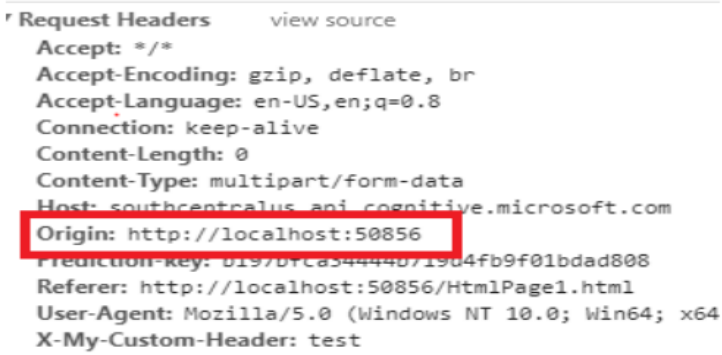


Рис.2.6. Заголовок origin

CSWH (Cross-Site-Web-Socket-Hijacking) - це різновид CSRF-атак. Вона створює серйозні проблеми з безпекою і може викрасти всю інформацію клієнта. Як і інші баги, він може бути використаний в залежності від креативності хакера і може робити все, що він забажає. Ця атака є більш клієнтською і працює з інформацією користувача.

Для того, щоб зловмисник здійснив свій план, йому потрібно привести жертву на шкідливу сторінку, тому він використовує такі методи, як як XSS (Cross-Site-Scripting) або Open-Redirection. Хакер може творчо підійти до відправки шкідливого JavaScript коду в браузер жертви, використовуючи хакерський інструмент, такий як BeEF. Якщо заголовок origin неконтрольований, шкідливий код може бути запущений. Він дозволяє встановити зв'язок з цільовим сайтом.

Хакери завжди прагнуть зробити щось краще, вдосконалити, і працювати до тих пір, поки їх рішення не стане майже бездоганним. Зловмисники можуть використовувати цю помилку трьома можливими способами:

Припустимо, що ми використовуємо WebSocket для спілкування, наприклад в обережних операціях, таких як грошові транзакції, додатки в реальному часі та передача даних. В такому випадку хакер може видати себе за людину і почати надсилати конфіденційну інформацію.

Припустимо, що ми використовуємо WebSocket в місцях, де ми можемо відновити нашу інформацію, наприклад, забувши пароль. В такому випадку хакер може зробити фальшивий запит і почати отримувати конфіденційні дані і, врешті-решт захоплення облікового запису.

Хакер може почати прослуховувати трафік між клієнтом і сервером, реалізувавши атаку «людина посередині» (MITM).

Зловмисник відправляє запит і починає зловживати вхідними параметрами, починаючи надсилати фейкову інформацію в браузер жертви. Він відправляє запит на з'єднання, але перед цим додає токен заголовка. Тепер хакери можуть виконати JavaScript-шкідливе програмне забезпечення, щоб отримати доступ до до даних клієнта.

```
<html>
    <body>
        <script>
            var ws= new WebSocket("http://examplesite.org","chat");
            ws.send("attacker, message");
            ws.onmessage=function(event)
            {
                $.post("attacker-example.com",event.data);
            }
        </script>
    </body>
</html>
```

Рис.2.7. Сценарій атаки CSWSH

Коли відбувається рукостискання, ми повинні контролювати вхідні дані і бачити, якими даними обмінюються. Особливо потрібно бути більш уважними до значень заголовків під час рукостискання і перевіряти значення їх походження. Таким чином, щоб дані з певних доменів могли потрапляти тільки в наш домен. Нам не потрібно встановлювати в заголовку origin значення '*', щоб відправити нам будь-які дані; ми повинні надати список допустимих доменів, щоб Origin мав до них доступ.

У цьому випадку слід використовувати метод типу CSRFTOKEN. Нам потрібно згенерувати на сервері токен або випадкове значення на сервері, через

який хакер не зможе підключитися, поки не отримає його. Це значення буде відправлено на сервер, і йому буде дозволено встановити, чи є воно правильним; якщо хакер не має дійсного токену, йому не дозволений доступ, і це найкраще рішення.

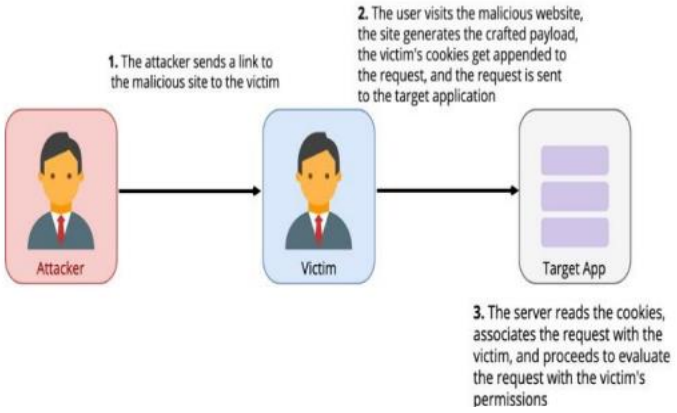


Рис.2.8. CSWH

Шифрування використовується для захисту конфіденційності даних. Коли дані передаються, про це знають тільки дві сторони. Без шифрування зловмисник може прочитати трафік за допомогою MITM (man in the middle). Якщо зловмисник може виконати MITM-атаку, він може прочитати всю інформацію, якою обмінюються, що є неприпустимим.

За замовчуванням WebSocket використовує порт 80, який не шифрується. Ми використовуємо порт 443 для захисту наших з'єднань, а саме SSL, який зберігає нашу інформацію від будь-яких MITM-атак.

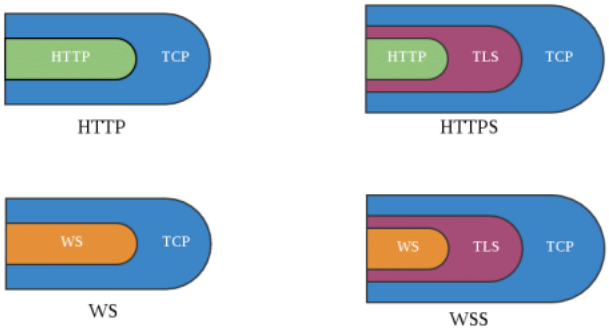


Рис.2.9. Шифрування даних під час використання WebSocket

Ключ Sec-WebSocket-Key - це випадковий рядок з 16 байтами в діапазоні ASCII значень, що представляють "nonce" для захищеного зв'язку. Клієнт надсилає 16 байт даних у форматі, закодованому в base64 форматі. Заголовок Sec-WebSocket-Accept використовується в рукостисканні при відкритті WebSocket.Цей заголовок відправляється від сервера клієнту, щоб повідомити, що сервер готовий ініціювати WebSocket-з'єднання. Сервер спочатку отримує Sec-WebSocket-Key, відправлений клієнтом в першому запиті і об'єднує його з "магічним рядком". Потім обчислює SHA-1 нового результату і відправляє base64 хешованого значення.

Sec-WebSocket-Accept:base64(SHA-1(sec-WebSocketkey + “258EAFA5-E914-47DA-95CA-C5AB0DC85B11“))

258EAFA5-E914-47DA-95CA-C5AB0DC85B11 - це магічний рядок.

У деяких випадках програміст не хоче створювати кращий розв'язок задачі і використовує подібні ключі для спрощення етапів тестування.

2.3. Безпека Web-додатку при використанні технології WebRTC

Web Real-Time Communication (WebRTC) - це API, призначений для підтримки взаємодії між браузерами. Дане рішення покликане об'єднати розрізнений кластер рішень, який до цього часу домінував на ринку. Всі нововведення на сьогоднішній день являють собою окремі програми або плагіни для браузерів. Це обмежує інтеграцію цих рішень у зростаючий світ браузерів. WebRTC є рішенням для обміну файлами P2P, потокового P2P відео та аудіо дзвінків, а також для комунікації в режимі реального часу.

На даному етапі було обрано практичний підхід до вивчення існуючих загроз для додатків WebRTC. Зловмисник може намагатися отримати інформацію та маскуватися під учасника, а також довільної активації медіа-пристроїв.

Потрібно розглянути кожного із задіяних гравців: клієнта, сервер, та інформаційний канал - і для кожного потрібно шукати ймовірні загрози та їх вплив на систему, щоб відповісти на питання:

Клієнт - Які наслідки ін'єкції JavaScript/HTML?

Клієнт - чи можна вкрасти облікові дані WebRTC?

Клієнт - чи можна викрасти привілейовану інформацію про клієнта?

Сервер - Які наслідки захоплення сигнального сервера?

Сервер - Чи можемо ми вивести з ладу сервер або зробити його неактивним?

Інформаційний канал - яка інформація може бути вилучена?

Інформаційний канал - чи можемо ми змусити клієнта підключитися до мережі?

Для кожного учасника слід досліджувати, чи можна перервати розмову, викрасти інформацію або видати себе в розмові, викрасти інформацію або видати себе за користувача.

