Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

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

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: «Засоби захисту технології WebRTC від DoS-атак»

Виконавець: студент IV курсу, групи КБ-42

_____ **Денис ДРУЖКО**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я ПРІЗВИЩЕ**
Керівник		Юрій БАБЕНКО
Нормоконтроль		Інна МИХАЛЬЧУК

Київ 2025

Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

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

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-42** (група) **Дружку Денису Сергійовичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Засоби захисту технології WebRTC від DoS-атак

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

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

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Структура та архітектура технології WebRTC, технічні принципи обміну даними в реальному часі, механізми побудови сигнальних з'єднань, сценарії реалізації DoS-атак, програмні інструменти для захоплення, аналізу й фільтрації мережевого трафіку.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Огляд архітектури та компонентів технології WebRTC, аналіз типових вразливостей до DoS-атак, дослідження сценаріїв реалізації DoS-атак у WebRTC-додатках, методи аналізу трафіку в реальному часі, розробка та тестування системи виявлення вторгнень на основі характеристик WebRTC-трафіку.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Програмне рішення, яке реалізує функціонал системи виявлення DoS-атак для WebRTC-додатків

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав _____ (підпис) Юрій БАБЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Денис ДРУЖКО (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 01.12.2024	*виконано*
2	Аналіз літератури	02.12.2024 – 20.01.2025	*виконано*
3	Обґрунтування вибору рішення	21.01.2025 – 26.01.2025	*виконано*
4	Вивчення архітектури технології WebRTC та механізми її функціонування	27.01.2025 – 15.02.2025	*виконано*
5	Виявлення можливих векторів DoS-атак у WebRTC-середовищі	16.02.2025 – 28.02.2025	*виконано*
6	Підготовка тестового середовища	01.03.2025 – 09.03.2025	*виконано*
7	Аналіз трафіку WebRTC-додатку в різних умовах	10.03.2025 – 26.03.2025	*виконано*
8	Реалізація системи виявлення DoS-атак для WebRTC-додатків	27.03.2025 – 20.04.2025	*виконано*
9	Оформлення пояснювальної записки	21.04.2025 – 25.05.2025	*виконано*
10	Підготовка до захисту кваліфікаційної роботи	26.05.2025 – 13.06.2025	*виконано*

Завдання видав _____ (підпис) Юрій БАБЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Денис ДРУЖКО (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить вступ, три розділи, загальні висновки, список використаних джерел та додатки. Обсяг основного тексту становить 72 сторінки, робота містить 16 рисунків. Список використаних джерел налічує 26 найменувань і займає 3 сторінки.

Метою роботи є підвищення захищеності WebRTC-додатків від DoS-атак шляхом розробки системи виявлення вторгнень на основі моніторингу мережевого трафіку.

Для досягнення мети були поставлені наступні завдання:

- провести аналіз архітектури та принципу роботи технології WebRTC;
- визначити вразливості WebRTC-додатків до DoS-атак та проаналізувати існуючі захисні механізми;
- підготувати тестове середовище для подальшого дослідження WebRTC-додатків;
- виконати захоплення та аналіз мережевого трафіку WebRTC-додатку для виявлення характерних ознак DoS-атак;
- розробити систему виявлення DoS-атак у WebRTC-додатках на основі моніторингу мережевого трафіку.

Об'єктом дослідження є процес забезпечення стійкості технології WebRTC до атак типу «відмова в обслуговуванні» на основі аналізу характеристик мережевого трафіку в режимі реального часу.

Предметом дослідження є методи та засоби виявлення атак типу «відмова в обслуговуванні» на WebRTC-додатки шляхом аналізу мережевого трафіку.

Практична цінність отриманих результатів полягає у розробці системи виявлення DoS-атак у WebRTC-додатках, що базується на аналізі характеристик трафіку в реальному часі.

Ключові слова: WebRTC, DoS-атаки, signaling-сервер, STUN, TURN, DTLS, реальний час, аналіз трафіку, IDS, система виявлення вторгнень.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

CORS	–	Cross-Origin Resource Sharing
DoS	–	Denial-of-Service
DTLS	–	Datagram Transport Layer Security
eBPF	–	Extended Berkeley Packet Filter
HTTP	–	Hypertext Transfer Protocol
ICE	–	Interactive Connectivity Establishment
IDS	–	Intrusion Detection System
NAT	–	Network Address Translation
P2P	–	Peer-to-Peer
RTCP	–	Real-time Transport Control Protocol
RTP	–	Real-time Transport Protocol
SCTP	–	Stream Control Transmission Protocol
SDP	–	Session Description Protocol
SIP	–	Session Initiation Protocol
SOP	–	Same Origin Policy
SRTP	–	Secure Real-time Transport Protocol
STUN	–	Session Traversal Utilities for NAT
TURN	–	Traversal Using Relays around NAT
VoIP	–	Voice over Internet Protocol
WebRTC	–	Web Real-Time Communication
XMPP	–	Extensible Messaging and Presence Protocol

ВСТУП

Актуальність. В результаті світової нестабільності останніх років у вигляді пандемій, військових конфліктів та інших катаклізмів спостерігається стрімке зростання попиту на дистанційні засоби комунікації. Все більше людей та організацій покладаються на онлайн-платформи для проведення нарад, навчання, надання медичних послуг і щоденної взаємодії. У цьому контексті WebRTC стала однією з ключових технологій, яка забезпечує захищений зв'язок у реальному часі між користувачами через веббраузери та мобільні додатки. Проте разом із широким розповсюдженням WebRTC зростає й кількість загроз, спрямованих на її компоненти, зокрема атаки на відмову в обслуговуванні, які можуть легко вивести з ладу критичні комунікаційні сервіси.

З постійним розвитком різноманітних кіберзагроз важливо не лише захистити систему від зламу, а й забезпечити її стабільну роботу навіть під час активної атаки. Атаки типу DoS на WebRTC-додатки можуть призвести до зупинки важливих сервісів – таких як відеозв'язок під час операцій, екстрених нарад чи освітніх заходів. Тому дослідження, спрямовані на виявлення та попередження таких атак, мають не лише технічну, але й соціальну вагу. Розробка простих у реалізації та ефективних засобів захисту WebRTC від DoS-атак є важливою умовою для підтримки стабільної та безпечної комунікації в цифрову епоху.

Метою роботи є підвищення захищеності WebRTC-додатків від DoS-атак шляхом розробки системи виявлення вторгнень на основі моніторингу мережевого трафіку.

Для досягнення мети були поставлені наступні *завдання*:

- провести аналіз архітектури та принципу роботи технології WebRTC;
- визначити вразливості WebRTC-додатків до DoS-атак та проаналізувати існуючі захисні механізми;
- підготувати тестове середовище для подальшого дослідження WebRTC-додатків;

- виконати захоплення та аналіз мережевого трафіку WebRTC-додатку для виявлення характерних ознак DoS-атак;
- розробити систему виявлення DoS-атак у WebRTC-додатках на основі моніторингу мережевого трафіку.

Об'єктом дослідження є процес забезпечення стійкості технології WebRTC до атак типу «відмова в обслуговуванні» на основі аналізу характеристик мережевого трафіку в режимі реального часу.

Предметом дослідження є методи та засоби виявлення атак типу «відмова в обслуговуванні» на WebRTC-додатки шляхом аналізу мережевого трафіку..

Практична цінність отриманих результатів полягає у розробці системи виявлення DoS-атак для WebRTC-додатків. Запропонований інструмент працює в реальному часі, аналізуючи параметри трафіку (частоту пакетів, їх розмір, наявність відповідей), що дозволяє оперативно реагувати на загрози без значного навантаження на обчислювальні ресурси.

Апробація роботи. Основні результати роботи доповідались на VIII Міжнародній науково-практичній конференції «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSICS) 11 квітня 2025 року.

РОЗДІЛ 1
ТЕХНОЛОГІЯ WebRTC ТА ЇЇ ОСНОВНІ ХАРАКТЕРИСТИКИ

1.1 Сутність технології WebRTC та її призначення

Із розвитком вебтехнологій зросла потреба в швидкому та безперебійному обміні даними між користувачами, пристроями та системами призвела до активного розвитку технологій, здатних забезпечити прямий зв'язок без проміжних серверів або встановлення додаткового програмного забезпечення. Однією з таких інноваційних технологій стала Web Real-Time Communication.

WebRTC – це відкритий стандарт і набір API, який дозволяє здійснювати браузер-браузерну або браузер-мобільну комунікацію в реальному часі за допомогою звичайних вебтехнологій. Основна мета WebRTC полягає в тому, щоб забезпечити безпосередню передачу аудіо, відео та довільних даних між кінцевими пристроями без необхідності встановлення плагінів чи сторонніх клієнтів, використовуючи лише функціональність браузера.

Завдяки широкому набору функцій WebRTC надає можливість реалізовувати різні типи комунікацій у реальному часі: аудіо- та відеозв'язок, обмін файлами, трансляцію екрана та інтерактивну взаємодію між користувачами. Однією з ключових особливостей є підтримка прямого з'єднання, що дозволяє мінімізувати затримки, зменшити навантаження на серверну інфраструктуру та підвищити якість передавання даних.

Крім комунікаційних можливостей, WebRTC підтримує обмін довільними даними між клієнтами, що відкриває шлях до створення не лише відеозв'язку, а й багатьох інших інтерактивних застосунків: від онлайн-ігор до спільної роботи з документами. Така універсальність функціоналу зробила WebRTC базовою технологією для багатьох сучасних вебсервісів, орієнтованих на інтерактивність та доступність.

Ініціатором розробки WebRTC стала компанія Google, яка у 2011 році придбала компанію Global IP Solutions (GIPS), що спеціалізувалася на розробці

технологій VoIP. Саме на основі рішень GIPS були реалізовані базові можливості для передачі медіа в реальному часі. Після цього Google відкрила вихідний код цих технологій і передала проєкт на розвиток відкритій спільноті під назвою WebRTC.

У тому ж 2011 році Google анонсувала WebRTC як open-source-проєкт, метою якого є забезпечення нативної підтримки аудіо- та відеозв'язку в браузерах без використання сторонніх компонентів. Основу платформи становлять такі браузерні API, як getUserMedia, RTCPeerConnection та RTCDataChannel, які дозволяють отримувати доступ до камери та мікрофона, встановлювати peer-to-peer з’єднання та передавати довільні дані [1].

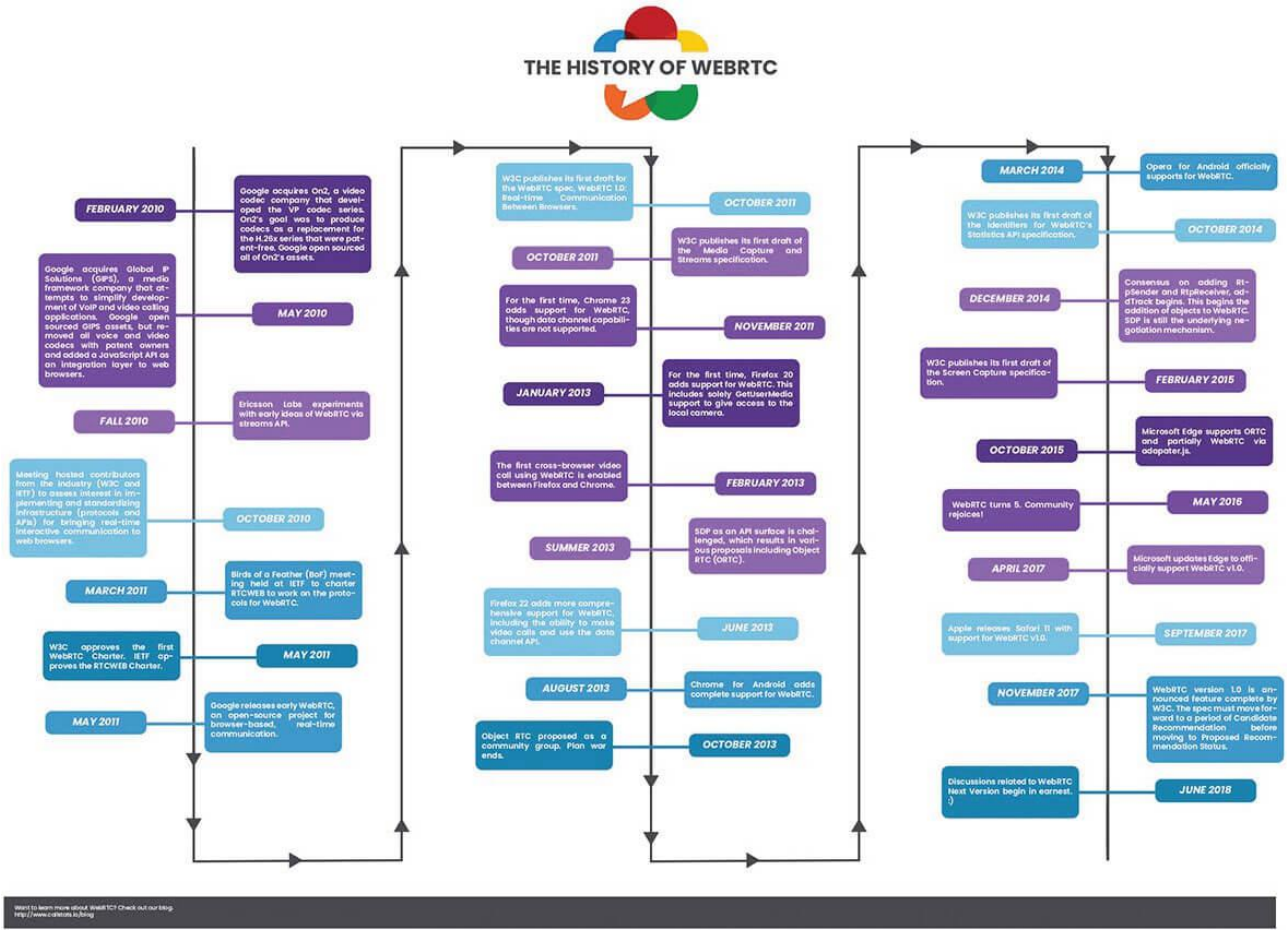

Рисунок 1.1 – Схема розвитку технології WebRTC

З 2012 року технологія почала активно підтримуватися основними браузерами – Google Chrome, Mozilla Firefox, Opera, а згодом і Microsoft Edge. У 2018 році WebRTC була офіційно стандартизована консорціумом W3C (World

Wide Web Consortium) та робочою групою IETF (Internet Engineering Task Force), що стало важливою віхою у розвитку цієї технології.