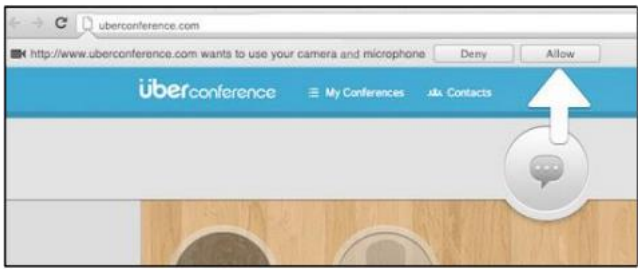


Рис.2.10. Запит на отримання дозволу використання камери та мікрофону

Хоча WebRTC впроваджує заходи безпеки в міру свого контексту, важливо зазначити, що WebRTC працює в межах хостингу браузера. Таким чином, звичайна атака веб-додатків впливає на середовище WebRTC і може бути використана для отримання інформації та виконання дій від імені жертви.

Припустимо, що додаток вразливий до міжсайтового скриптингу, що дозволяє користувачеві вставляти HTML і JavaScript в контекст програми. Хоча класичні XSS-атаки (міжсайтовий скриптинг) здебільшого пом'якшуються перевіркою вхідних даних на стороні сервера, додатки WebRTC дозволяють користувачам обмінюватися інформацією безпосередньо без сервера в якості посередника. Такий зв'язок означає, що зловмисник може відправити жертві шкідливий код жертві, вбудований в текстове повідомлення, ім'я користувача або

ім'я вкладеного файлу, який надсилається в режимі P2P.

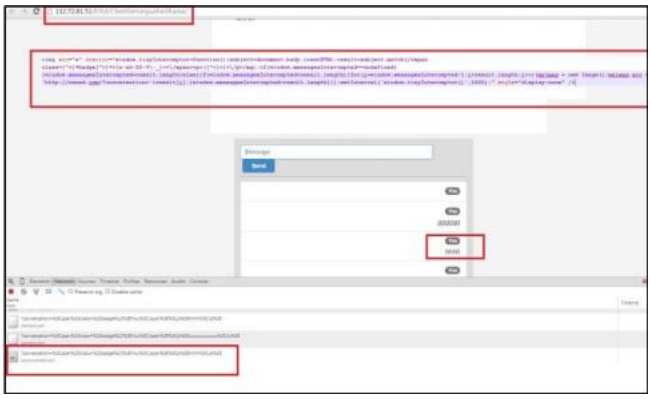


Рис.2.11. Поле чату WebRTC вразливе до XSS

Для пом'якшення наслідків цієї атаки необхідно, щоб приймаюча сторона виконувала перевірку вхідних даних, наданих користувачем. Шкідливий вміст може виконуватися в певних сценаріях, але не в інших. Наприклад, шкідливе корисне навантаження може бути відфільтроване в панелі чату або воно може виконуватися в панелі історії чату.

Розглянемо можливість атаки на сервер сигналізації WebRTC, відправивши на сервер сигналізації неправильний JSON. Об'єкт JSON аналізується як єдине ціле, що означає, що помилка в одній частині об'єкта, призведе до помилки в усьому процесі розбору. Зловмисник відправляє на сервер неправильно сформований JSON у складі легітимного запиту (наприклад, нове поле в повідомленні ICE кандидата в повідомленнях ICE). Коли сервер намагається розібрати JSON, він видасть помилку та вийде з основного циклу програми. Наслідком цієї атаки є те, що сервер перестане реагувати на нові запити. Результатом атаки стане компрометація доступності сервера.

Якщо припустити, що зловмисник може отримати доступ до сервера сигналізації, то він може:

(1) Вивести систему з ладу.

(2) Підключити випадкових користувачів до розмови між собою.

(3) Переадресувати зв'язок, створивши "невидимого" користувача що бере

участь у розмові в конференції

(4) Виконання MITM шляхом створення фальшивого користувача; кожен клієнт буде спілкуватися з фальшивим користувачем, вважаючи, що вони спілкуються один з одним. Фальшивий користувач буде проксі-сервером.

Хоча WebRTC використовує стандартне шифрування для своєї інформації, все одно може бути можливим отримати деяке уявлення про тип розмови користувачів і, можливо, навіть витягти іншу корисну інформацію, не вдаючись до шифрування. Тому і досі викликає питання, чи може зловмисник класифікувати підкатегорії WebRTC-зв'язку(наприклад, надсилається файл, аудіо- чи відеодзвінок), а також додаткову корисну інформацію (якщо ми можемо приблизно оцінити розмір та/або визначити тип файлу який був відправлений).

Окрім того не слід забувати про атаки на відмову в обслуговуванні (DOS), спрямовані на сервер або канал передачі даних.

WebRTC є хорошим прикладом правильно спроектованої технології; вона демонструє швидке розгортання, а також високу гнучкість і адаптацію до оточуючих протоколів, схем шифрування та хостингових середовищ. Для більшості, WebRTC демонструє чіткі можливості коли мова йде про конфіденційність і безпеку. Але WebRTC не є автономним рішенням; технологія вимагає хост-додаток, хостинг-сервер і транспортний рівень, що з'єднує їх. Було описано декілька атак на кожен з цих основних компонентів що показує, що хоча додаток WebRTC поставляється з набором сильних функцій безпеки, він не застрахований від прямих атак проти його компонентів, а також уразливостей на його базовому хост-додатку.

3 РОЗРОБЛЕННЯ РЕКОМЕНДАЦІЙ ЩОДО УДОСКОНАЛЕННЯ БЕЗПЕКИ WEB-ДОДАТКІВ НА БАЗІ WEBSOCKET ВІД СУЧАСНИХ ЗАГРОЗ

3.1. Бібліотека з відкритим кодом Socket.io

Socket.IO - це бібліотека JavaScript для веб-додатків у режимі реального часу. Вона забезпечує двосторонній зв'язок між веб-клієнтами та серверами в режимі реального часу. Вона складається з двох частин - клієнтської бібліотеки, яка працює в браузері, та серверної бібліотеки для node.js. Обидва компоненти мають ідентичний API. Написання додатку реального часу з використанням популярних стеків веб-додатків, таких як LAMP (PHP), традиційно було дуже складним. Це пов'язано з опитуванням сервера на предмет змін, відстеженням міток часу, а це набагато повільніше, ніж повинно бути.

Сокети традиційно є рішенням, навколо якого будується більшість систем реального часу, забезпечуючи двосторонній канал зв'язку між клієнтом і сервером. Це означає, що сервер може передавати повідомлення клієнтам. Всякий раз, коли відбувається подія, ідея полягає в тому, що сервер отримає його і передасть відповідним підключеним клієнтам.

Socket.IO є досить популярним, його використовують Microsoft Office, Yammer, Zendesk, Trello та багато інших організацій для побудови надійних систем реального часу. Це один з найпотужніших JavaScript фреймворків на GitHub, і найбільш залежний від модуля NPM (Node Package Manager).

Хоча Socket.IO дійсно використовує WebSocket для транспортування, коли це можливо, він додає додаткові метадані до кожного пакету. Тому клієнт WebSocket не зможе успішно підключитися до сервера Socket.IO, а клієнт Socket.IO не зможе підключитися до звичайного сервера WebSocket.

Створення екземплярів клієнта та сервера за допомогою бібліотеки дуже легке.

Для початку потрібно завантажити бібліотеку і всі потрібні пакети.

```
const socket = new WebSocket("ws://localhost:3000");

socket.addEventListener("open", () => {
  // send a message to the server
  socket.send(JSON.stringify({
    type: "hello from client",
    content: [ 3, "4" ]
  }));
});

// receive a message from the server
socket.addEventListener("message", ({ data }) => {
  const packet = JSON.parse(data);

  switch (packet.type) {
    case "hello from server":
      // ...
      break;
  }
});
```

Рис.3.1. Створення екземпляру клієнта

```
import { WebSocketServer } from "ws";

const server = new WebSocketServer({ port: 3000 });

server.on("connection", (socket) => {
  // send a message to the client
  socket.send(JSON.stringify({
    type: "hello from server",
    content: [ 1, "2" ]
  }));

  // receive a message from the client
  socket.on("message", (data) => {
    const packet = JSON.parse(data);

    switch (packet.type) {
      case "hello from client":
        // ...
        break;
    }
  });
});
```

Рис.3.2 Створення екземпляру сервера

Тепер бібліотека готова до використання.