Протягом наступних років WebRTC зазнала значних змін, зокрема було покращено якість медіа передавання, реалізовано підтримку масштабованої багатокористувацької взаємодії (SFU/MCU сервери), а також інтегровано засоби безпеки, такі як шифрування трафіку через DTLS і SRTP. Більш детально розвиток цієї технології можна ознайомитись за рис. 1.1.

Сьогодні WebRTC є однією з ключових технологій для реалізації відеоконференцій, онлайн-дзвінків, ігрових сервісів, телемедицини, систем дистанційного навчання та багатьох інших рішень, де важливе значення має передача даних у реальному часі. Наприклад, Google Meet використовує WebRTC для організації відеоконференцій прямо у браузері Chrome, забезпечуючи одночасний зв’язок між десятками учасників. Facebook Messenger реалізує відео- та аудіодзвінки через WebRTC як у вебверсії, так і в мобільному браузері. Також технологія лежить в основі Slack, де вона відповідає за вбудовані голосові дзвінки між користувачами.

У галузі відкритих рішень варто виділити Jitsi Meet – повноцінну систему відеоконференцій, яка повністю працює на WebRTC. Її використовують державні установи, освітні заклади та бізнес-компанії як безкоштовну альтернативу комерційним платформам. Ще одним прикладом є BigBlueButton – система для дистанційного навчання, де WebRTC забезпечує аудіо- та відеозв’язок між викладачем і студентами, інтеграцію з Moodle, обмін екранами та спільну роботу над документами [2].

Також WebRTC використовується у спеціалізованих сервісах, зокрема в телемедицині (наприклад, Doxy.me і VSee) для проведення консультацій між лікарями та пацієнтами, або у відеоспостереженні – камери з підтримкою WebRTC дозволяють транслювати зображення у браузер без встановлення клієнтів (наприклад, Ant Media Server або Kerberos.io). В e-commerce WebRTC забезпечує інтерактивну підтримку клієнтів у реальному часі через вбудовані відеочати на сайтах. Така широка інтеграція демонструє гнучкість та універсальність цієї технології у вирішенні завдань сучасного цифрового світу.

1.2 Архітектура та ключові принципи функціонування WebRTC

WebRTC побудована як набір відкритих стандартів і програмних інтерфейсів, що забезпечують організацію медіа- та дата-комунікацій безпосередньо між веббраузерами або іншими сумісними пристроями. Архітектура цієї технології базується на принципі peer-to-peer (P2P) взаємодії, що дозволяє уникати використання серверів для обробки мультимедійного трафіку, зменшуючи затримки та підвищуючи якість передавання.

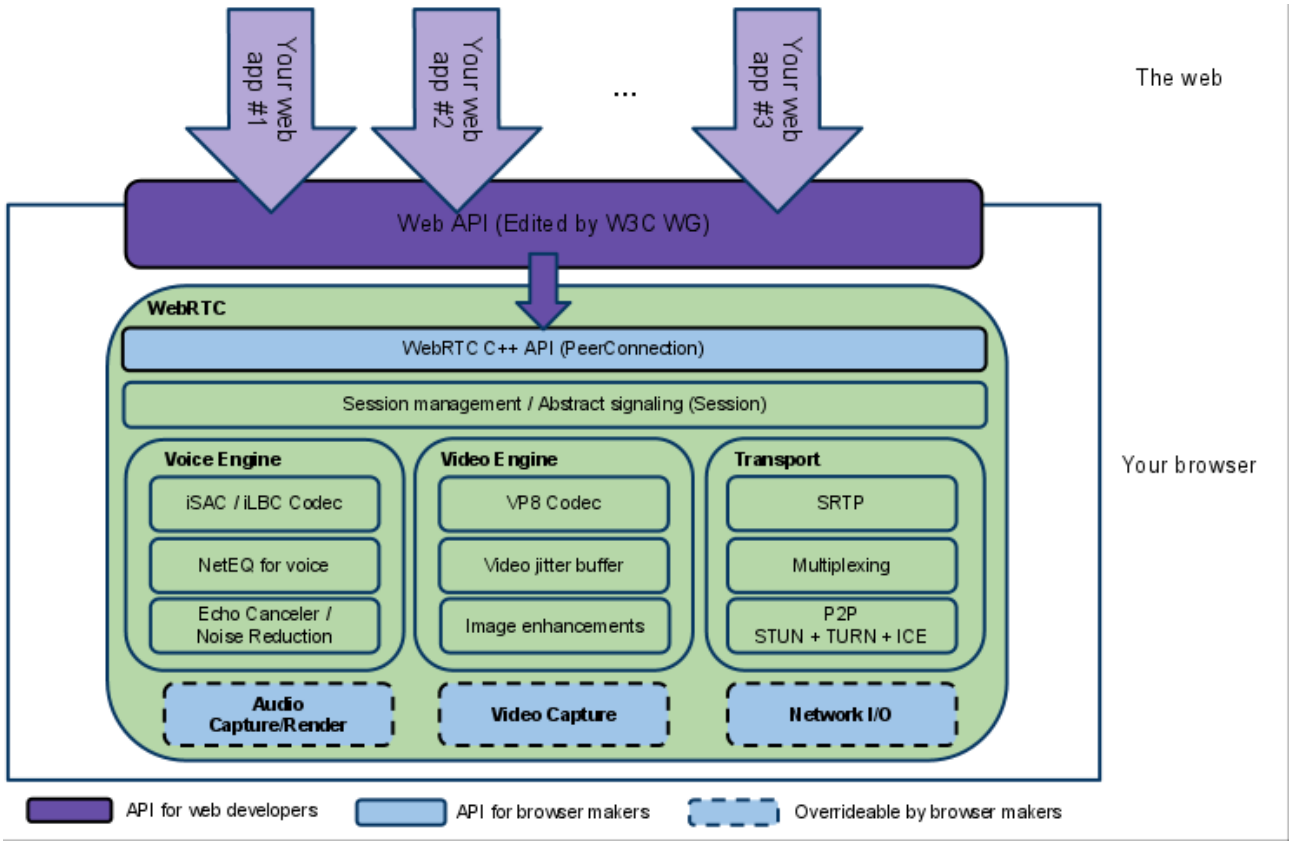

Рисунок 1.2 – Архітектура технології WebRTC

На рис. 1.2 відображено загальну архітектуру технології WebRTC, яка реалізується всередині веббраузера і забезпечує безпосередню комунікацію між вебдодатками користувачів без потреби в сторонньому програмному забезпеченні. Вона складається з кількох ключових рівнів та компонентів, які забезпечують мультимедійну передачу в реальному часі [3].

У верхній частині схеми показано, що WebRTC взаємодіє з різними вебдодатками через Web API, який редагується W3C та надає інтерфейс

веброзробникам для створення функціональності реального часу. Цей API абстрагує нижчі рівні реалізації і дає змогу створювати peer-to-peer з’єднання між користувачами.

Нижче представлено внутрішню структуру WebRTC, зосереджену у браузері. Основними її компонентами є:

- WebRTC C++ API (PeerConnection) – внутрішній інтерфейс для браузерів, який відповідає за управління сесіями, встановлення з’єднань, передачу медіа та сигналізацію.
- Voice Engine – підсистема для обробки голосу, яка включає кодеки (iSAC, iLBC), компенсацію exo, зменшення шуму та адаптацію до мережевих затримок (NetEQ).
- Video Engine – модуль для обробки відео, що підтримує кодування VP8, буферизацію відеопотоку для зменшення "jitter" та покращення якості зображення.
- Transport – транспортний рівень, який забезпечує безпечну передачу даних за допомогою SRTP, підтримує мультиплексування, пряме з’єднання (P2P), а також використовує протоколи STUN, TURN та ICE для проходження NAT/фаєрволів.
- Audio Capture/Render, Video Capture, Network I/O – нижчі рівні взаємодії з апаратними або системними ресурсами браузера, які відповідають за захоплення, виведення і обмін медіаданими через мережу.

Процес встановлення з’єднання за допомогою WebRTC є складним, але ефективним механізмом, що дозволяє браузерам та іншим клієнтським застосункам встановлювати прямий зв'язок для обміну аудіо, відео та довільними даними в режимі реального часу. Цей процес складається з кількох ключових етапів:

- Ініціація сигналізації: обмін інформацією для встановлення параметрів з’єднання.
- Налагодження з’єднання через ICE, STUN/TURN.
- Створення захищеного каналу передачі (DTLS, SRTP).

- Передача аудіо/відео чи даних напряму між користувачами.

Однак через децентралізовану архітектуру та високу доступність WebRTC стає також потенційно вразливим до різного типу атак, зокрема DoS (відмова в обслуговуванні), що й обумовлює необхідність ретельного дослідження її безпеки.

1.3 Протоколи та механізми взаємодії в технології WebRTC

Технологія WebRTC базується на комплексі мережевих протоколів, які забезпечують обмін мультимедійними даними між учасниками у режимі реального часу. Ці протоколи працюють разом, утворюючи надійну, ефективну та безпечну систему для передачі аудіо, відео та інших типів даних без потреби в серверному посереднику.

На рис. 1.3 представлено архітектуру WebRTC як стек протоколів, що ілюструє, як компоненти взаємодіють на різних мережевих рівнях. Зліва зображено традиційні методи встановлення з'єднання через HTTP/1.x або HTTP/2, поверх захищених сесій TLS і транспортного рівня TCP. Праворуч показано специфічні для WebRTC протоколи: SRTP для передачі медіаданих, SCTP для обміну даними через DataChannel, DTLS для захисту сесій, а також ICE, STUN і TURN для встановлення та підтримки з'єднання. Усі ці протоколи функціонують поверх UDP і базового IP-рівня [4].

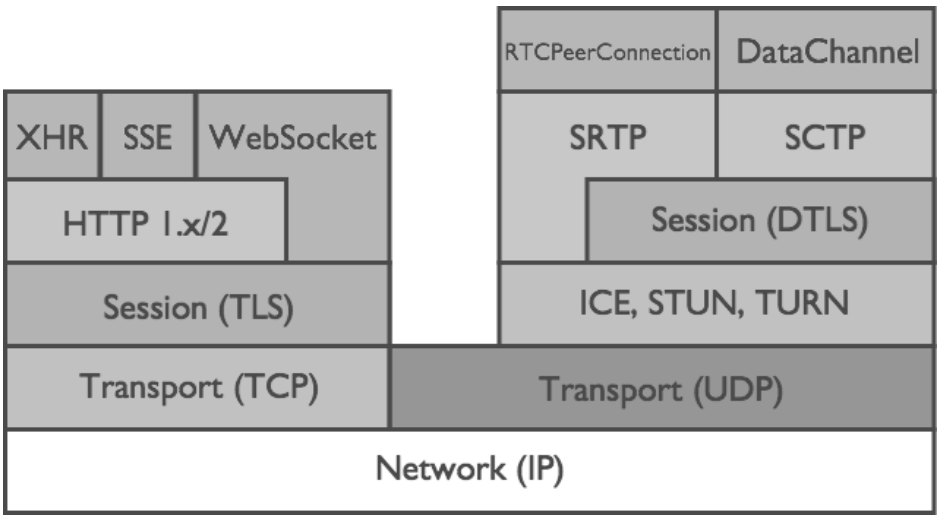

Рисунок 1.3 – Стек протоколів технології WebRTC

Для глибшого розуміння роботи WebRTC важливо розглянути основні протоколи та методи взаємодії, які забезпечують її функціональність. Умовно їх можна поділити на три ключові групи: сигналізація, встановлення з’єднання та передача даних.

1.3.1 Протоколи сигналізації

У контексті технології WebRTC сигналізація відіграє важливу роль на етапі встановлення з’єднання між двома клієнтами. Вона включає обмін службовими повідомленнями, що містять інформацію про ініціалізацію сесії, конфігурацію медіапотоків, параметри мережевого з’єднання та підтримувані кодеки. Попри те, що WebRTC не нав'язує використання конкретного протоколу сигналізації, у більшості випадків застосовуються такі технології, як SIP, XMPP, HTTP(S) або WebSocket [5].

Одним із найуживаніших варіантів є WebSocket – двонаправлений протокол обміну даними в режимі реального часу поверх TCP. WebSocket забезпечує постійне з’єднання між клієнтом і сервером, що ідеально підходить для обміну сигналізаційними повідомленнями (наприклад, SDP-описами, ICE-кандидатами або статусами виклику).

Ще одним поширеним протоколом є SIP – текстовий протокол рівня застосування, розроблений для ініціалізації, модифікації та завершення мультимедійних сесій. Хоча SIP історично пов’язаний з VoIP, його також адаптують для WebRTC у корпоративних середовищах, де вже розгорнуті SIP-інфраструктури. У такому разі WebRTC-клієнти можуть інтегруватися з існуючими SIP-серверами для управління викликами.

XMPP, протокол обміну повідомленнями в реальному часі, також може використовуватись для сигналізації. Його гнучкість і розширюваність дозволяють легко адаптувати його під потреби WebRTC, особливо коли йдеться про інтеграцію в системи миттєвих повідомлень або соціальні платформи.

Важливо зазначити, що сигналізація у WebRTC є зовнішнім процесом, який не стандартизований самим API WebRTC. Розробник повинен самостійно

реалізувати логіку сигналізації, вибравши відповідний протокол або створивши власний механізм. Типовий сигналізаційний процес включає обмін SDP-даними для узгодження медіапараметрів та ICE-кандидатами для встановлення маршруту між клієнтами.

1.3.2 Протоколи встановлення з’єднання

Процес встановлення з’єднання у WebRTC базується на поетапному узгодженні параметрів між клієнтами для забезпечення прямої передачі даних через мережу. Основними протоколами, що забезпечують встановлення з’єднання між вузлами, є ICE, STUN та TURN [6]. У комплексі ці технології вирішують складне завдання – як забезпечити наскрізне з’єднання між клієнтами, які можуть перебувати за NAT або фаєрволом.