3.2. Програмна реалізація захищеного real-time Web-додатку на базі WebSocket

Постановка задачі:

Створення кімнати для групових відеодзвінків з наступною поведінкою:

- Коли хтось починає відеодзвінок (відкриваючи url), користувачеві присвоюється нова кімната з унікальним ідентифікатором кімнати користувач чекає, слухає і чекає на нові з'єднання, щоб прийняти їх
- Користувач може скопіювати і відправити url (з ідентифікатором кімнати) іншій людині

- Користувач може приєднуватись до інших кімнат.
- Сервер сповіщає існуючих учасників кімнати про те, що з'явився новий користувач
- Користувач може зберігати відеозаписи існуючих учасників кімнати

При розробці сучасного Web-додатку, слід звертати увагу не тільки на UI/UX, а, в першу чергу, на захист даних, якими оперує застосунок. Про безпеку інформації потрібно думати безпосередньо перед початком проектування Web-додатку для корпоративної інформаційної системи та використовувати перевірені технології налаштування захисту та тестування в процесі самого проектування. Зазвичай, перед розробкою MERN-додатку, першочергово розробляється код для серверної частини – backend. В даному випадку, в якості мови програмування на бекенді було обрано Node JS. Перед написанням коду повинен бути ініціалізований проект за допомогою менеджеру пакетів для Node JS – npm (node package manager). При написанні команди npm init, ініціалізується новий проект і створюється спеціальний файл – package.json, в якому зберігається назва та версія інстальованих залежностей для проекту та скрипти, які запускають додаток. Рекомендується с самого початку написання проекту подбати про те, які пакети в якості залежностей потрібно інсталювати. Це залежить від того, якої складності Web-додаток і на що він націлений. В даному випадку, застосунок взаємодіє із базою даних MongoDB, в якій зберігаються зареєстровані користувачі. Тому знадобляться пакети для роботи із СУБД, щоб підв’язати код Web-додатку із базою даних та для того, щоб забезпечити захист передачі конфіденційних даних користувачів шляхом їх шифрування.

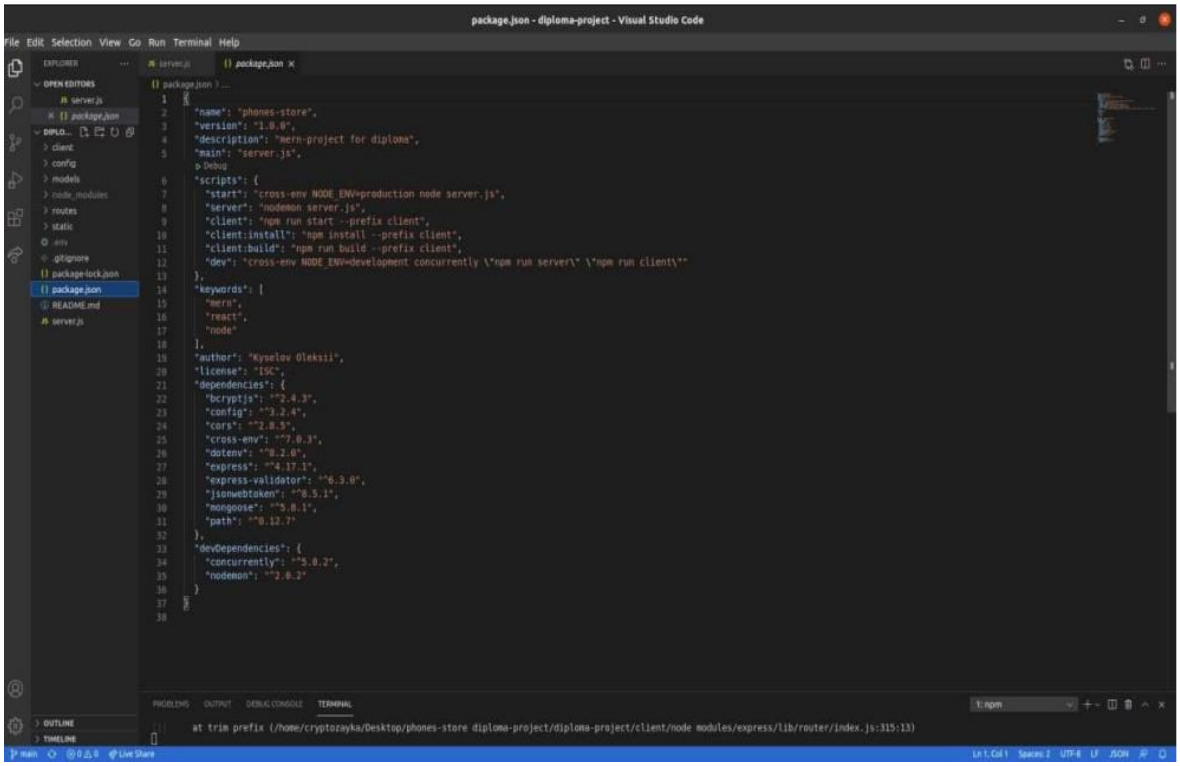


Рис.3.3 Файл встановлених пакетів package.json

В будь-якому проекті повинен бути головний файл, який містить посилання на інші – цей файл називається “точка входу”, бо коли спрацьовує скрипт запуску додатку, машина шукає саме цей файл. В цьому проекті, на стороні бекенду – це файл server.js, на стороні фронтенду – index.html. Саме в файл server.js було імпортовано деякі головні залежності проекту, серед яких – бібліотека Express JS, для роботи із безпечною маршрутизацією сторінок Web-додатку, налаштування заголовків політики безпеки CORS та з’єднання із базою даних MongoDB.

Спочатку необхідно створити екземпляр сервера. Для безпеки додаємо до cors тільки ті домейн, які нам знайомі, localhost:3000, власний IP та домейн для задеплоєного додатку.

```
const server = http.createServer(app)
const io = new Server(server, {
  cors: {
    origin: process.env.REACT_APP_API_URL || [
      'http://localhost:3000',
      'http://192.168.43.146:3000'
    ]
  }
})
```

Рис.3.4. Оголошення екземпляру сервера

У той момент, коли клієнт входить до системи, ми відправляємо до сокета подію, яка приєднує клієнта до чату, тобто створить для клієнта кімнату. Ідентифікатором цієї кімнати є id чату. Це поле міститься в контексті чату.

```
useEffect(() => {
  if (currentChatData?._id) {
    const joinData = { chatId: currentChatData?._id }

    socket.emit(SOCKET_EVENTS.CHAT_JOIN, joinData)
  }
}, [currentChatData, isAdmin])
```

Рис.3.5. Подія для приєднання до чату

На сервері створюються обробники для цих подій (функції, які будуть спрацьовувати, коли ці події стануть активними).

```
socket.on(SOCKET_EVENTS.CHAT_JOIN, ({ chatId }) => {
  socket.join(chatId)
  socket.on(SOCKET_EVENTS.MESSAGE, ({ messageData, to }) => {
    socket.to(chatId).to(to).emit(SOCKET_EVENTS.MESSAGE, {
      messageData,
      from: socket._id
    })
  })
  socket.on(SOCKET_EVENTS.ORDER_CHANGE, (orderData) => {
    socket.to(chatId).emit(SOCKET_EVENTS.ORDER_CHANGE, {
      orderData,
      from: socket._id
    })
  })
})
```

Рис.3.6. Обробники подій на сервері

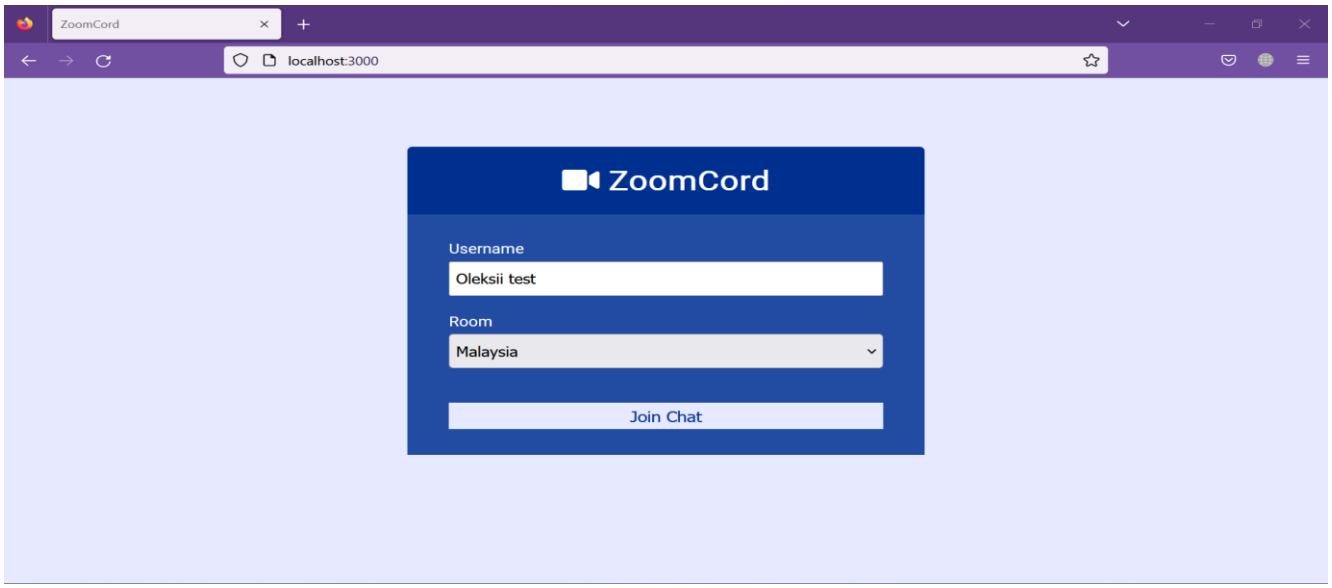


Рис.3.7. Інтерфейс додатку

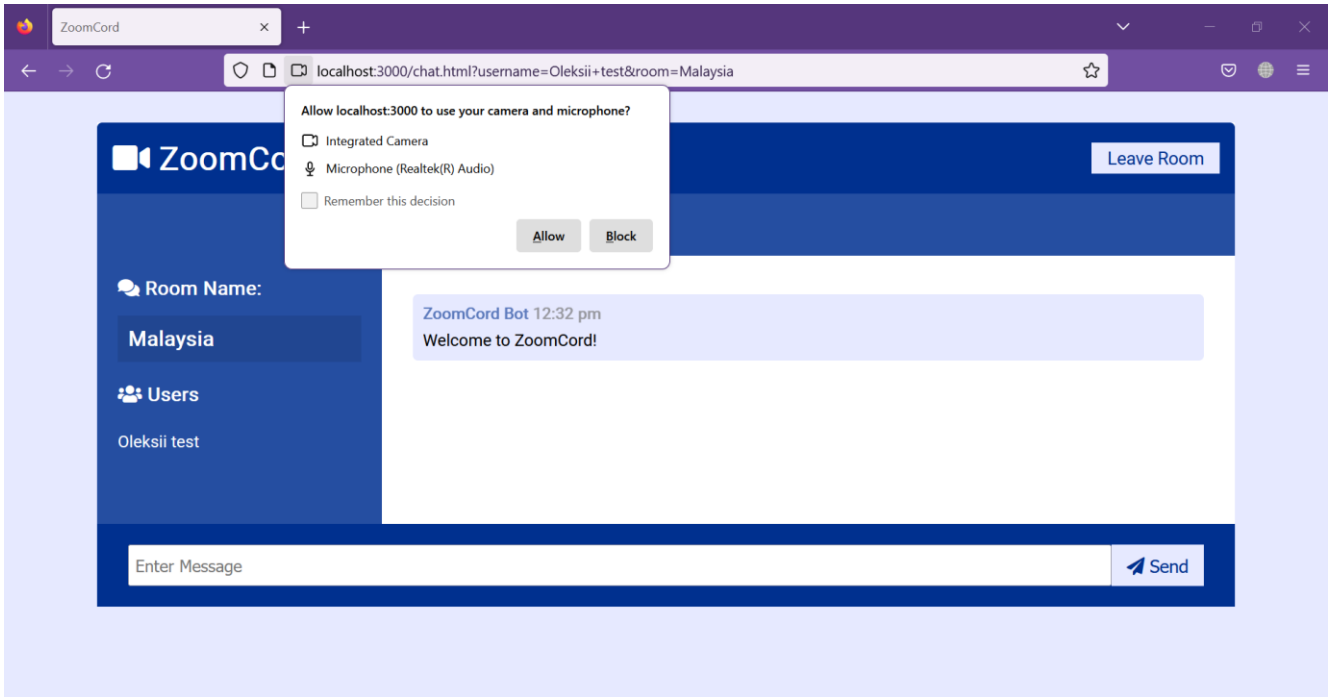


Рис.3.8. Запит на підключення камери та мікрофону

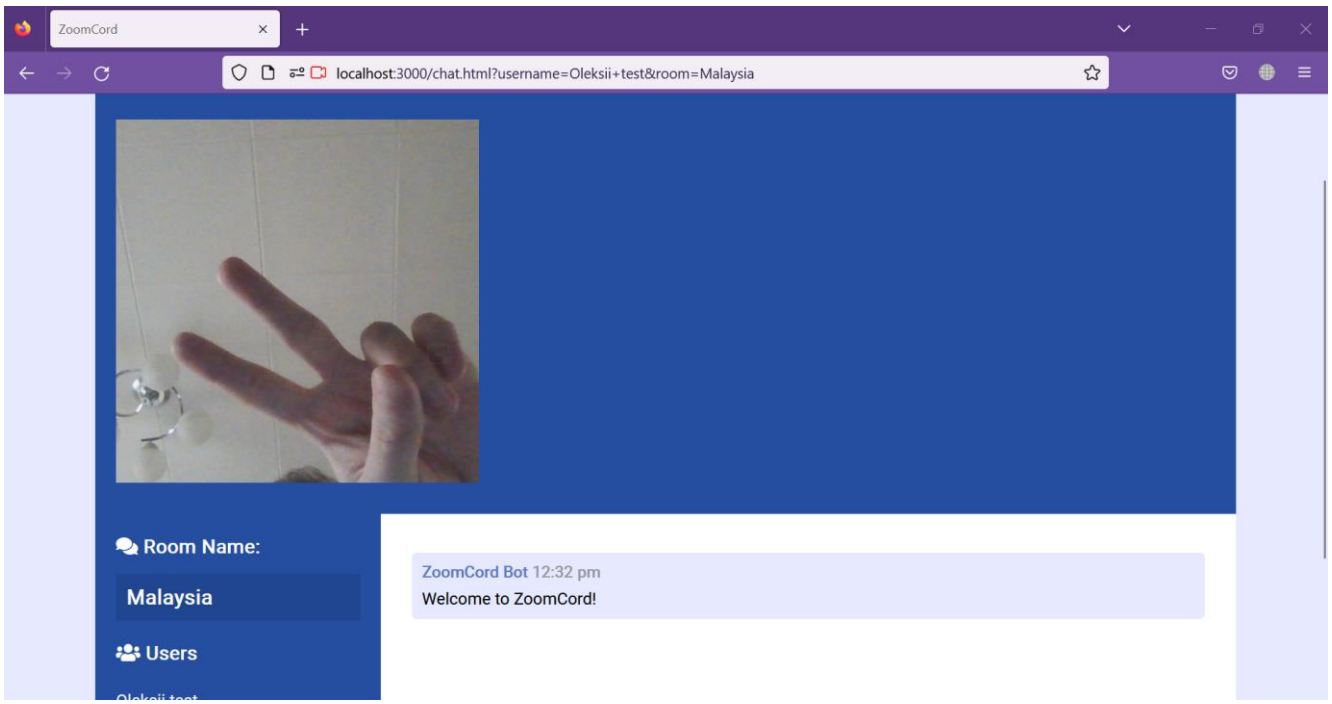


Рис.3.9. Працююча камера

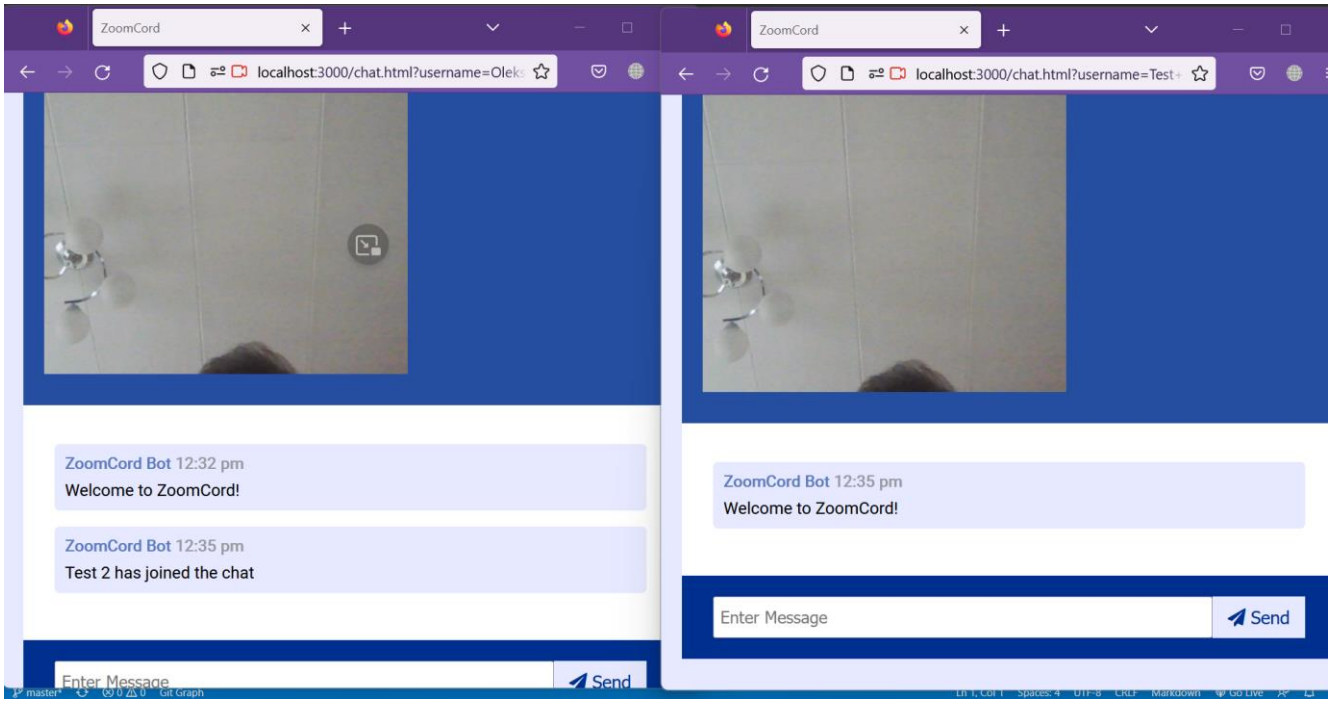


Рис.3.10. Під’єднання іншого учасника в кімнату

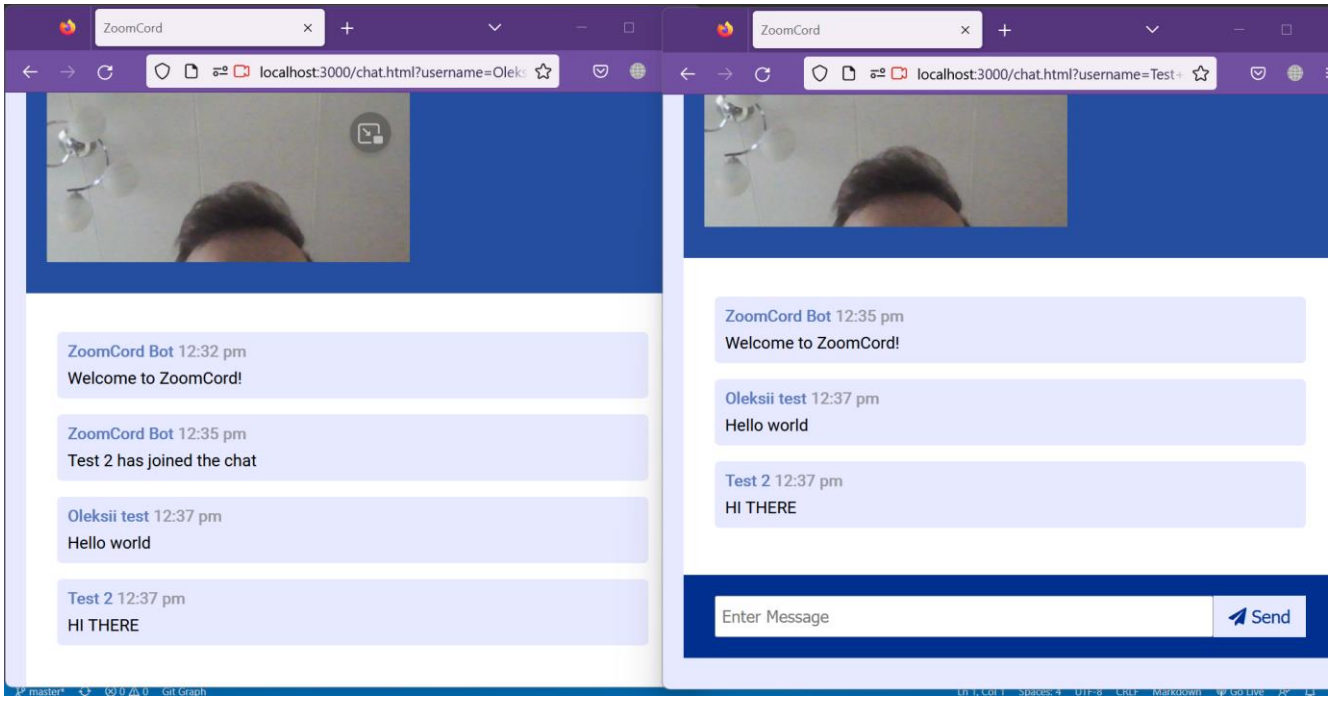


Рис.3.11. Демонстрація працюючого чату

```
const path = require('path');
const express = require('express');
const app = express();
const server = require('http').createServer(app);
const io =  require('socket.io')(server);
const formatMessage = require('./utils/messages');
const { userJoin, getCurrentUser, userLeave, getRoomUsers } =
require('./utils/users');

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => console.log(`Server listening on port ${PORT}`));

// Routing
app.use(express.static(path.join(__dirname, 'public')));

const botName = 'ZoomCord Bot';
const ACCEPTED_ROOMS = ["Malaysia", "Indonesia", "Singapore"];

// Run when client connets
io.on('connection', socket => {
    socket.on('joinRoom', ({ userPeerId, username, room }) => {
        if (!ACCEPTED_ROOMS.includes(room)) {
            socket.emit('roomNotValid');
        }

        const user = userJoin(userPeerId, username, room);

        if (!user) {
            socket.emit('sameName');
        } else {

            socket.join(user.room);

            // only show in client to the user connecting
            socket.emit('message', formatMessage(botName, 'Welcome to ZoomCord!'));

            // Broadcast to all except the user itself in a specif room
            socket.broadcast
                .to(user.room)
                .emit(
                    'message',
                    formatMessage(botName, `${user.username} has joined the chat`)
                );

            socket.broadcast.to(user.room).emit('user-connected', userPeerId)

            // Send users and room info
            io.to(user.room).emit('roomUsers', {
                room: user.room,
```

```
                users: getRoomUsers(user.room) // E.g return: [{id:
'6JhtU8cQGMZzzzj5AAAB', username: 'kaiimran', room: 'Malaysia'}]
            });

            socket.on('typing', () => {
                console.log('typing');

                socket.broadcast
                    .to(user.room)
                    .emit('typing', {
                        username: user.username
                    });
            });

            socket.on('stop typing', () => {
                console.log('stop typing');

                socket.broadcast
                    .to(user.room)