Interactive Connectivity Establishment – це фреймворк, який координує взаємодію STUN і TURN-серверів для пошуку найкращого маршруту між клієнтами. ICE виконує процес відомий як candidate gathering, під час якого кожен учасник збирає список можливих IP-адрес і портів, які потенційно можуть бути використані для зв’язку. Далі відбувається перевірка зв’язку між цими кандидатами, і вибирається оптимальний шлях.

Session Traversal Utilities for NAT – легковагий протокол, який дозволяє клієнту дізнатися свою публічну IP-адресу, коли він знаходиться за NAT. STUN використовується для отримання так званих host candidates або server reflexive candidates, які можуть бути використані в з’єднанні, якщо маршрутизація NAT дозволяє це зробити напряму. STUN-сервери зазвичай є публічно доступними та не вимагають високих ресурсів.

У випадках, коли STUN не може забезпечити пряме з’єднання (наприклад, при жорсткому NAT або фаєрволах), використовується TURN. TURN забезпечує ретрансляцію трафіку через спеціальний сервер, який виступає посередником між клієнтами. Це гарантує встановлення з’єднання, хоча і ціною затримок та додаткових навантажень на сервер. TURN-сервери є критично важливими для

забезпечення доступності сервісу в мережевих умовах із сильно обмеженим трафіком.

Загалом, ICE-алгоритм автоматично визначає, які комбінації STUN/TURN-перевірок є можливими, та встановлює з'єднання по найефективнішому маршруту. Цей механізм є надійною основою для динамічного встановлення P2P-з'єднань у реальному часі, забезпечуючи гнучкість WebRTC в умовах різноманітних мережевих сценаріїв.

1.3.3 Протоколи передачі даних

Після успішного встановлення з'єднання між клієнтами, технологія WebRTC переходить до основного етапу – безпосередньої передачі даних у режимі реального часу. Для цього застосовуються спеціалізовані протоколи, що забезпечують низькі затримки, високу якість передачі мультимедійного контенту та захист даних. До основних протоколів цього рівня належать SRTP, RTP, RTCP, а також SCTP.

Real-time Transport Protocol – базовий транспортний протокол, який відповідає за передачу аудіо- та відеопотоків у WebRTC. Його особливість полягає в тому, що він дозволяє синхронізувати мультимедійні потоки, передавати їх з часовими мітками та контрольними номерами, що дає змогу отримувачу правильно відтворити дані навіть за умов втрати деяких пакетів. RTP не гарантує доставку, однак забезпечує оптимізацію для реального часу, що є ключовим у відео- та аудіозв'язку [7].

RTCP функціонує паралельно з RTP і відповідає за контроль якості сервісу. Він передає інформацію про затримки, втрати пакетів та інші характеристики з'єднання, що дозволяє адаптувати потік до поточних умов мережі. RTCP також використовується для синхронізації декількох RTP-потоків, наприклад, аудіо й відео.

Для забезпечення конфіденційності й цілісності даних WebRTC використовує SRTP – розширення до RTP, яке реалізує шифрування, автентифікацію та захист від модифікації даних. Шифрування здійснюється за

допомогою ключів, які були попередньо узгоджені в процесі встановлення з'єднання через DTLS. Таким чином, SRTP дозволяє передавати медіаінформацію захищеним каналом, що критично важливо для захисту від перехоплення або підміни трафіку.

Окрему роль відіграє SCTP – транспортний протокол, що використовується в WebRTC для передачі нестандартних даних, таких як текстові повідомлення, файли або службова інформація. На відміну від RTP, SCTP гарантує доставку, підтримує контроль порядку пакетів і мультиплексування кількох потоків в одному з'єднанні. У контексті WebRTC SCTP передається поверх DTLS, що забезпечує його захищеність.

1.3.4 Взаємодія протоколів

Ефективне функціонування технології WebRTC забезпечується тісною взаємодією між різними мережевими протоколами, кожен з яких виконує окрему роль у встановленні, захисті та підтримці мультимедійного з'єднання в реальному часі. Координація між цими протоколами дозволяє реалізувати безпечний, надійний та гнучкий обмін аудіо-, відео- й іншими даними між клієнтами навіть в умовах складної мережевої інфраструктури, зокрема NAT і файєрволів.

Процес взаємодії починається з протоколів сигналізації, які, хоча й не є частиною специфікації WebRTC, відіграють критичну роль у встановленні ініціального зв'язку між клієнтами. Наприклад, за допомогою SIP або WebSocket передаються SDP-повідомлення, що містять опис параметрів медіасесії, зокрема підтримувані кодеки, IP-адреси, порти та криптографічні ключі.

Далі вступають в дію протоколи встановлення з'єднання. ICE координує взаємодію між STUN і TURN для визначення оптимального маршруту передачі даних між клієнтами. STUN намагається встановити пряме з'єднання, а TURN використовується як резервний варіант ретрансляції через сервер, якщо пряме з'єднання неможливе. Усі ці протоколи функціонують у зв'язці з DTLS, який шифрує сигнальний обмін та забезпечує безпечну генерацію ключів для SRTP.

Передача медіаданих відбувається через SRTP – захищену версію RTP, яка використовує ключі, отримані в ході DTLS-обміну. Дані користувача, не пов’язані з медіа (наприклад, файли чи повідомлення), передаються за допомогою протоколу SCTP, що функціонує поверх DTLS.

Злагоджена взаємодія протоколів WebRTC забезпечує не лише безперебійну передачу даних у реальному часі, а й високий рівень безпеки та стійкості до несприятливих мережевих умов. Така модульна структура дозволяє гнучко адаптувати WebRTC до широкого спектра застосувань – від відеоконференцій до IoT-рішень.

1.4 Переваги та недоліки технології WebRTC

Як і будь-яка інша сучасна інтернет-технологія, WebRTC має комплекс сильних сторін і певні обмеження, які необхідно враховувати при її розробці, впровадженні та використанні. Її застосування у вебдодатках, мобільних сервісах, відеоконференцзв’язку та інших сферах робить її потужним інструментом для організації комунікацій у реальному часі. Проте ефективність і безпека цієї технології напряму залежать від того, наскільки глибоко розуміють її архітектуру, функціональні можливості та обмеження.

Однією з ключових переваг WebRTC є її відкритий код і підтримка більшістю сучасних веб-браузерів, таких як Google Chrome, Mozilla Firefox і Microsoft Edge, що дозволяє розробникам інтегрувати функції реального часу комунікації без необхідності встановлення додаткових плагінів чи програм, спрощуючи процес розробки і покращуючи користувацький досвід, а також забезпечуючи високу якість передачі аудіо та відео завдяки сучасним кодекам, таким як Opus і VP8, що є критично важливим для застосунків, де якість зв’язку відіграє ключову роль, наприклад, у відеоконференціях або телемедицині. WebRTC також економічна, оскільки працює через інтернет, що дозволяє уникнути витрат на традиційну телекомунікаційну інфраструктуру, таку як VoIP-сервіси чи класичні телефонні мережі, роблячи її привабливою для бізнесу, який прагне знизити витрати на комунікації.

З точки зору безпеки, WebRTC пропонує обов’язкове шифрування даних за допомогою протоколів SRTP для медіа-потоків і DTLS для каналів даних, що забезпечує конфіденційність і цілісність переданої інформації, роблячи технологію більш безпечною порівняно з застарілими рішеннями, такими як Flash, які мали відомі вразливості, крім того, WebRTC підтримує інтеграцію з протоколами, такими як SIP, що дозволяє використовувати її в існуючих телекомунікаційних системах, а її peer-to-peer архітектура сприяє децентралізованій комунікації, що покращує приватність і зменшує залежність від центральних серверів.

Попри ці переваги, WebRTC має низку недоліків, які впливають на її ефективність і безпеку, зокрема залежність від якості мережевого з’єднання, що може призводити до затримок, погіршення якості або відключень, особливо в умовах слабкого інтернету, що є критичним для застосунків, де стабільність зв’язку є ключовою, таких як онлайн-навчання чи телемедицина, а з точки зору безпеки, хоча WebRTC має вбудоване шифрування, неналежно захищені сигнальні канали можуть стати вразливими до перехоплення метаданих, що створює ризики для атак, включаючи DoS, які можуть перевантажити сервери чи мережу.

Масштабованість є ще однією проблемою, оскільки WebRTC розроблена для peer-to-peer зв’язку, що ідеально для малих груп, але стикається з труднощами при великій кількості користувачів, наприклад, у вебінарах чи трансляціях, де потрібна додаткова інфраструктура, як SFU (Selective Forwarding Unit) або MCU (Multipoint Control Unit), що збільшує складність і витрати, до того ж WebRTC споживає значні ресурси, такі як пам’ять і процесорний час, особливо на мобільних пристроях, що може призводити до швидкого розряду батареї та високого споживання даних, створюючи незручності для користувачів з обмеженими ресурсами.

Хоча WebRTC підтримується більшістю сучасних браузерів, проблеми сумісності з деякими старішими версіями чи менш поширеними браузерами можуть обмежувати її доступність, а оскільки WebRTC все ще перебуває в

процесі розробки, її API є робочою чернеткою, що може створювати труднощі для розробників через можливі зміни в майбутніх версіях.

1.5 Фреймворки та альтернативи WebRTC

Фреймворки для WebRTC спрощують розробку додатків, надаючи готові інструменти для реалізації функцій реального часу комунікації, таких як відеодзвінки, голосові чати чи передача даних, що зменшує потребу в низькорівневому програмуванні та дозволяє швидше створювати стабільні та безпечні рішення, і серед них OpenVidu, Janus і Jitsi Meet є одними з найпопулярніших завдяки своїй функціональності, гнучкості та доступності.

OpenVidu – це потужний фреймворк, який пропонує розробникам гнучке налаштування відображення відео, включаючи можливість створення власних макетів для учасників відеоконференцій, а також підтримує як хмарне, так і серверне розгортання, що дозволяє адаптувати його до різних інфраструктур, при цьому він забезпечує високий рівень безпеки завдяки вбудованому шифруванню та детальним посібникам, які допомагають розробникам швидко освоїти платформу, а доступний OpenVidu у двох версіях: безкоштовній для базового використання та Pro за ціною $0.0006 за ядро за хвилину, що робить його привабливим для проєктів із різними бюджетами [8].

Janus вирізняється своєю модульною архітектурою та розширюваністю через плагіни, що дозволяють додавати функціонал, як запис відеодзвінків, потокове мовлення чи інтеграція з SIP-протоколами [9]. Також має детальну документацію API, яка спрощує інтеграцію та налаштування, при цьому Janus є повністю безкоштовним і відкритим, що робить його гнучким рішенням для розробників, які прагнуть створювати кастомізовані системи реального часу комунікації, а його підтримка спільнотою та регулярні оновлення забезпечують стабільність і актуальність.

Jitsi Meet, будучи 100% відкритим кодом, пропонує HD-аудіо, інтеграцію з функціями, такими як Etherpad для спільного редагування документів, і можливість блокування кімнат для підвищення безпеки, що робить його

популярним вибором для відеоконференцій, а також Jitsi Meet є безкоштовним і не вимагає створення облікового запису, що спрощує доступ для користувачів. При цьому він підтримує масштабування через розгортання на власних серверах, що дозволяє адаптувати його до потреб великих організацій, а його активна спільнота розробників постійно додає нові функції, такі як покращення якості відео чи інтеграція з календарями [10].

Альтернативи WebRTC стають актуальними, коли потрібні краща масштабованість, простота інтеграції чи специфічні функції, які WebRTC може не забезпечувати через свою складність чи обмеження, наприклад, у великих конференціях чи при потребі в додаткових аналітичних інструментах. Наприклад такі технології, як Twilio Video та Agora.

Twilio Video підтримує до 50 учасників у відеоконференціях, що робить його придатним для середніх груп, і пропонує універсальні SDK для веб, iOS та Android, що забезпечує кросплатформеність, а також надає аналітику дзвінків для моніторингу якості та продуктивності, що є цінним для бізнес-аналітику [11]. Twilio використовує SFU-архітектуру (Selective Forwarding Unit) для ефективного масштабування, але його ціна становить $4 за 1000 хвилин відео, що може бути дорогим для проєктів із великою кількістю дзвінків, однак безкоштовний тестовий період дозволяє протестувати функціонал перед покупкою.

Agora вирізняється низькою затримкою завдяки своїй мережі SD-RTN (Software-Defined Real-Time Network), яка оптимізує маршрутизацію даних, забезпечуючи стабільний голосовий і відеочат, а також підтримку трансляцій для великої аудиторії, що робить її ідеальною для стрімінгових платформ чи вебінарів. Agora пропонує аналітику в реальному часі та інтеграцію з хмарними сервісами, а її ціна починається від $0.99 за 1000 хвилин аудіо, що є економічним рішенням для проєктів із обмеженим бюджетом, хоча для HD-відео та більших груп витрати зростають [12].

Ці фреймворки та альтернативи пропонують різноманітні можливості для реального часу комунікації, але їхній вибір залежить від потреб проєкту, зокрема щодо безпеки, масштабованості та стійкості до атак, таких як DoS, що є ключовим аспектом цього дослідження. Наприклад, фреймворки, як OpenVidu,

можуть полегшити захист від DoS-атак завдяки вбудованим механізмам безпеки, тоді як альтернативи, як Agora чи Twilio, пропонують додаткові інструменти для масштабування, що може зменшити вразливість до перевантажень.

Висновки за розділом 1

У першому розділі було здійснено комплексний аналіз технології WebRTC, яка сьогодні є одним із ключових інструментів для реалізації комунікацій у реальному часі без необхідності встановлення додаткового програмного забезпечення чи плагінів. Розкрито сутність WebRTC як відкритого стандарту, що забезпечує прямий обмін аудіо, відео та даними між пристроями через браузерні інтерфейси, що суттєво спрощує процес розробки інтерактивних вебдодатків.

Проаналізовано архітектуру WebRTC, яка базується на принципах peer-to-peer взаємодії, використанні медіа- та дата-каналів, а також ключових компонентів, таких як ICE, STUN і TURN для встановлення з'єднання. Було розглянуто протоколи і механізми, які забезпечують належну взаємодію та безпеку переданих даних, зокрема SRTP, DTLS та SDP, що відіграють критичну роль у функціонуванні систем реального часу.

Також було детально висвітлено переваги WebRTC – відкритість, кросбраузерна підтримка, висока якість передавання медіа, безпека комунікації та економічність у використанні. Водночас проаналізовано низку обмежень, зокрема проблеми масштабованості, сумісності, споживання ресурсів та потенційні вразливості, які можуть бути використані зловмисниками, зокрема в контексті DoS-атак.