                    .emit('stop typing', {
                        username: user.username
                    });
            });

            // Listen for chatMessage
            socket.on('chatMessage', msg => {
                const user = getCurrentUser(userPeerId);

                // Broadcast to all clients in the room
                io
                    .to(user.room)
                    .emit('message', formatMessage(user.username, msg));
            });

            // Runs when client disconnects
            socket.on('disconnect', () => {
                const user = userLeave(userPeerId);

                if (user) {
                    io
                    .to(user.room)
                    .emit('message', formatMessage(botName, `${user.username} has
left the chat`));
                }

                // Send users and room info
                io.to(user.room).emit('roomUsers', {
                    room: user.room,
                    users: getRoomUsers(user.room)
                });
```

```
                socket.broadcast.to(user.room).emit('user-disconnected', userPeerId)
            });
        }
    });

});
```

Рис.3.12. Код сервера додатка

Після безпосередньо розробки Web-додатку слід його протестувати. Тести потрібні не тільки для того, щоб знайти і виправити помилки в додатку при розробці нового функціоналу. Тести дуже часто допомагають при рефакторінгу коду переконатися, що нічого не було зламано. Тестування в процесі розробки Web-додатку потрібно для того, щоб контролювати всі логічні компоненти робочого коду задля гарантії, що вони будуть робити те, для чого їх було створено і функції будуть повертати результат, який від них очікується. Для тестування React-компонентів рекомендується використовувати такі відомі бібліотеки, як: @testing-library/react, Jest та Enzyme. Кожна з них має свої переваги та недоліки. Вони більше відрізняються синтаксисом і мають практично однаковий функціонал. Чим складніше взаємозв’язок між компонентами в логіці Web-додатку, тим складніше за побудовою будуть тести.

У створеному застосунку присутні браузерні події, такі як кліки по кнопкам, скролл тощо. В тестах повинна бути явно прописана логіка і функціонал - що буде відбуватися, коли виникне та чи інша подія. Також, при тестуванні рекомендується використовувати спеціальні функції моки, щоб замінити ними функціонал зовнішніх бібліотек і функцій для того, щоб тести спрацьовували коректно.

Функція мок - це функція без реалізації, яку можна передати в компонент, щоб не використовувати код реальної функції і щоб тестувати код компонента без прив'язки до зовнішніх залежностей.

Для демонстрації тестування був обраний один із компонентів.

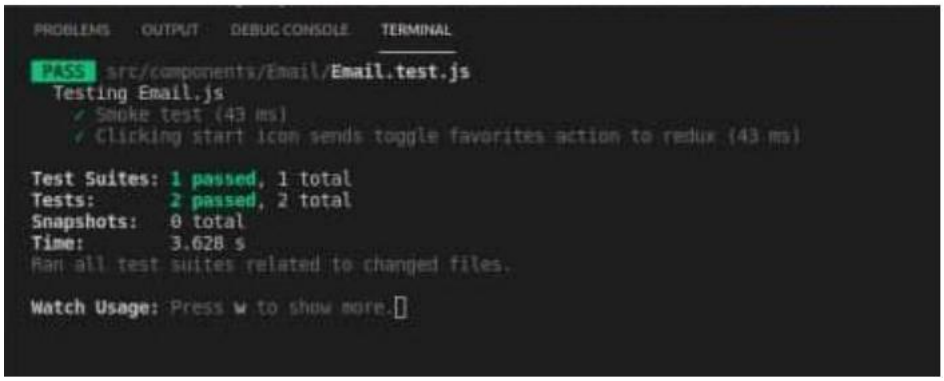


Рис.3.13. Результати тестування

В сучасному світі слід впроваджувати різні технології безпечної розробки. Таким чином, окрім тестування логіки роботи коду, рекомендується використовувати спеціальні утиліти, розроблені спеціально для пошуку потенційних вразливостей, які може використати зловмисник у своїх цілях. Хорошу репутацію має плагін eslint-security-plugin, який використовує SAST-тестування. Даний пакет, окрім того, що вказує формат розробки додатку, щоб синтаксис всього коду був однаковий, без зайвих відступів і символів, знаходить помилки в проекті, які потенційно можуть бути небезпечними.

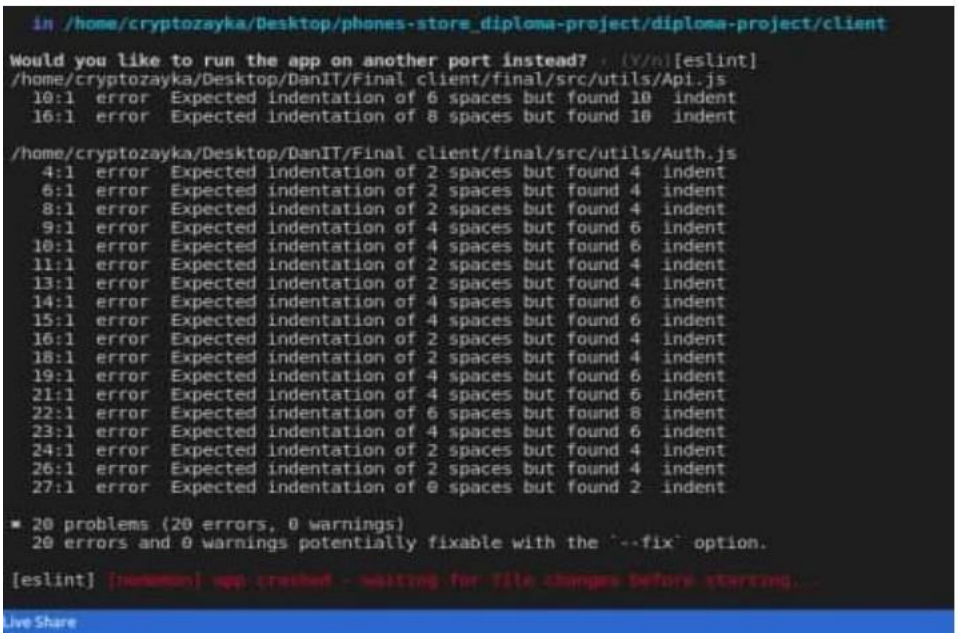


Рис.3.14. eslint-plugin

SAST-тестування дозволяє розробникам знаходити вразливості безпеки у вихідному коді додатку на ранніх етапах життєвого циклу проектування програмного забезпечення, також даний тип тестування забезпечує відповідність руководствам і стандартам кодування без фактичного виконання коду. За статистикою, 90% інцидентів безпеки виникають в результаті використання зловмисниками відомих програмних помилок. Усунення вразливостей на етапі розробки значно знижує ризики інформаційної безпеки.

3.3. Загальні рекомендації щодо застосування технології забезпечення безпеки Web-додатку на базі WebSocket

Після того, як розглянули вразливості, слід теж звернути увагу на комплексний підхід захисту Web-додатку на базі WebSocket.

Використання тільки WSS:

Не варто використовувати ws://, це не безпечний транспортний протокол. Замість цього використовуйте wss://, який є набагато безпечнішим через шифрування. WSS є безпечним, тому він запобігає таким речам, як атаки типу "зловмисник посередині".

WebSockets не є стандартною реалізацією сокетів. WebSockets універсальні, встановлене з'єднання завжди відкрите, і повідомлення можуть надсилатися і прийматися безперервно. Однак, DOS-атаки, відсутність аутентифікації/авторизації, вразливість до атак на введення даних - все це вразливості, які можуть бути використані. Ось чому важливо використовувати перевірку даних, що вводяться клієнтом і сервером, автентифікацію на основі квитків і WSS.

Перевірка даних, що вводяться клієнтом:

З'єднання WebSocket можуть бути легко встановлені поза браузером. Ви будете мати справу з довільними даними, незважаючи ні на що. Ці дані потребують перевірки, як і будь-які інші дані, що надходять від клієнта, перш ніж вони будуть

оброблені. Чому? Тому що через WebSockets можливі атаки типу OS, SQL, Blind SQL.

Перевірка даних на сервері:

Вам не потрібно турбуватися тільки про перевірку клієнтських даних. Дані, які повертає сервер, також можуть нести в собі проблеми. Повідомлення, отримані на стороні клієнта, завжди повинні оброблятися як дані. Присвоювати ці повідомлення безпосередньо DOM або оцінювати як код не рекомендується. У випадку відповідей у форматі JSON використовуйте JSON.parse() у поєднанні з обробкою винятків і, якщо необхідно, спеціальними методами санітарної обробки для безпечного аналізу даних.

Аутентифікація на основі квитків:

Як згадувалося раніше, протоколи WebSocket не обробляють авторизацію або аутентифікацію. Як же тоді підвищити безпеку WebSocket? Оптимізувавши і захистивши ваше з'єднання. Ми не можемо налаштувати заголовки WebSocket з JavaScript. На жаль, всі обмежуються "неявною" авторизацією (cookies), яку надсилає браузер. І це ще не все, оскільки сервери, які обробляють WebSockets, зазвичай відокремлені від тих, які обробляють стандартні HTTP-запити. Це значно ускладнює спільне використання заголовків авторизації. На щастя, існує шаблон, який допомагає вирішити проблему аутентифікації WebSocket. Система аутентифікації на основі квитків, яка працює наступним чином:

У випадку, коли код на стороні клієнта намагається відкрити WebSocket, HTTP-сервер зв'язується з клієнтським кодом, щоб дозволити клієнтському коду отримати (авторизаційний) квиток, після чого генерується сам квиток, який містить ідентифікатор користувача/облікового запису, IP-адресу того, хто запитує квиток, мітку часу та інші внутрішні записи. Квиток зберігається на сервері/базі даних і повертається клієнту, який тепер може відкрити з'єднання WebSocket і відправити цей квиток разом з початковим рукостисканням. Тепер у сервера є можливість порівняти тікет, оцінити IP-адресу джерела, перевірити безпеку тікета (якщо він використовується повторно) і т.д. Коли все підтверджується, WebSocket-з'єднання отримує верифікацію

ВИСНОВКИ

В магістерській роботі опрацьовано сучасну літературу за даною темою, проведено аналіз міжнародних норм та експлуатаційної документації. Досліджено існуючі Web-додатки на прикладі системи комунікації в реальному часі та проведено аналіз стану їх захищеності. Проведено дослідження методів та засобів захисту інформації Web-додатку на базі WebSocket. Розроблено комплексний підхід щодо проектування захищеного Web-додатку в корпоративній інформаційній системі за допомогою мови програмування JavaScript та сучасних фреймворків. Протестовано створений Web-додаток. Розроблено рекомендації щодо удосконалення безпеки інформації Web-додатків від сучасних загроз. Як наслідок, дані рекомендації можна вважати актуальними і доцільно використовувати в сфері забезпечення кібербезпеки та при розробці сучасних Web-додатків у корпоративних інформаційних системах.