Окрему увагу приділено фреймворкам і альтернативам WebRTC, що дозволяють розширювати її можливості або застосовувати інші рішення в залежності від потреб розробників. Отже, отримані у цьому розділі знання створюють теоретичне підґрунтя для подальшого аналізу ризиків безпеки WebRTC, з акцентом на атаки типу «відмова в обслуговуванні», що буде розглянуто в наступному розділі.

РОЗДІЛ 2
АНАЛІЗ ВРАЗЛИВОСТІ ТЕХНОЛОГІЇ WebRTC ДО DoS-АТАК

2.1 Поняття DoS-атак та їх класифікація

Багато років хакери не цураються всіма доступними методами задля впливу на стабільність роботи інформаційних систем, зокрема тих, що працюють у реальному часі, як-от WebRTC. Мета таких дій може бути різна: від компрометації інформації з обмеженим доступом до повного виведення системи з ладу з метою порушення бізнес-процесів, створення фінансових збитків або дискредитації компанії.

Атаки типу «відмова в обслуговуванні» є одними з найпоширеніших та найнебезпечніших загроз для інформаційно-комунікаційних систем. Їхня мета полягає в тому, щоб зробити інформаційний ресурс – вебсайт, сервер, мережевий сервіс або додаток – недоступним для легітимних користувачів шляхом перевантаження його запитами або створення несприятливих умов для обробки даних. У разі успішного здійснення атаки, система перестає відповідати на запити, працює нестабільно або повністю втрачає працездатність [13].

DoS-атаки можуть реалізовуватися як з одного джерела, так і у вигляді розподілених атак (DDoS), коли велика кількість скомпрометованих пристроїв (ботнет) одночасно надсилає трафік на цільовий сервер. Розподіленість суттєво ускладнює виявлення та фільтрацію атак, оскільки запити надходять із великої кількості легітимних на вигляд джерел [14].

За механізмом реалізації можна виділити три типи DoS-атак [15]:

- Об'ємні атаки. Зловмисник переповнює мережевий рівень тим, що спочатку видається нормальним трафіком. Прикладом такої атаки може слугувати «Посилення DNS-сервера», коли через відкриті DNS-сервери цільовий об’єкт переповнюється трафіком із відповідей на запити DNS. Схему реалізації такої атаки продемонстровано на рис. 2.1.

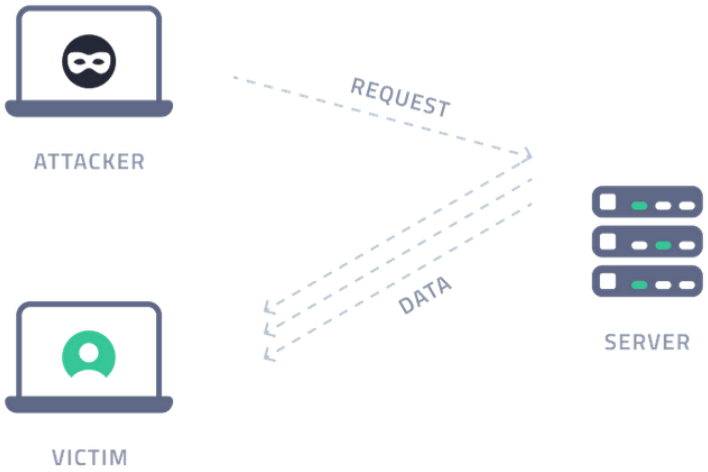

Рисунок 2.1 – Схема DoS-атаки об’ємного типу

- Атаки на протокол. Включають SYN-флуди, схема якої можна переглянути за рис. 2.2, а також атаки фрагментованих пакетів, Ping of Death, Smurf DDoS тощо. Цей тип атаки споживає фактичні ресурси сервера або ресурси проміжного комунікаційного обладнання, такого як брандмауери та балансувальники навантаження, і вимірюється в пакетах за секунду.

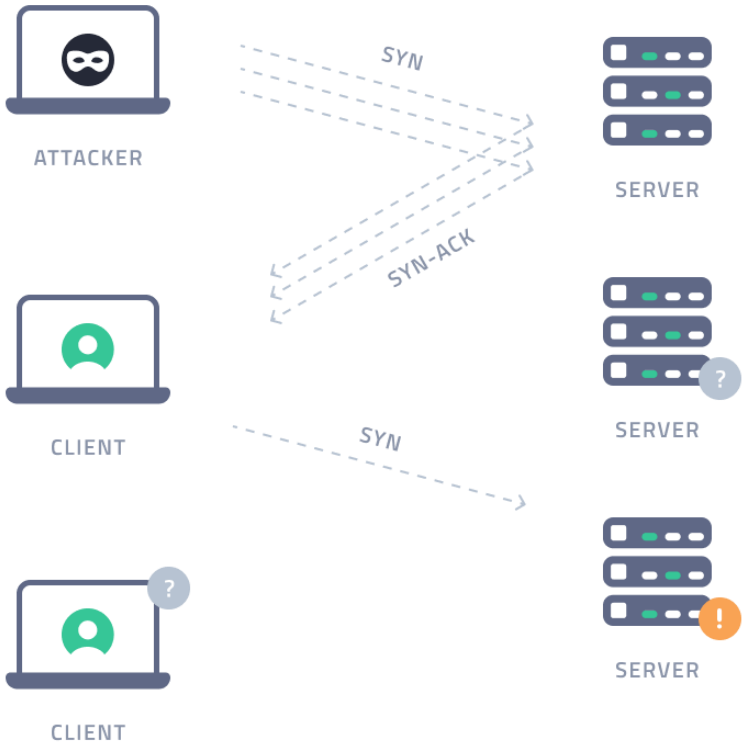

Рисунок 2.2 – Схема атаки SYN-флуд

- Атаки ресурсного (прикладного) рівня. Включають атаки з низькою швидкістю та повільністю, GET/POST-флуди, атаки, спрямовані на вразливості Apache, Windows або OpenBSD тощо. Ці атаки, що складаються з, здавалося б, законних та невинних запитів, спрямовані на збій веб-сервера, а величина вимірюється в запитах за секунду. Схему цього типу атак DoS-атаки можна побачити на рис. 2.3.

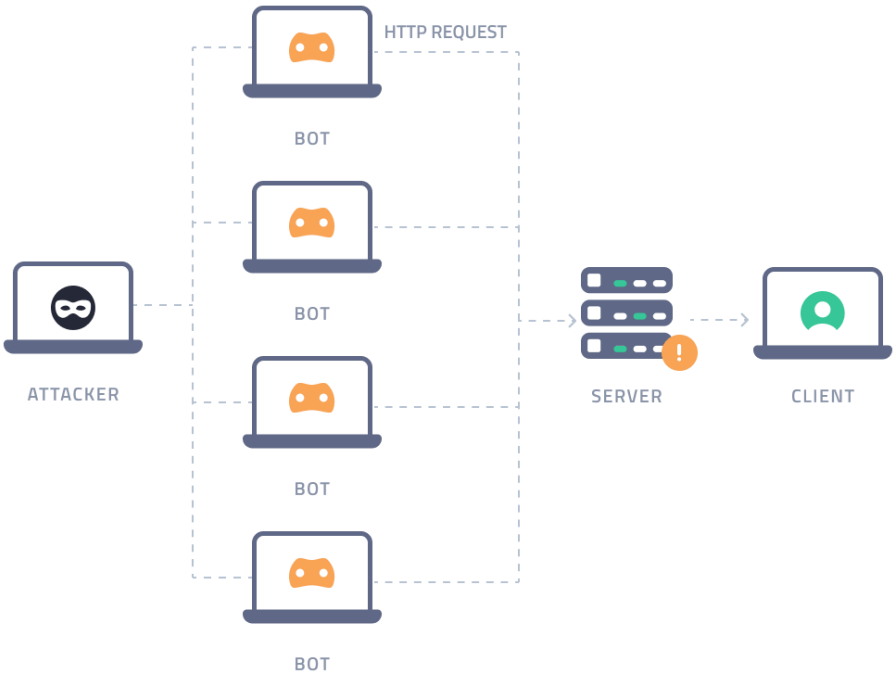

Рисунок 2.3 – Схема атаки HTTP-флуд

Історія DoS-атак бере початок ще з 1970–1980-х років, коли комп'ютерні мережі були недостатньо захищеними, а ресурси – обмеженими. Одним із перших інцидентів, який вказав на вразливість мереж, став черв'як Morpica (1988), що випадково вивів з ладу приблизно 10% тодішніх вузлів ARPANET, перевантаживши їх повторними підключеннями. Ця подія привернула увагу до проблеми надмірного навантаження на системи як способу завдати шкоди.

У 1990-х роках почали з'являтися більш структуровані та спрямовані атаки. Однією з перших стала атака Smurf, при якій зловмисник надсилав підроблені ICMP-пакети на широкомовну адресу мережі з IP-адресою жертви як відправника. У результаті велика кількість вузлів відповідала на ці пакети,

спрямовуючи відповіді до жертви й перевантажуючи її. Іншою поширеною атакою став Ping flood, при якій зловмисник засипав цільовий сервер великою кількістю ICMP-запитів, перевищуючи його здатність відповідати, що призводило до збоїв. Не менш відомий приклад – Ping of Death, де жертві надсилалися неправильно сформовані пакети, що спричиняли збої, перезавантаження або повну зупинку операційної системи.

Зі зростанням складності мереж та збільшенням швидкості інтернет-з'єднань класичні DoS-атаки поступово еволюціонували в DDoS (Distributed Denial of Service) [16], які мають більшу руйнівну силу завдяки використанню багатьох скомпрометованих пристроїв (ботнетів) одночасно. Ці атаки стали масовим явищем на початку 2000-х років, коли такі компанії як Yahoo!, CNN, Amazon та eBay зазнали масових відмов у роботі. З того часу DoS-атаки стали частиною арсеналу кіберзлочинців, активістів та державних акторів.

Саме тому роль DoS-атак в сучасному кіберпросторі важко переоцінити, оскільки вони залишаються одним із найпоширеніших і найефективніших інструментів дестабілізації інформаційних систем. Їх використовують як злочинці з метою шантажу або саботажу, так і державні чи політично мотивовані групи для кібервійн та протестів. Атаки типу DoS здатні тимчасово вивести з ладу вебсайти, сервери, комунікаційні платформи чи сервіси реального часу, що призводить до фінансових збитків, втрати довіри користувачів і порушення надання критичних послуг. У контексті розвитку технологій, таких як WebRTC, які орієнтовані на прямий обмін даними в режимі реального часу, DoS-атаки становлять особливу загрозу через їхню здатність порушити зв'язок, зірвати роботу сервісу чи створити передумови для подальших вторгнень у систему.

2.2 Захисні механізми вбудовані у технологію WebRTC

Безпека є однією з ключових переваг WebRTC, оскільки технологія з самого початку проектувалась із врахуванням вимог конфіденційності, цілісності та автентичності даних, що передаються в реальному часі. Завдяки вбудованим механізмам шифрування та контролю доступу WebRTC забезпечує високий

рівень захисту при передачі аудіо, відео та довільних даних між кінцевими точками зв’язку.

2.2.1 Захист браузеру

Безпека технології WebRTC значною мірою залежить від браузера, який виконує роль довіреного середовища для виконання коду та обробки комунікацій. Саме браузер забезпечує перший рівень захисту користувача, контролюючи доступ до ресурсів, мережевих з’єднань та апаратного забезпечення (камери, мікрофона тощо). Усі скрипти в браузері ізольовані в так званих «пісочницях», що унеможливлює їхню неконтрольовану взаємодію між собою або з системними ресурсами комп’ютера. Такий механізм ізоляції суттєво знижує ризик зловмисного використання WebRTC у шкідливих цілях, зокрема в контексті DoS-атак [4].

Ключову роль у безпеці взаємодії відіграє політика SOP, яка є базовим механізмом безпеки у веббраузерах, що забороняє скриптам, завантаженим з одного джерела, взаємодіяти з ресурсами, завантаженими з іншого джерела. Наприклад, JavaScript-код, запущений зі сторінки https://example.com:443, не зможе отримати доступ до DOM або cookie сторінки https://otherdomain.com або навіть http://example.com. Ця політика забезпечує запобігання атакам типу Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та використанню браузера як ретранслятора запитів у DoS-атаках. Завдяки SOP браузер ізолює виконання скриптів і забезпечує, що жоден компонент WebRTC не зможе створити несанкціоноване з’єднання або передати дані на сторонній сервер без явного дозволу. Що у свою чергу зменшує можливість запуску DoS-атак через браузер користувача на сторонні системи. SOP виконує подвійне завдання – захищає як користувача від потенційних атак, так і сервери від використання браузера як інструмента в розподілених атаках.

Для легітимної міжсайтової взаємодії використовуються спеціальні механізми, як-от CORS та WebSockets, які дозволяють обхід SOP, але лише за умови попередньої верифікації та згоди з боку цільового сервера.

Cross-Origin Resource Sharing – це стандартизований механізм, що дозволяє серверам надавати доступ до своїх ресурсів скриптам, які виконуються в контексті іншого джерела. Він реалізується через HTTP-заголовки, зокрема Access-Control-Allow-Origin, які сервер додає до відповіді. Запити CORS бувають простими і попередніми. У випадку preflight-запиту браузер надсилає OPTIONS-запит, щоб перевірити, чи дозволяє сервер відповідні методи (POST, PUT, тощо), заголовки та походження. Тільки після позитивної відповіді браузер надсилає фактичний запит. Також CORS дозволяє легітимну міждоменну взаємодію в WebRTC, наприклад, під час сигналізації або автентифікації, але при цьому зберігає контроль у руках сервера, що запобігає використанню клієнтського браузера як джерела шкідливого трафіку.

На відміну від звичайних HTTP-запитів, WebSockets відкривають постійне двостороннє TCP-з’єднання між клієнтом і сервером. Хоча початкова ініціалізація WebSocket-з’єднання виконується через HTTP-заголовки, політика SOP не застосовується до цього протоколу в прямому вигляді. Це означає, що браузер дозволяє відкривати WebSocket-з’єднання з будь-яким сервером, незалежно від походження, але сервер має самостійно перевірити заголовки запиту і вирішити, чи дозволяти з’єднання.