ПЕРЕЛІК ПОСИЛАНЬ

1. OWASP Top 10:2021, Режим доступу: World Wide Web - URL - https://owasp.org/Top10/A01_2021-Broken_Access_Control/
2. Seven tips for effective identity management, Режим доступу: World Wide Web. - URL - https://www.computerworld.com/article/2566203/seven-tips-effective-identity-management.html
3. SocketIO, Режим доступу: World Wide Web. - URL – https://socket.io/docs/v4/
4. What is Single Sign On, Режим доступу: World Wide Web. - URL - https://cloudinfrastructureservices.co.uk/what-is-single-sign-on-and-how-does-sso-work/
5. Кисельов Олексій Володимирович. ПРОТОКОЛ WEBSOCKET ЯК МЕХАНІЗМ ЗАХИСТУ КАНАЛІВ ПЕРЕДАВАННЯ ІНФОРМАЦІЇ. ВСЕУКРАЇНСЬКА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ «АКТУАЛЬНІ ПРОБЛЕМИ КІБЕРБЕЗПЕКИ», Державний Університет Телекомунікацій. 27 жовтня 2022 р., Тези доповідей С. 163-166. https://dut.edu.ua/uploads/p_2121_20358827.pdf
6. 1. Закон України «Про захист інформації в автоматизованих системах»: Закон України від 05.10.94 № 80/94-ВР // Відомості Верховної Ради України. – 1994. – № 31. – Ст. 286.
7. Закон України “Про захист персональних даних”
8. Information Security and Development Problems eGoverment System in Ukraine - KhmelevskoyR., Khmelevskoy Y., Kozachok V., Semko V., Ilin O. [p.74]
9. Кібербезпека в інформаційному суспільстві: Інформаційно-аналітичний дайджест / відп. ред. О.Довгань; Науково-дослідний інститут інформатики и права НАПрН України; Національна бібліотека України ім. В.І. Вернадського. -К., 2020 No11 (листопад). -281с
10. Хмелевський Р.М. Дослідження оцінки загроз інформаційній безпеці об’єктів інформаційної діяльності / Р.М. Хмелевський // Сучасний захист інформації. –2016. –No 4. –С. 65-70.

11. БЕЗПЕКА WEB-ДОДАТКІВ: АКТУАЛЬНІ ПРОБЛЕМИ ТА ЇХ АНАЛІЗ О. Бондаренко, І.Ушкаленко

12. Дослідження і розробка методики та моделі захисту інтернет-магазинів на основі метода побудови дерев атак - к. т.н., доц. Губенко Н.Є.,Гаранжа О.В.

13. Богуш В. М., Юдін О. К.Б74 Інформаційна безпека держави. –К.: “МК-Прес”, 2005.

14. В. Л. Бурячок, В. Б. Толубко, В. О. Хорошко, С. В. Толюпа (2015). У В. Б. Толубко (загальна редакція).Інформаційна кіберберзпека: соціотехнічний аспект. Київ:Державний університет телекомунікацій.ISBN978-966-2970-86-9. 10.

15. User Guide for Cisco Security MARS Local Controller, release 5.3.2 – Cisco System, Inc, 2007. – 702 c

16. Информационная безопасность открытых систем: учебник для вузов. В 2-х томах. Том 1 – Угрозы уязвимости, атаки и подходы к защите / С.В. 69 Запечников, Н.Г. Милославская, А.И. Толстой, Д.В. Ушаков. – М.: Горячая линия-Телеком, 2006 – 536с.

17. Рекомендації щодо захисту інформації на етапах проектування Web-додатку. Web 2.0 [Електронний ресурс] Режим доступу: http://www.dut.edu.ua/uploads/n_9126_17047934.pdf

18. Стандарт HTML [Електронний ресурс] Режим доступу: https://html.spec.whatwg.org/multipage/workers.html

19. Amazon Web Services [Електронний ресурс] Режим доступу: https://www.ripublication.com/ijaer18/ijaerv13n22_87.pdf

20. AWS Security [Електронний ресурс] Режим доступу: https://d1.awsstatic.com/whitepapers/Security/Intro_to_AWS_Security.pdf?did=wp_card&trk=wp_card

21. AWS Encryption for JS and Node JS [Електронний ресурс] Режим доступу: https://aws.amazon.com/blogs/security/how-to-enable-encryption-browser-aws-e ncryption-sdk-javascript-node-js/

22. Securing SPA`s [Електронний ресурс] Режим доступу:

https://wso2.com/blogs/thesource/securing-spas-best-practices/

23. CVSSv3.1 [Електронний ресурс] Режим доступу: https://www.first.org/cvss/specification-document

24. Web Application Security Consortsium [Електронний ресурс] Режим доступу: http://www.webappsec.org/

25. Technical guide to information security testing NIST [Електронний ресурс] Режим доступу: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf

26. OWASP Web Security Testing Guide [Електронний ресурс] Режим доступу: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWAS P_Testing_Guide_v4.pdf 22.OWASP TOP-10 [Електронний ресурс] Режим доступу: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017-ru.pdf

27. Дерева атак [Електронний ресурс] Режим доступу: https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D1%8C%D1%8F_%D0%B0%D1%82%D0%B0%D0%BA 24.НД ТЗІ 1.1-002-99 [Електронний ресурс] Режим доступу: https://tzi.com.ua/downloads/1.1-002-99.pdf 2

28. Threat modelling for Node JS Applications [Електронний ресурс] Режим доступу: https://laptrinhx.com/threat-modelling-for-node-js-applications-395172481/

29. Кисельов Олексій Володимирович. Дослідження шляхів та вироблення рекомендацій щодо захисту інформації на етапах проектування Web-додатку для корпоративних інформаційних систем», Бакалаврська робота, Державний Університет Телекомунікацій.

30. https://brightsec.com/blog/websocket-security-top-vulnerabilities/#improve-websocket-security

31. https://realtime-apps-iap.github.io/docs/introduction/webrtc

32. https://medium.com/geekculture/understand-whats-behind-real-time-web-apps-e60168129480

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ
КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Магістерська робота
на тему:

«ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ WEB-ДОДАТКА НА БАЗІ WEBSOCKET»

Виконав: КИСЕЛЬОВ Олексій Володимирович

Керівник: ГАЙДУР Галина Іванівна, д.т.н., проф.

2

Об'єкт дослідження – процес захисту інформації на етапах проектування Web-додатку на базі WebSocket

Предмет дослідження – технологія забезпечення безпеки Web-додатка на базі WebSocket

Мета роботи – розробити варіант захисту інформації Web-додатку на базі WebSocket від сучасних загроз та надати рекомендації щодо удосконалення їх безпеки під час проектування

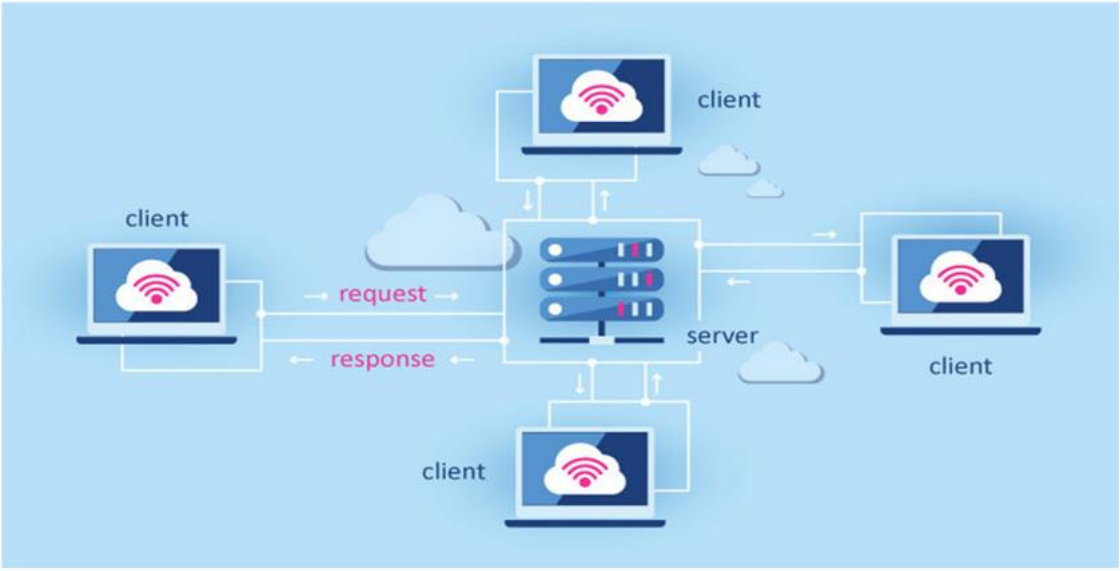
Наукові завдання:

- дослідити сутність проблеми безпеки Web-додатків на базі WebSocket;
- проаналізувати існуючі системи комунікації в реальному часі;
- проаналізувати методи та засоби захисту Web-додатків на базі WebSocket;
- розкрити порядок проектування технології забезпечення безпеки Web-додатка на базі WebSocket.

Аналіз роботи сучасних Web-додатків на базі архітектури клієнт-сервер

3

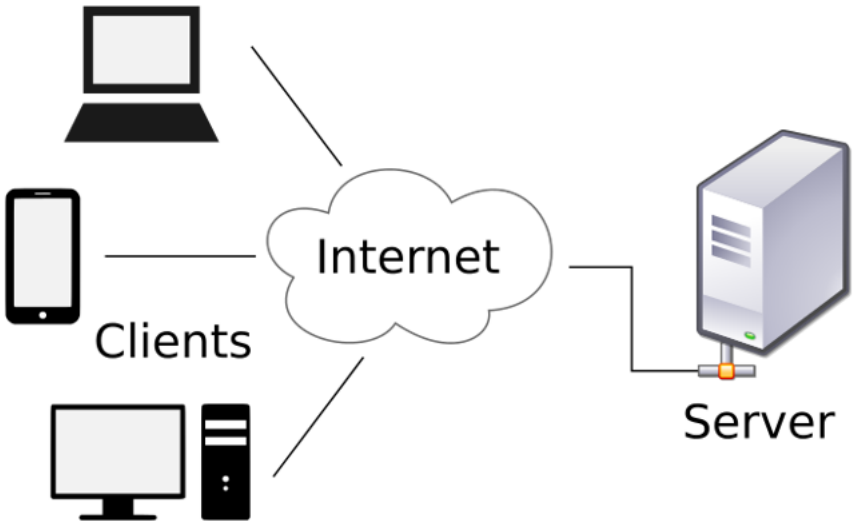
Система клієнт-сервер є різновидом клієнт-серверних обчислень. Клієнти та сервери можуть працювати між різними комп'ютерами, об'єднаними в мережу. Зазвичай клієнтом є персональний комп'ютер, який запитує і використовує послугу, а сервером - комп'ютер, який може бути ПК або міні-мейнфреймом, що надає послугу. Таким чином, основна архітектура системи клієнт-сервер - це клієнт, сервер і третій рівень, тобто модель, яка має два типи в своїй системі, яка називається дворівневими системами.



Використання технологій під час проектування захищених real-time Web-додатків

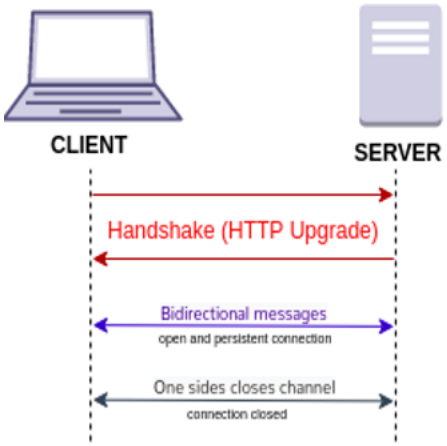
4

Short Polling - клієнт робить HTTP-запити з фіксованою затримкою
Long Polling - клієнт робить HTTP-запит, сервер чекає максимально дозволений час, перш ніж відповісти, клієнт робить ще один HTTP-запит і так далі.
Обидва ці методи пов'язані з невеликими накладними витратами на пропускну здатність мережі. Що ще важливіше, вони зловживають функціональністю HTTP для досягнення бажаної поведінки!



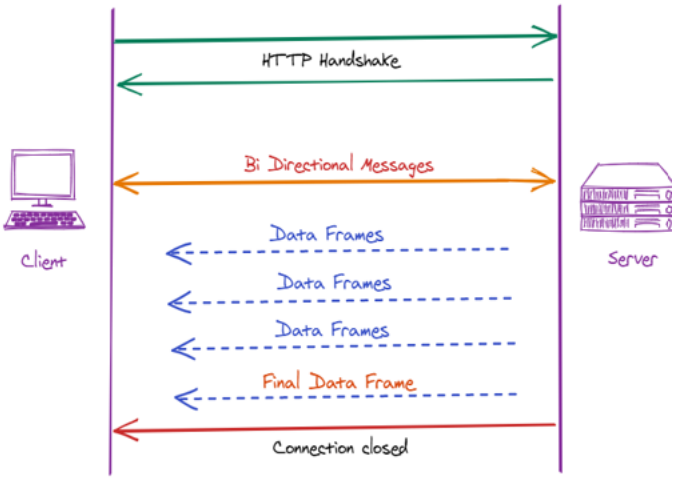
WebSocket

5



WebSocket - протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу. Протокол повністю асинхронний та симетричний. Він застосовується для організації чатів, онлайн-табло тощо і створює постійне з'єднання між клієнтом та сервером, яке обидві сторони можуть використовувати для надсилання даних.

WebSocket можуть створювати до 50 з'єднань, уникаючи обмежень на максимум 6 з'єднань, які мають події, що надсилаються сервером. Вони можуть передавати повідомлення в текстовому або двійковому форматі. Таким чином, їх можна використовувати для передачі текстових повідомлень, відео та аудіо.



WebSocket client

6

```
const socket = new WebSocket("ws://localhost:3000");

socket.addEventListener("open", () => {
  // send a message to the server
  socket.send(JSON.stringify({
    type: "hello from client",
    content: [ 3, "4" ]
  }));
});

// receive a message from the server
socket.addEventListener("message", ({ data }) => {
  const packet = JSON.parse(data);

  switch (packet.type) {
    case "hello from server":
      // ...
      break;
  }
});
```

WebSocket server

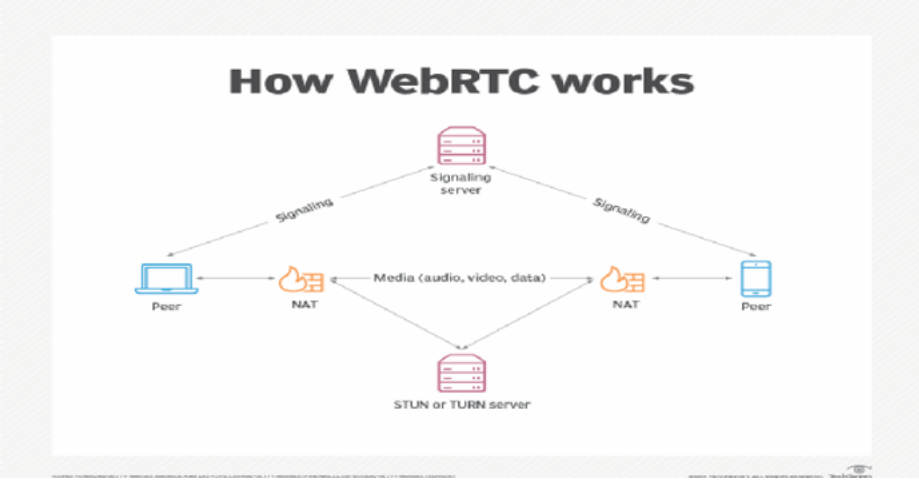
7

```
import { WebSocketServer } from "ws";

const server = new WebSocketServer({ port: 3000 });

server.on("connection", (socket) => {
  // send a message to the client
  socket.send(JSON.stringify({
    type: "hello from server",
    content: [ 1, "2" ]
  }));

  // receive a message from the client
  socket.on("message", (data) => {
    const packet = JSON.parse(data);

    switch (packet.type) {
      case "hello from client":
        // ...
        break;
    }
  });
});
```

WebRTC

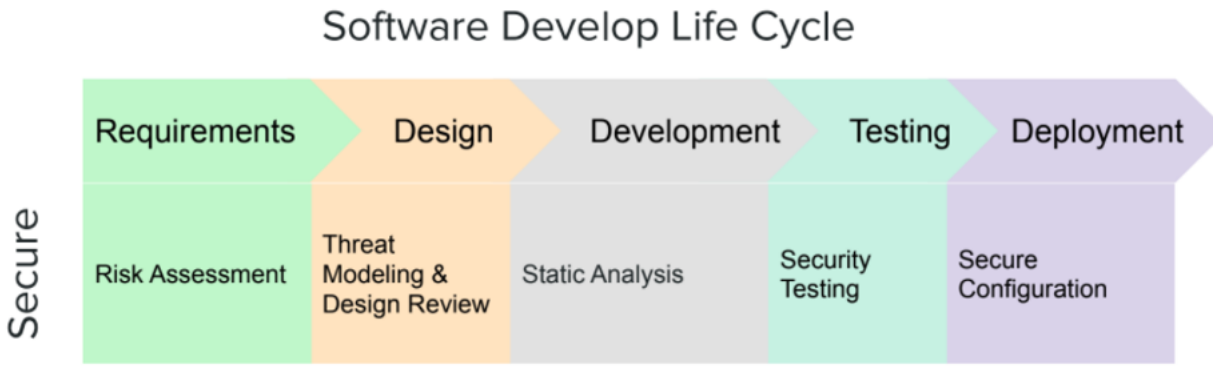
8

WebRTC є хорошим прикладом правильно спроектованої технології; вона демонструє швидке розгортання, а також високу гнучкість і адаптацію до оточуючих протоколів, схем шифрування та хостингових середовищ. Для більшості, WebRTC демонструє чіткі можливості коли мова йде про конфіденційність і безпеку. Але WebRTC не є автономним рішенням; технологія вимагає хост-додаток, хостинг-сервер і транспортний рівень, що з'єднує їх.



Secure Software Development Life Cycle

9



На етапі опрацювання вимог потрібно зробити оцінку ризиків. На етапі розробки архітектури - архітектурне рев'ю, моделювання загроз. На етапі розробки - перевірку аналізаторами.

Після чого варто зробити тестування безпеки. А під час деплою подбати про правильну конфігурацію застосунків.