В контексті WebRTC, WebSockets часто використовуються для обміну сигналами (наприклад, SDP або ICE-кандидатами). Якщо не контролювати джерело з’єднання на сервері, це може призвести до DoS-атак або витоку інформації. Тому сервери повинні ретельно перевіряти поле Origin та реалізовувати механізми автентифікації перед встановленням WebSocket-сесії.

Таким чином, браузер виступає важливим елементом у архітектурі захисту WebRTC, мінімізуючи ризики зловживання технологією для реалізації DoS-атак. Його механізми, зокрема політика одного джерела (SOP), ізоляція скриптів та контроль доступу до мережевих ресурсів, створюють надійний бар'єр між шкідливим кодом та системними ресурсами. Завдяки цьому браузер не лише обмежує потенційно небезпечні дії, а й активно запобігає їхньому здійсненню в середовищі WebRTC.

2.2.2 Безпека встановлення та оновлення програмного забезпечення

Однією з ключових переваг WebRTC у контексті безпеки є те, що для його використання не потрібно встановлювати додаткові плагіни чи компоненти, як це було характерно для застарілих технологій, таких як Flash або Java-аплети. Це суттєво знижує ризики, пов’язані з поширенням шкідливого програмного забезпечення під виглядом оновлень або інсталяцій, що є типовим вектором атак у традиційних архітектурах.

Технологія WebRTC вбудована безпосередньо в ядро сучасних браузерів – таких як Google Chrome, Mozilla Firefox, Safari та Microsoft Edge – що забезпечує готовність до використання без додаткових налаштувань чи втручання з боку користувача. Завдяки цьому, застосунки на основі WebRTC автоматично користуються всіма механізмами безпеки, реалізованими в самій браузерній платформі.

Ще одним важливим аспектом є оперативність усунення вразливостей. Браузери оновлюються за короткими циклами, а більшість з них підтримують автоматичне оновлення. Це означає, що як тільки виявлено уразливість у компонентах WebRTC, виправлення можуть бути розгорнуті практично миттєво, охоплюючи широку аудиторію користувачів без їхньої прямої участі. Такий підхід дає значну перевагу в порівнянні з традиційним програмним забезпеченням чи апаратними пристроями, оновлення яких часто потребує ручного втручання або має затримку в поширенні патчів.

Для додаткового захисту WebRTC-додатки повинні завжди використовувати HTTPS, що гарантує цілісність і конфіденційність переданих даних під час ініціалізації з’єднання та взаємодії з сигнальними серверами. Це не лише забезпечує шифрування переданих даних, а й дозволяє браузерам блокувати потенційно небезпечний контент, який не відповідає сучасним стандартам безпеки. Таким чином, архітектура WebRTC, заснована на вбудованій інтеграції та автоматичних оновленнях, робить її менш вразливою до атак, пов’язаних з людським фактором та недоліками у керуванні версіями програмного забезпечення.

2.2.3 Захист від несанкціонованого доступу до локальних ресурсів

Один із найважливіших аспектів безпеки WebRTC – це контроль за доступом до локальних ресурсів користувача: камери, мікрофона, а також функцій спільного доступу до екрана. Оскільки WebRTC надає можливість передавати аудіо- та відеодані в режимі реального часу, будь-які вразливості в системі доступу можуть призвести до серйозного порушення конфіденційності та стати вектором потенційних DoS-атак або шпигунського програмного забезпечення. Саме тому захист цих ресурсів є критичним компонентом безпеки всієї екосистеми WebRTC.

Згідно зі специфікацією Media Capture and Streams API, всі браузери, які підтримують WebRTC, реалізують механізм getUserMedia(), що дозволяє запитувати доступ до камери та мікрофона [17]. Цей метод завжди потребує явної згоди користувача: при його виклику браузер виводить діалогове вікно, в якому користувач повинен вручну дозволити або заборонити доступ. Автоматичне або приховане надання доступу виключене за дизайном – цей етап є інтегрованим елементом інтерфейсу браузера, який не може бути обійдений скриптом чи стороннім сайтом.

Реалізація цього механізму має деякі відмінності у популярних браузерах:

- Chrome відображає іконки доступу до камери і мікрофона поруч з адресним рядком, автоматично зупиняє сесію при закритті вкладки, підтримує Permissions Policy для контролю API в iframe, а користувач може керувати дозволами через chrome://settings/content.
- Firefox показує спливаюче вікно для вибору пристрою, інформує про активність медіа через індикатори, дозволяє вручну керувати сесіями, підтримує Permissions Policy і має сторінку about:permissions для управління правами.
- Safari відображає глобальний індикатор активної камери або мікрофона у верхній частині екрану, інтегрується з системними налаштуваннями безпеки macOS, зупиняє медіа-сеанси при втраті фокусу вкладки, а доступ до WebRTC вимагає явної згоди користувача.

- Edge, базований на Chromium, має такі ж механізми, як Chrome: сповіщення при запиті доступу, відображення активних сесій, автоматичне припинення роботи при зміні вкладки, централізоване керування дозволами через edge://settings/content і інтеграцію з безпекою Windows для обмеження доступу на рівні ОС.

Окрім доступу до аудіо- та відеопристроїв, WebRTC також підтримує спільний доступ до екрана за допомогою методу getDisplayMedia(). Ця функція є більш чутливою з погляду безпеки, оскільки дозволяє транслювати весь екран або окреме вікно, що може містити конфіденційні дані. Для цього користувач також має дати явну згоду через системний діалог, який не можна підмінити або стилізувати. В більшості браузерів доступ до екрана дозволяється лише у відповідь на взаємодію з користувачем , а також припиняється, якщо вкладка стає неактивною або змінюється фокус.

Загальна філософія WebRTC щодо доступу до локальних ресурсів базується на принципі user-centric privacy – користувач повинен усвідомлювати, яким саме сайтом і за яких умов його пристрої використовуються. Усі дозволи видаються явно, часто – лише на одну сесію. Браузери також підтримують індикатори активності, дозволяють швидко припинити трансляцію, а також реалізують Permissions Policy для обмеження доступу в контексті iframe.

Разом з тим, існують певні ризики, пов’язані з соціальною інженерією: зловмисні сайти можуть спробувати переконати користувача надати доступ, не розуміючи наслідків. Також можливі DoS-атаки через зловживання ресурсами – наприклад, тривале використання камери або мікрофона може створити навантаження на процесор чи батарею пристрою.

У зв’язку з цим рекомендується дотримуватись додаткових заходів безпеки: WebRTC-додатки повинні перевіряти, чи сторінка запущена у захищеному контексті, застосовувати політики автоматичного завершення сесій при втраті фокусу та за можливості обмежувати функціонал WebRTC лише необхідними компонентами. Також варто регулярно переглядати налаштування дозволів у браузері, особливо для сайтів, які використовують WebRTC-підключення.

Загалом, WebRTC реалізує багаторівневу систему захисту локальних ресурсів користувача, яка поєднує технічні механізми (API, політики безпеки, обмеження на рівні браузера) з прозорим та інтерактивним інтерфейсом користувача. Це дозволяє значною мірою мінімізувати ризики несанкціонованого доступу або потенційного використання WebRTC як каналу для шкідливої активності.

2.2.4 Шифрування медіаконтенту та безпека зв'язку

Існує кілька способів, якими додаток для комунікації в реальному часі може становити загрозу безпеці. Особливо важливою загрозою для технології комунікації в реальному часі є перехоплення незашифрованих медіа- або даних під час їх передачі. Це може статися як у взаємодії браузер–браузер, так і браузер–сервер, коли стороння особа отримує доступ до всього переданого трафіку. Однак шифрування робить практично неможливим для зловмисника розшифрувати вміст комунікаційних потоків. Розшифрувати їх можуть лише учасники, які володіють секретним ключем шифрування.

Саме тому шифрування є обов’язковою складовою WebRTC і застосовується до всіх компонентів, включно з механізмами сигналізації. Таким чином, усі медіа-потоки, що передаються через WebRTC, надійно захищені за допомогою стандартизованих і відомих протоколів шифрування. Тип протоколу залежить від виду каналу: дані шифруються за допомогою Datagram Transport Layer Security, а медіа-потоки – за допомогою Secure Real-time Transport Protocol [18].

Datagram Transport Layer Security – це протокол шифрування, який забезпечує захищений обмін даними поверх ненадійних транспортних протоколів, зокрема UDP, що активно використовується в WebRTC. DTLS є адаптацією широко відомого TLS, але оптимізованою для роботи з дейтаграмними протоколами, де немає гарантій доставки або порядку переданих пакетів [19]. У WebRTC DTLS виконує критичну роль у захисті сигнального

каналу та забезпеченні аутентифікації між учасниками сеансу, а також у створенні ключів для подальшого шифрування медіа за допомогою SRTP.

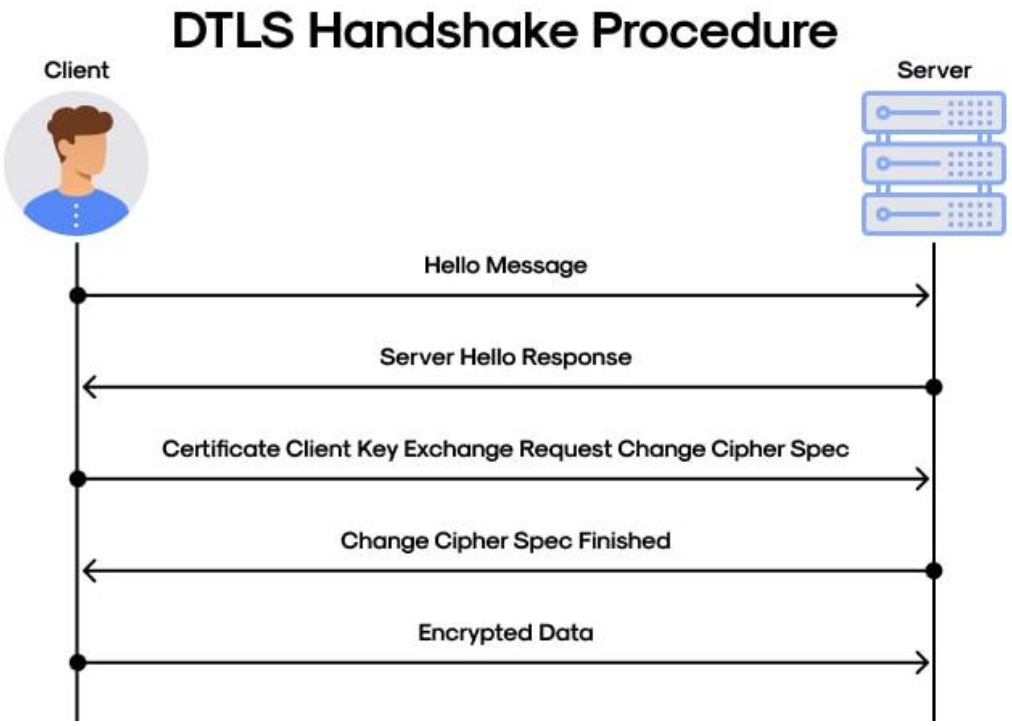

Рисунок 2.4 – Принцип виконання процедури DTLS Handshake

DTLS у WebRTC використовується безпосередньо для встановлення захищеного каналу між учасниками. Під час встановлення з'єднання, коли створюється DTLS handshake, браузери обмінюються сертифікатами X.509, що дозволяє їм взаємно автентифікувати один одного. DTLS handshake, з принципом роботи якого можна ознайомитись за рис. 2.4, також генерує ключі, які далі застосовуються для шифрування SRTP-потоків. DTLS забезпечує конфіденційність, цілісність і автентичність даних, переданих у межах WebRTC-сесії, зберігаючи мінімальні затримки, що критично для реального часу.

DTLS використовується в WebRTC для шифрування peer-to-peer обміну даними, а також для аутентифікації підключень між клієнтами перед початком медіа-стримінгу. Його завдання – забезпечити надійне встановлення зашифрованого каналу через потенційно ненадійне середовище (наприклад, Інтернет).

DTLS відповідає за автентифікацію пірів, конфіденційність переданої інформації та захист від атак типу «людина посередині». Усі дані, що передаються через Data Channels у WebRTC, шифруються за допомогою ключів, згенерованих під час DTLS-handshake, а це означає, що сторонні учасники не мають жодного доступу до їхнього вмісту.

Особливий випадок роботи DTLS у WebRTC виникає тоді, коли між користувачами неможливо встановити пряме пірингове з'єднання, і дані маршрутизуються через реле-сервер типу TURN. У таких сценаріях DTLS може функціонувати «поверх» TURN, тобто встановлення DTLS-з'єднання відбувається не напряму між користувачами, а через посередника – TURN-сервер. При цьому зберігається наскрізне шифрування: DTLS все ще використовується для захисту даних між клієнтами, але передача відбувається через релейний сервер, який не має доступу до ключів шифрування. Таким чином, навіть у випадку використання TURN, WebRTC гарантує високий рівень безпеки, не допускаючи перехоплення чи модифікації трафіку з боку посередника.

Secure Real-time Transport Protocol – це розширення стандартного протоколу RTP, яке забезпечує шифрування, автентифікацію та перевірку цілісності медіа-потоків в реальному часі [20]. Різниця між протоколами SRTP та RTP продемонстрована на рис. 2.5. Він був спеціально розроблений для захисту комунікацій у VoIP, відеоконференціях та інших системах потокового передавання, зокрема у WebRTC. SRTP є легким і ефективним з погляду обчислювальних ресурсів, що робить його придатним для сценаріїв з низькою затримкою.

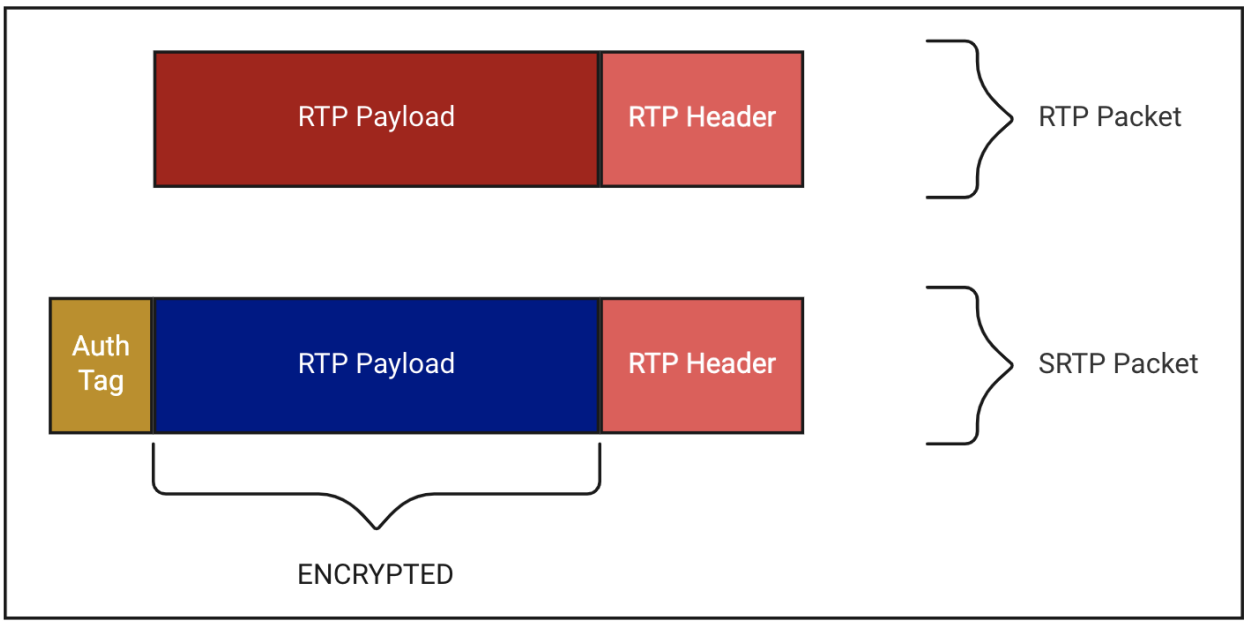

Рисунок 2.5 – Різниця між протоколами SRTP та RTP

У WebRTC SRTP забезпечує повну конфіденційність медіа-трафіку між піринговими клієнтами. Ключі для шифрування SRTP-потоків не передаються відкрито, як і було вище зазначено – вони генеруються та обмінюються під час DTLS-handshake у процесі встановлення з'єднання. Таким чином, SRTP не має вбудованого власного механізму обміну ключами, що виключає можливість їх перехоплення.

SRTP підтримує використання декількох алгоритмів шифрування, найпоширенішим з яких є AES (Advanced Encryption Standard) у режимі CTR (Counter Mode) з довжиною ключа 128 біт. Для автентифікації та виявлення змін у медіапотоках використовується HMAC-SHA1 (зазвичай з тегом довжиною 80 або 32 біти). Цей механізм дозволяє виявляти несанкціоновану зміну чи підробку RTP-пакетів у реальному часі.

Оскільки SRTP працює на прикладному рівні поверх UDP, він не залежить від транспортного з’єднання і зберігає стійкість до втрат пакетів або їхньої зміни в мережі. Навіть у випадках використання TURN-серверів, які ретранслюють зашифровані RTP-пакети, SRTP зберігає цілісність – ретранслятор не має доступу до розшифрованих медіа, адже не бере участі в процесі обміну ключами DTLS-SRTP.

Попри надійне шифрування вмісту медіаданих, SRTP має важливе обмеження: він шифрує лише корисне навантаження RTP-пакетів, але залишає їх заголовки у відкритому вигляді. Це створює потенційну вразливість, адже в заголовках міститься низка метаданих, які можуть мати чутливий характер. Зокрема, до таких даних належить інформація про рівень гучності аудіо (audio levels), що дає змогу сторонньому спостерігачу визначити, чи говорить користувач у певний момент часу. Хоча сам зміст розмови лишається недоступним для зловмисника, факт наявності або відсутності мовлення може бути використаний для непрямого аналізу комунікаційної активності. Наприклад, це дозволяє правоохоронцям або атакуючим визначити, чи контактує ціль з певною особою, навіть не знаючи змісту розмови, що ставить під загрозу конфіденційність на метаданому рівні.

2.2.5 Процес автентифікації та ідентифікації

WebRTC дозволяє браузерам безпосередньо встановлювати P2P-з’єднання, минаючи серверну інфраструктуру після встановлення зв’язку. Тому автентифікація співрозмовника повинна виконуватись незалежно від сигналізаційного сервера, який може бути скомпрометованим або просто недовіреним. Для цього WebRTC використовує Identity Provider (IdP) API, який дозволяє інтегрувати зовнішніх постачальників ідентичності (наприклад, Google, Microsoft, Facebook, GitHub).

У процесі ініціалізації, браузер генерує cryptographic assertion (затвердження) – це підписаний токен, створений IdP, який засвідчує особу користувача. Токен передається іншому учаснику WebRTC-сесії, який може перевірити його справжність за допомогою відкритого ключа IdP. Таким чином, особу користувача перевіряє не сервер додатку, а авторитетний IdP, що мінімізує ризик фальсифікації.

До ключових етапів автентифікації можна віднести:

- Генерація виклику, де RTCPeerConnection.setIdentityProvider() ініціює процес IdP-автентифікації.

- Запит токена, де браузер надсилає запит до IdP через iframe-секцію, що ізольована SOP.
- Створення assertion, де IdP повертає assertion, зазвичай у форматі JSON Web Token (JWT), підписаний приватним ключем IdP.
- Валідація токена, де отримувач сесії WebRTC може перевірити токен через JavaScript або сервер, використовуючи відкритий ключ IdP.

Інфраструктура IdP у WebRTC дозволяє реалізувати end-to-end автентифікацію, де довіра будується не на сервері сигналізації, а на перевірених сторонніх провайдерах, які виконують ідентифікацію користувача через криптографічні підписи. У випадку правильної реалізації це значно знижує ризик атак типу man-in-the-middle та виключає можливість того, що зловмисник видасть себе за легітимного користувача.

2.2.6 Конфіденційність IP-адреси

Використання Interactive Connectivity Establishment у WebRTC може призвести до розкриття IP-адреси користувача, що потенційно дозволяє зловмисникам визначити його приблизне місцезнаходження. Під час процесу ICE-узгодження WebRTC обмінюється кандидатами, які включають локальні та публічні IP-адреси, що може бути використано для ідентифікації користувача, особливо якщо зловмисний веб-сайт або інший учасник зв’язку має намір зібрати цю інформацію [21]. Хоча WebRTC не призначений для захисту від веб-сайтів, які навмисно намагаються отримати IP-адресу, він пропонує кілька механізмів для забезпечення конфіденційності IP-адреси від іншого співрозмовника, що є важливим для захисту приватності користувачів.

Одним із ключових механізмів є можливість для реалізації WebRTC дозволяти JavaScript-скриптам призупиняти ICE-узгодження до моменту, коли користувач явно прийме виклик. Цей підхід запобігає передчасному обміну кандидатами, що містять IP-адресу, і приховує онлайн-статус користувача. Завдяки цьому зловмисник не може отримати інформацію про IP-адресу або

статус підключення, доки користувач не підтвердить свою готовність до зв’язку, що значно підвищує рівень конфіденційності.

Крім того, WebRTC дозволяє додаткам налаштувати використання виключно TURN кандидатів. TURN-сервери діють як посередники, ретранслюючи трафік між учасниками зв’язку, що повністю приховує реальну IP-адресу користувача від співрозмовника. Це особливо корисно в сценаріях, де приватність є пріоритетом, оскільки весь трафік проходить через релейний сервер, а локальні або публічні IP-адреси не розкриваються.

Додатково, WebRTC підтримує механізм, який дозволяє додавати не-TURN кандидати до вже встановленого дзвінка. Цей підхід дає змогу розпочати дзвінок із використанням TURN-кандидатів для забезпечення конфіденційності, а після підтвердження з’єднання додати прямі кандидати для зменшення затримки передачі даних. Така гнучкість дозволяє оптимізувати продуктивність без компрометації безпеки, оскільки IP-адреса розкривається лише після того, як користувач погодився на зв’язок.

Ці механізми забезпечують гнучкість у балансуванні між конфіденційністю та продуктивністю WebRTC-додатків. Однак їх ефективність залежить від правильної реалізації та конфігурації як на стороні клієнта, так і на стороні сервера, а також від свідомого вибору користувачем налаштувань приватності.

2.2.7 Безпека рівня сигналізації

Рівень сигналізації в WebRTC є критично важливим для встановлення з’єднань між учасниками, забезпечуючи обмін метаданими, такими як SDP і ICE-кандидати. Одним із найпоширеніших протоколів для реалізації сигналізації в WebRTC є Session Initiation Protocol, який широко використовується в телекомунікаційних системах для ініціації, підтримки та завершення мультимедійних сесій. Однак SIP, як і інші протоколи сигналізації, вразливий до атак, таких як перехоплення, підробка повідомлень і DoS-атаки, що можуть порушити конфіденційність, цілісність або доступність WebRTC-сесій. Для

забезпечення безпеки рівня сигналізації необхідно застосовувати сучасні методи шифрування, аутентифікації та захисту від атак.

SIP, як основний протокол сигналізації, передає повідомлення, такі як INVITE, REGISTER і BYE, які можуть бути перехоплені зловмисниками для отримання чутливих даних, наприклад, IP-адрес або параметрів сесії. Типові загрози включають атаки типу «людина посередині» (MITM) [22], де зловмисник модифікує SDP-дані, щоб перенаправити медіа-потоки, а також атаки захоплення реєстрації (SIP registration hijacking), коли зловмисник підробляє повідомлення REGISTER для перенаправлення викликів. Крім того, DoS-атаки на SIP-сервери, такі як надсилання великої кількості фальшивих INVITE-запитів, можуть перевантажити сервер і порушити доступність сервісу. Для захисту від цих загроз необхідно використовувати Transport Layer Security для шифрування SIP-повідомлень, що унеможливлює їх перехоплення та маніпуляції. Наприклад, використання SIPS (SIP over TLS) забезпечує конфіденційність і цілісність даних, як рекомендовано в Digital Samba.

Окрім SIP, для сигналізації в WebRTC можуть використовуватися інші протоколи, такі як WebSocket (з HTTPS для безпеки) або XMPP. WebSocket є популярним завдяки своїй простоті та підтримці в браузерах, але потребує HTTPS для захисту від перехоплення. XMPP, хоча менш поширений у WebRTC, використовується в системах із розподіленим обміном повідомлень і також вимагає TLS для безпеки. Для всіх цих протоколів важливо впроваджувати надійні механізми аутентифікації, такі як JSON Web Tokens (JWT) або OAuth 2.0, щоб верифікувати учасників і запобігти несанкціонованому доступу до сигналізаційних каналів. Наприклад, SIP-сервер може вимагати аутентифікацію через Digest Authentication або токени для підтвердження легітимності клієнтів.

Для захисту від DoS-атак на сигналізаційні канали, які є особливо актуальними для SIP через його широке використання, рекомендується впроваджувати обмеження швидкості запитів (rate limiting) і фільтрацію вхідного трафіку на основі білого списку IP-адрес. Додатково, використання механізмів, таких як CAPTCHA або тимчасові токени, може зменшити ризик автоматизованих атак, спрямованих на перевантаження серверів. Наприклад,

SIP-сервери можуть блокувати надмірні INVITE-запити від неавторизованих джерел, використовуючи системи виявлення аномалій. Регулярне оновлення програмного забезпечення, такого як Asterisk або Kamailio, які часто використовуються для обробки SIP, а також дотримання стандартів, наприклад, RFC 3261 для SIP і RFC 8827 для WebRTC, є критично важливим для усунення відомих вразливостей і забезпечення безпеки сигналізаційних каналів.

2.3 Вектори DoS-атак у WebRTC-середовищі

Архітектура технології Web Real-Time Communication, яка включає сигнальні сервери, TURN-сервери та мережеві з'єднання, створює численні вектори для атак типу DoS, спрямованих на перевантаження ресурсів і порушення доступності сервісу. Ці атаки можуть суттєво вплинути на стабільність WebRTC-додатків, таких як відеоконференції чи онлайн-ігри, і тому важливо детально проаналізувати кожен потенційний вектор, щоб зрозуміти, як вони можуть бути використані зловмисниками та як мінімізувати їхній вплив, що є ключовим аспектом цього дослідження [23].

Одним із основних векторів DoS-атак є сигнальний сервер, який відіграє центральну роль у координації обміну метаданими між peer'ами, такими як SDP offers, answers та ICE candidates, необхідними для встановлення з'єднання. Атакуючий може скористатися цією вразливістю, надсилаючи масову кількість фальшивих запитів, що призводить до перевантаження сервера і блокування легітимних користувачів від доступу до сервісу. Якщо сигнальний канал не захищений належним чином, наприклад, через використання незашифрованих протоколів замість HTTPS чи WSS (WebSocket Secure), це полегшує перехоплення або підробку метаданих, таких як SDP чи ICE candidates, що може посилити ефект атаки, дозволяючи зловмиснику маніпулювати процесом зв'язку або змусити сервер обробляти нерелевантний трафік.

Інший значний вектор пов'язаний із TURN-серверами, які використовуються для маршрутизації трафіку, коли пряме peer-to-peer з'єднання неможливе через мережеві обмеження, такі як NAT чи брандмауери. Атакуючий

може зловживати цими серверами, надсилаючи надмірну кількість запитів або фальшивих з’єднань, що виснажує їхні обчислювальні ресурси, такі як пропускну здатність чи пам’ять, і робить сервіс недоступним для легітимних користувачів. Без належної аутентифікації, наприклад, через токени доступу з обмеженим терміном дії чи міцні паролі, TURN-сервери стають особливо вразливими, оскільки зловмисник може легко отримати доступ і використовувати їх як платформу для атаки, спрямовуючи трафік на інші цілі або перевантажуючи систему.

Перевантаження мережі є ще одним критичним вектором DoS-атак у WebRTC-середовищі, оскільки технологія залежить від стабільного мережевого з’єднання для передачі медіа-потоків і даних у реальному часі. Атакуючий може генерувати величезну кількість пакетів, наприклад, через фальшиві ICE candidates або масові запити до STUN-серверів (Session Traversal Utilities for NAT), що призводить до заторів у мережі, переривання з’єднань між peer’ами та деградації якості сервісу. Цей вектор особливо небезпечний у peer-to-peer архітектурі WebRTC, де кожна кінцева точка може стати мішенню, а атака на одну точку може вплинути на всю групу учасників, наприклад, під час відеоконференції, коли масовий трафік може змусити всіх учасників втратити зв’язок.

Нарешті, варто зазначити, що вразливості можуть виникати через недостатню безпеку реалізації додатків на основі WebRTC, наприклад, коли розробники не обмежують кількість одночасних з’єднань чи не впроваджують моніторинг аномалій у трафіку. Такі слабкі місця можуть бути використані для атаки типу Distributed DoS, коли кілька джерел одночасно надсилають запити, посилюючи ефект перевантаження. Хоча WebRTC сама по собі не визначає конкретних захисних механізмів проти таких атак, її залежність від зовнішніх компонентів, таких як сигнальні та TURN-сервери, робить аналіз цих векторів важливим для розробки ефективних стратегій захисту.

Загалом, вектори DoS-атак у WebRTC-середовищі включають сигнальні сервери, TURN-сервери та перевантаження мережі, кожен із яких створює унікальні виклики для стабільності системи. Розуміння цих вразливостей є

основою для подальшого аналізу вразливих компонентів і розробки захисних механізмів, які будуть детально розглянуті в наступних розділах роботи.

2.4 Методи та інструменти реалізації DoS-атак на WebRTC

WebRTC-сервіси залишаються вразливими до DoS-атак, попри вбудовані засоби безпеки, через відкриту природу архітектури та підтримку роботи через NAT. Однією з поширених цілей зловмисників є TURN-сервери, які забезпечують ретрансляцію трафіку в умовах недоступності прямого P2P-з’єднання. У рамках атаки типу allocation flooding зловмисники створюють сотні або тисячі псевдо-сеансів через ботнети або headless-браузери, які ініціюють WebRTC-з'єднання, надсилаючи DTLS-запити або RTP-пакети через TURN. Кожен такий запит потребує обробки та буферизації, навіть якщо не несе корисного навантаження, що вичерпує ресурси TURN-сервера та може призвести до його недоступності для легітимних користувачів.

Ще одним вектором атаки є зловживання протоколом ICE, який використовується для встановлення найкращого маршруту для медіа-трафіку. Під час фази обміну кандидатами (ICE candidate exchange) браузери надсилають STUN-запити для перевірки можливих шляхів доставки. Зловмисник може автоматично згенерувати велику кількість підроблених ICE-кандидатів, які вказують на неіснуючі або навмисно перевантажені вузли. Це створює зайве навантаження на сигналізаційний сервер і TURN-інфраструктуру, затримує або повністю блокує встановлення легітимних з'єднань.

Крім цього, зловмисники можуть проводити media flood-атаки, засновані на масовій генерації SRTP-потоків із фальшивими RTP-пакетами. Хоча ці пакети не можуть бути розшифровані через відсутність правильного ключа, браузери змушені їх обробляти на рівні транспортного стеку. У випадку поганої реалізації обробника медіа це призводить до перевантаження CPU, витоку пам’яті або навіть краху браузерної вкладки. Такий підхід, відомий як RTP injection, особливо небезпечний у випадках використання вразливих мобільних браузерів або IoT-пристроїв.

Сигналізаційна частина WebRTC також вразлива до перевантаження. Наприклад, якщо сигналізація побудована на WebSocket або HTTP polling без захисту від частих запитів (rate limiting), зловмисник може надіслати тисячі запитів на ініціалізацію з'єднання, реєстрацію ICE-кандидатів або повторний запуск peer-з’єднань. Це викликає перевантаження черг повідомлень, витоки пам’яті та падіння серверної логіки. У більшості випадків такі атаки реалізуються за допомогою автоматизованих браузерів (такі як Puppeteer, Playwright, Selenium), які одночасно відкривають десятки або сотні WebRTC-сесій із фальшивими користувачами.

Окрему категорію становлять reflection-атаки на основі STUN. Оскільки STUN-відповіді повертаються на порт клієнта, який ініціював запит, зловмисник може використати браузер жертви як ретранслятор, створюючи STUN-запити зі зміненими IP-адресами одержувача. У результаті третя сторона отримує великий обсяг неочікуваного трафіку, що призводить до перевантаження або мережевого шуму. Такі атаки можуть використовуватись як складова частина масштабніших DDoS-кампаній із маскуванням джерела атаки.

Для реалізації DoS-атак на WebRTC зловмисники активно застосовують різноманітні спеціалізовані інструменти та фреймворки, що дозволяють автоматизувати створення великої кількості підключень, інжекцію шкідливого трафіку або перевантаження TURN/STUN-серверів.

Одним із найпоширеніших підходів є використання кастомних скриптів Node.js, часто зібраних у ботнет-подібні архітектури під загальною назвою wrtc-botnet. Такі скрипти працюють на базі бібліотек типу wrtc – нативного WebRTC-біндінгу для Node.js, який дозволяє створювати peer-з’єднання без використання реального браузера. Скрипти масово ініціюють WebRTC-з'єднання з підробленими SDP-даними, генерують тисячі TURN allocation-запитів до сервера, використовуючи різні облікові дані та IP-спуфінг. Основна мета – вичерпати пул доступних relay-портів, перевантажити буфери обробки медіа та змусити сервер витрачати ресурси на обробку порожнього або навмисно пошкодженого трафіку.

Іншим потужним засобом є SIPp у зв'язці з WebRTC-проксі. SIPp – це інструмент для генерації трафіку на основі SIP-протоколу, який, хоча і не використовується напряму в WebRTC, може слугувати для імітації сигналізаційних повідомлень у WebRTC-додатках, де SIP реалізований через WebSocket або HTTP як частина кастомного Signaling layer. Завдяки проксі, що транслює SIP у SDP/ICE-формат, атакувальник може масово надсилати неправдиві запити на початок виклику, ICE-кандидати та STUN-запити, що призводить до перевантаження сигналізаційної логіки серверів або підвищеного навантаження на механізми NAT traversal.

Mitmproxy у комбінації з WebRTC API використовується для атаки більш високого рівня – інжекції медіа-потоків у вже встановлені WebRTC-сесії. Через перехоплення трафіку, зловмисник модифікує або вставляє шкідливі RTP-пакети безпосередньо у трафік SRTP. Уразливість полягає в тому, що SRTP шифрує лише payload, а заголовок RTP залишається відкритим. Це дозволяє здійснювати ін'єкцію добре сформованих, але фальшивих RTP-пакетів, які можуть впливати на декодер або створювати фонове навантаження на процесор жертви. Особливо критичною є така атака у випадку мобільних пристроїв, де обробка медіа має обмежені ресурси.

Не менш ефективними є засоби автоматизації браузера, такі як Puppeteer або Playwright. Ці інструменти дозволяють створювати сценарії, які одночасно запускають десятки або сотні вкладок браузера, кожна з яких ініціює окрему WebRTC-сесію. Через getUserMedia() та RTCPeerConnection можна запускати реальні або псевдо-медіа потоки, підключатися до TURN-серверів та передавати трафік на серверну інфраструктуру. У разі паралельної генерації фальшивих ICE-кандидатів або циклічного відкриття-закриття з'єднань (connection churn), атакувальник може призвести до краху сигналізаційного сервера або перенавантаження TURN/ICE-механізму на клієнтській стороні.

Комбінація вищезгаданих інструментів дозволяє будувати багатошарові DoS-атаки на WebRTC-сервіси, які спрямовані не лише на обчислювальні ресурси серверів, а й на мережеву інфраструктуру, канал зв’язку та навіть вразливості у клієнтських реалізаціях WebRTC у браузерах. З огляду на це, для

запобігання таким атакам необхідно впроваджувати захист на всіх рівнях: від обмеження числа peer-з’єднань і rate-limiting TURN-запитів до ретельної фільтрації RTP-трафіку і контролю доступу до сигналізаційних API.

2.5 Сучасні підходи до запобігання DoS-атакам у WebRTC

У відповідь на зростання складності та частоти DoS-атак на WebRTC-інфраструктуру, сучасні дослідження та розробки зосереджені на інноваційних методах захисту, які виходять за межі традиційних підходів, таких як обмеження трафіку чи CAPTCHA. Новітні рішення включають використання штучного інтелекту, квантово-стійкої криптографії, спеціалізованих honeypot-систем, Proof-of-Work механізмів, eBPF-фільтрації та апаратних рішень. Ці технології спрямовані на підвищення стійкості WebRTC до атак, забезпечуючи надійність і безпеку реального часу спілкування.

Одним із найперспективніших напрямів є використання штучного інтелекту та машинного навчання (ML) для виявлення аномалій у мережевому трафіку WebRTC. Алгоритми ML, включаючи класичні класифікатори, такі як дерева рішень і Support Vector Machines (SVM), а також глибокі нейронні мережі, здатні ідентифікувати складні патерни шкідливої поведінки в протоколах RTP, DTLS і STUN. Наприклад, моделі можуть виявляти аномально високу частоту SDP-обмінів, характерну для флуд-атак на сигналізаційний рівень, або надмірну кількість TURN allocation-запитів, що вказує на спроби виснаження ресурсів сервера. Алгоритми без нагляду, такі як автоенкодери чи методи кластеризації (DBSCAN, k-means), дозволяють детектувати zero-day атаки без попередньо відомих сигнатур. У деяких реалізаціях ML-моделі інтегруються в TURN-сервери, такі як coturn, для динамічного блокування аномального трафіку в реальному часі, забезпечуючи швидку реакцію на загрози.

Ще одним інноваційним інструментом є WebRTC-aware honeypot-системи, які симулюють повноцінну WebRTC-інфраструктуру, включаючи сигналізаційні інтерфейси (WebSocket, SIP) і TURN/STUN-сервери. Такі системи приймають ICE-кандидати, обробляють SDP-відповіді, але не маршрутизують реальні дані,

дозволяючи фіксувати поведінку зловмисників. Наприклад, honeypot може виявити нетипові STUN-запити з однаковими transaction ID від різних IP-адрес, що вказує на атаку. Зібрані дані використовуються для тренування систем виявлення вторгнень (IDS) або для створення динамічних списків контролю доступу (ACL), які автоматично застосовуються до продуктивних серверів. Це дозволяє адаптивно реагувати на нові типи атак і вдосконалювати захист WebRTC-систем.

Впровадження квантово-стійкої криптографії в WebRTC-протоколи є ще одним передовим підходом, який має непрямий вплив на захист від DoS-атак. Хоча основна мета таких технологій – забезпечення конфіденційності, вони ускладнюють replay-атаки завдяки використанню постквантових алгоритмів, таких як CRYSTALS-Kyber, SABER або NewHope. Експериментальні реалізації, наприклад, у бібліотеці pqwebrtc, поєднують традиційний обмін ключами ECDHE з постквантовими алгоритмами, створюючи гібридний handshake-механізм. Це забезпечує захист від компрометації трафіку навіть у разі появи квантових комп’ютерів, що особливо важливо для довготривалих peer-to-peer з’єднань, уразливих до повторних DoS-атак із використанням старих сесійних параметрів.

Ефективним методом протидії DoS-атакам є впровадження Proof-of-Work (PoW) механізмів перед ініціалізацією WebRTC-сесій. Клієнт, який бажає підключитися до TURN-сервера або встановити з’єднання, повинен вирішити обчислювально складну задачу, наприклад, знайти хеш SHA256 із заданою кількістю початкових нулів. Це ускладнює масові атаки ботнетів, оскільки кожен запит потребує значних обчислювальних ресурсів, але легітимні користувачі з помірною частотою запитів не відчувають затримок. Проєкт ProofRTC, наприклад, інтегрує PoW у ICE-агенти, додаючи nonce-фільтрацію, що значно знижує ризик флуд-атак на STUN/TURN-запити.

Використання eBPF-фільтрації (Extended Berkeley Packet Filter) є ще одним потужним інструментом для захисту WebRTC-трафіку. eBPF дозволяє виконувати програми в ядрі Linux для аналізу та фільтрації пакетів у реальному часі, минаючи userspace. У WebRTC це дає змогу блокувати RTP-, STUN- або

DTLS-пакети з ознаками DoS-атак, наприклад, надмірну частоту STUN-запитів або однакові transaction ID. eBPF також зберігає статистику з'єднань, дозволяючи блокувати джерела аномального трафіку. У поєднанні з даними від ML-моделей або honeypot-систем, eBPF забезпечує швидке і точне реагування на загрози.

У високонавантажених середовищах дедалі частіше застосовуються апаратні рішення, такі як Intel SmartNIC або Mellanox ConnectX-6, які підтримують Deep Packet Inspection (DPI) і rate-limiting. Ці пристрої здатні аналізувати WebRTC-трафік, виявляючи підозрілі DTLS handshakes, STUN binding requests або RTP marker bits, і блокувати аномальні з'єднання на фізичному рівні. Завдяки FPGA або ASIC, вони обробляють мільйони пакетів на секунду, забезпечуючи захист від масових розподілених DoS-атак без впливу на продуктивність.

Сучасні підходи до запобігання DoS-атакам у WebRTC формують багаторівневу архітектуру захисту, що поєднує ШІ, квантово-стійку криптографію, honeypot-системи, PoW-механізми, eBPF-фільтрацію та апаратні рішення. Ці технології дозволяють ефективно виявляти та нейтралізувати атаки на сигналізаційному та медіа-рівнях, забезпечуючи надійність і безпеку WebRTC-додатків. Впровадження таких методів вимагає ретельної інтеграції та конфігурації, але їх комбінація створює стійку систему захисту, здатну протистояти сучасним і майбутнім загрозам.

Висновки за розділом 2

У другому розділі було проведено детальний аналіз вразливостей технології WebRTC до атак типу «відмова в обслуговуванні», а також охарактеризовано відповідні захисні механізми, що вбудовані в архітектуру цієї технології. На основі розглянутої класифікації DoS-атак встановлено, що WebRTC-середовище є потенційною мішенню як для традиційних, так і специфічних векторів атак, зокрема через STUN/TURN-сервери, сигнальні канали, обробку медіа та обхід обмежень доступу до системних ресурсів.

Було проаналізовано, як сучасні браузери реалізують базовий захист від DoS-спроб: обмеження частоти запитів, контроль доступу до пристроїв, реалізація політик безпеки під час оновлень. Значну увагу приділено криптографічним протоколам – DTLS і SRTP – які забезпечують шифрування трафіку, але водночас мають певні недоліки, зокрема можливість отримання метаінформації із заголовків RTP-пакетів. Також описано механізми автентифікації за допомогою Identity Provider, що дозволяють перевіряти особу співрозмовника незалежно від серверів сигналізації.

У розділі наведено технічний розгляд ключових векторів DoS-атак у WebRTC, серед яких: перевантаження STUN/TURN-серверів, атаки через ICE-агенти, флудинг SDP-повідомленнями, використання replay або malformed-пакетів для дестабілізації зв’язку. Показано, як зловмисники можуть здійснювати такі атаки за допомогою популярних інструментів, зокрема автоматизованих скриптів для генерації трафіку, кастомізованих клієнтів WebRTC та botnet-мереж.

Також було розглянуто найновіші підходи до запобігання DoS-атакам у WebRTC. Зокрема, проаналізовано використання Proof-of-Work як бар’єру перед встановленням з’єднання, застосування eBPF-програм для фільтрації трафіку на рівні ядра ОС, а також інтеграцію апаратних засобів DPI та rate-limiting на базі SmartNIC. Акцент зроблено на потенціал використання машинного навчання для виявлення аномального трафіку та динамічного коригування правил фільтрації.

Незважаючи на наявність серйозних вбудованих механізмів безпеки, WebRTC залишається чутливим до DoS-атак. Для забезпечення надійного захисту необхідно поєднувати стандартні засоби протоколу із сучасними адаптивними підходами, що враховують як технічну специфіку WebRTC, так і постійно еволюціонуючі методи атак.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ВИЯВЛЕННЯ ВТОРГНЕНЬ ДЛЯ ЗАХИСТУ ТЕХНОЛОГІЇ WebRTC ВІД DoS-АТАК

3.1 Підготовка та налаштування тестового середовища

Зі стрімким розвитком інформаційно-комунікаційних систем, технологія WebRTC дедалі частіше використовується як основа для реалізації відеозв'язку, аудіодзвінків та обміну даними в реальному часі без встановлення додаткового програмного забезпечення. Проте з поширенням цієї технології зростає і інтерес зловмисників до пошуку її вразливостей, зокрема до можливостей організації атак типу відмови в обслуговуванні, які здатні порушити стабільну роботу сервісів або повністю вивести їх з ладу. У зв'язку з цим виникає потреба у створенні ефективних засобів моніторингу, виявлення та реагування на ознаки потенційних атак. У цьому розділі представлено розробку прототипу системи виявлення вторгнень, орієнтованої на специфіку WebRTC-протоколів та призначеної для виявлення DoS-атак на основі аналізу мережевого трафіку в режимі реального часу.

Для розгортання тестового середовища з метою реалізації та тестування системи виявлення вторгнень було обрано Oracle VM VirtualBox – безкоштовну та функціональну платформу віртуалізації, що дозволяє створювати ізольовані середовища з власними мережевими конфігураціями, операційними системами та програмним забезпеченням. Завдяки цьому стало можливим розгортання окремих віртуальних машин для емуляції клієнта, сервера WebRTC, атакувальника та системи виявлення атак, що дозволяє повністю контролювати експериментальне середовище без ризику впливу на основну операційну систему чи зовнішні мережі.

VirtualBox також забезпечив гнучке налаштування мережевих інтерфейсів (NAT, внутрішні мережі, адаптери міст), що дозволило імітувати реальні сценарії комунікації між учасниками WebRTC-сеансів, у тому числі маршрутизацію

трафіку через фільтруючі вузли. Такий підхід дав змогу детально відслідковувати мережеву активність, здійснювати захоплення трафіку, а також створювати контрольовані умови для генерації DoS-атак, забезпечуючи репрезентативність результатів моделювання та тестування.

На основі віртуалізаційної платформи Oracle VM VirtualBox і було створено дві основні віртуальні машини з різними функціональними ролями. В якості базової платформи для розгортання WebRTC-додатку, а також реалізації системи збору та аналізу мережевого трафіку, було обрано операційну систему Ubuntu 22.04 LTS. Цей дистрибутив забезпечує стабільність, широку підтримку інструментів для роботи з мережею та безпекою, а також сумісність із сучасними бібліотеками, необхідними для роботи з WebRTC. На цій машині було встановлено як серверну, так і клієнтську частину експериментального WebRTC-застосунку, а також інструменти для моніторингу та фіксації аномальної активності.

Друга віртуальна машина, побудована на основі Kali Linux, використовувалась для моделювання атак з боку умовного зловмисника. Kali – це дистрибутив, спеціалізований на задачах інформаційної безпеки та тестування на проникнення, що містить великий набір попередньо встановлених інструментів для проведення атак різного типу, включно з DoS. На цій машині було налаштовано мережеві сканери, генератори трафіку та утиліти для навантаження WebRTC-каналів. Це дозволило створити реалістичні сценарії з перевантаженням певних компонентів системи (сигналізаційного сервера, медіа-серверів, STUN/TURN вузлів), щоб перевірити ефективність системи виявлення атак у реальному часі.

Важливим етапом підготовки стало раціональне розподілення апаратних ресурсів між віртуальними машинами для забезпечення їх стабільної роботи. Першій машині з ОС Ubuntu 22.04 LTS було виділено 4 ГБ оперативної пам’яті, 2 віртуальні ядра CPU та 50 ГБ віртуального дискового простору, другій з ОС Kali Linux – 2 ГБ оперативної пам’яті, 2 віртуальні ядра CPU та 50 ГБ віртуального дискового простору Така конфігурація дозволила гарантувати достатню продуктивність для роботи WebRTC-додатків, інструментів моніторингу

мережевого трафіку та генераторів атак. Адекватне виділення ресурсів також стало запорукою точності вимірювань навантаження під час проведення експериментів із моделювання DoS-атак.

Jitsi Meet – це безкоштовна, відкрита платформа для відеоконференцій, яка базується на технології WebRTC і не потребує встановлення клієнтів на стороні користувача. Однією з ключових переваг Jitsi є її простота у розгортанні та налаштуванні, а також повністю відкритий вихідний код, що дозволяє дослідникам та розробникам адаптувати її під власні потреби. Важливо також зазначити, що Jitsi підтримує наскрізне шифрування, масштабованість через компоненти типу JVB (Jitsi VideoBridge) і чудово підходить для експериментів із мережевими атаками та захисними механізмами. Саме через комбінацію функціональності, відкритості та активної спільноти Jitsi Meet було обрано як основну платформу для реалізації тестового WebRTC-середовища.

У рамках цього розділу інсталяція Jitsi Meet проводилась на віртуальній машині з Ubuntu 22.04 LTS, оскільки цей дистрибутив офіційно підтримується та має повну сумісність із компонентами Jitsi. Це дозволяє швидко інтегрувати компоненти, необхідні для роботи сигналізації, медіа-трафіку та автентифікації користувачів. Особливістю вибраного підходу є можливість самостійного хостингу без залежності від сторонніх хмарних сервісів, що підвищує рівень контролю над безпекою.

Під час підготовки системи до встановлення Jitsi Meet було дотримано ряду технічних вимог. Зокрема, забезпечено наявність необхідних пакунків: gnupg2, nginx-full, curl, apt-transport-https, а також переконанося, що в системі активовано репозиторій universe, який потрібен для доступу до всіх залежностей. Для коректної роботи було використано OpenJDK 11, як рекомендовано документацією [24]. Оскільки багато кроків інсталяції вимагають привілейованого доступу, всі команди запускались з використанням sudo для забезпечення належного рівня прав. Після оновлення системи було підготовлено середовище до безпосереднього встановлення компонентів Jitsi, зокрема Prosody (XMPP-сервер), Jicofo (контролер конференцій) і сам WebRTC-бекенд.

Наступним кроком після підготовки системи стало встановлення Jitsi Meet. Перед інсталяцією необхідно визначити доменне ім’я сервера, яке буде використовуватися для доступу до WebRTC-платформи. У нашому випадку, оскільки інсталяція виконувалась у локальному віртуальному середовищі для моделювання атак та дослідження трафіку, було прийнято рішення використати локальний домен `meet.local`. Цей домен було прив’язано до IP-адреси віртуальної машини, яка після налаштування мережевого адаптера у режимі "bridge mode " отримала адресу 192.168.1.7 у локальній мережі.

Для коректної роботи Jitsi Meet потрібно задати FQDN (Fully Qualified Domain Name) на самій машині. Це робиться командою `sudo hostnamectl set-hostname meet.local`, після чого необхідно додати запис у файл `/etc/hosts`, що пов’язує IP-адресу з доменним іменем, наприклад: `192.168.1.7 meet.local`. Після цього слід перевірити коректність налаштування, виконавши `ping "$(hostname)"`, що у випадку успішної конфігурації має повернути відповідь від `meet.local`, як продемонстровано на рис. 3.1. Такий підхід дозволив повністю імітувати публічне середовище у локальній мережі, що особливо важливо для реалізації DoS-атак, аналізу мережевого трафіку, збору логів і тестування системи виявлення вторгнень без ризику для зовнішніх сервісів.

```
druzhko@druzhko:~$ ping meet.local
PING meet.local (192.168.1.7) 56(84) bytes of data.
64 bytes from meet.local (192.168.1.7): icmp_seq=1 ttl=64 time=0.047 ms
64 bytes from meet.local (192.168.1.7): icmp_seq=2 ttl=64 time=0.028 ms
64 bytes from meet.local (192.168.1.7): icmp_seq=3 ttl=64 time=0.031 ms
^C
--- meet.local ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.028/0.035/0.047/0.008 ms
```

Рисунок 3.1 – Перевірка правильної конфігурації доменного імені

Для забезпечення стабільної роботи Jitsi Meet та активації додаткових функцій, зокрема функції «лобі», необхідно було додати офіційний репозиторій Prosody – XMPP-сервер, який використовується Jitsi для обробки сигналізації та керування конференціями. Встановлення актуальної версії Prosody є критичним, адже в старих пакетах можуть відсутні важливі функціональні оновлення та

виправлення безпеки. Для Ubuntu 22.04 було виконано команду завантаження ключа репозиторію та додано його у сховище ключів системи, після чого додано сам репозиторій Prosody до джерел пакунків. Також була встановлена залежність `lua5.2`, необхідна для коректної роботи Prosody. Такий підхід гарантує, що під час оновлення системи буде завжди встановлено найновішу стабільну версію сервера.

Наступним етапом було додавання офіційного репозиторію Jitsi, щоб забезпечити доступ до актуальних пакетів Jitsi Meet. Для цього було імпортовано офіційний GPG-ключ репозиторію та додано Jitsi-репозиторій у системні джерела пакетів з відповідним підписом, що гарантує цілісність та автентичність пакетів. Завдяки цьому система отримала можливість автоматично отримувати оновлення Jitsi Meet безпосередньо з офіційних джерел, що підвищує безпеку та стабільність роботи платформи.

Після додавання необхідних репозиторіїв було оновлено всі джерела пакунків командою `sudo apt update`, що дозволило системі отримати актуальні списки доступних версій програмного забезпечення. Ці дії є стандартною та обов’язковою частиною процесу підготовки системи перед встановленням Jitsi Meet і дозволяють забезпечити коректну та безпечну інсталяцію останніх версій усіх компонентів.

Для забезпечення стабільної та безпечної роботи сервера Jitsi Meet важливо налаштувати міжмережевий екран з відкриттям необхідних портів, що дозволяють коректний трафік до різних сервісів системи. Перш за все, слід відкрити порт 80 TCP, який використовується для перевірки та автоматичного оновлення SSL-сертифікатів за допомогою Let's Encrypt – це обов’язкова вимога для забезпечення безпечного HTTPS-з’єднання. Порт 443 TCP необхідний для загального доступу до Jitsi Meet через захищений протокол HTTPS.

Для передачі аудіо- та відеопотоку Jitsi Meet використовує порт 10000 UDP, який повинен бути відкритим, адже основна медіа-комунікація відбувається саме через цей порт. Для віддаленого адміністрування сервера необхідно дозволити підключення через SSH – за замовчуванням це порт 22 TCP, але у разі зміни порту його варто налаштувати відповідно. Порти 3478 UDP та 5349 TCP пов’язані із

сервером STUN/TURN, які забезпечують коректну роботу медіа-трафіку у складних мережевих умовах, зокрема для обходу NAT і фаєрволу; порт 5349 TCP також є резервним каналом для передачі мультимедіа, якщо UDP заблоковано. Відкриття цих портів є важливим для надійної роботи конференц-зв’язку, особливо у корпоративних мережах.

Якщо використовується інструмент ufw (Uncomplicated Firewall), то можна легко застосувати наведені правила за допомогою простих команд: `sudo ufw allow 80/tcp`, `sudo ufw allow 443/tcp`, `sudo ufw allow 10000/udp`, `sudo ufw allow 22/tcp`, `sudo ufw allow 3478/udp` та `sudo ufw allow 5349/tcp`. Після цього слід активувати фаєрвол командою `sudo ufw enable`. Для перевірки коректності налаштувань слід використати `sudo ufw status verbose`, що виведе детальний стан правил. Така конфігурація гарантує, що сервер буде доступним і захищеним від небажаних зовнішніх підключень.

Для забезпечення зашифрованого обміну даними у WebRTC-застосунку необхідно налаштувати TLS-сертифікат. Під час встановлення Jitsi Meet користувач має кілька варіантів вибору сертифікату. Рекомендованим є використання сертифікату від Let's Encrypt, який автоматично підтверджує автентичність сервера і не викликає попереджень у браузерах. Цей варіант забезпечує високий рівень довіри і безпеки без особливих витрат.

Проте, якщо з якоїсь причини потрібно використовувати власний сертифікат, його слід отримати заздалегідь і під час встановлення Jitsi Meet обрати відповідну опцію (рис. 3.2). Існує також можливість генерації самопідписного сертифікату безпосередньо під час інсталяції, що може бути корисним для тестових середовищ і локальних розгортань. Однак використання самопідписного сертифікату не рекомендується для продакшн-середовищ, оскільки браузери користувачів будуть показувати попередження про ненадійність сертифікату, а мобільні застосунки Jitsi Meet взагалі не зможуть підключитися до сервера без валідного сертифікату від довіреного центру сертифікації.

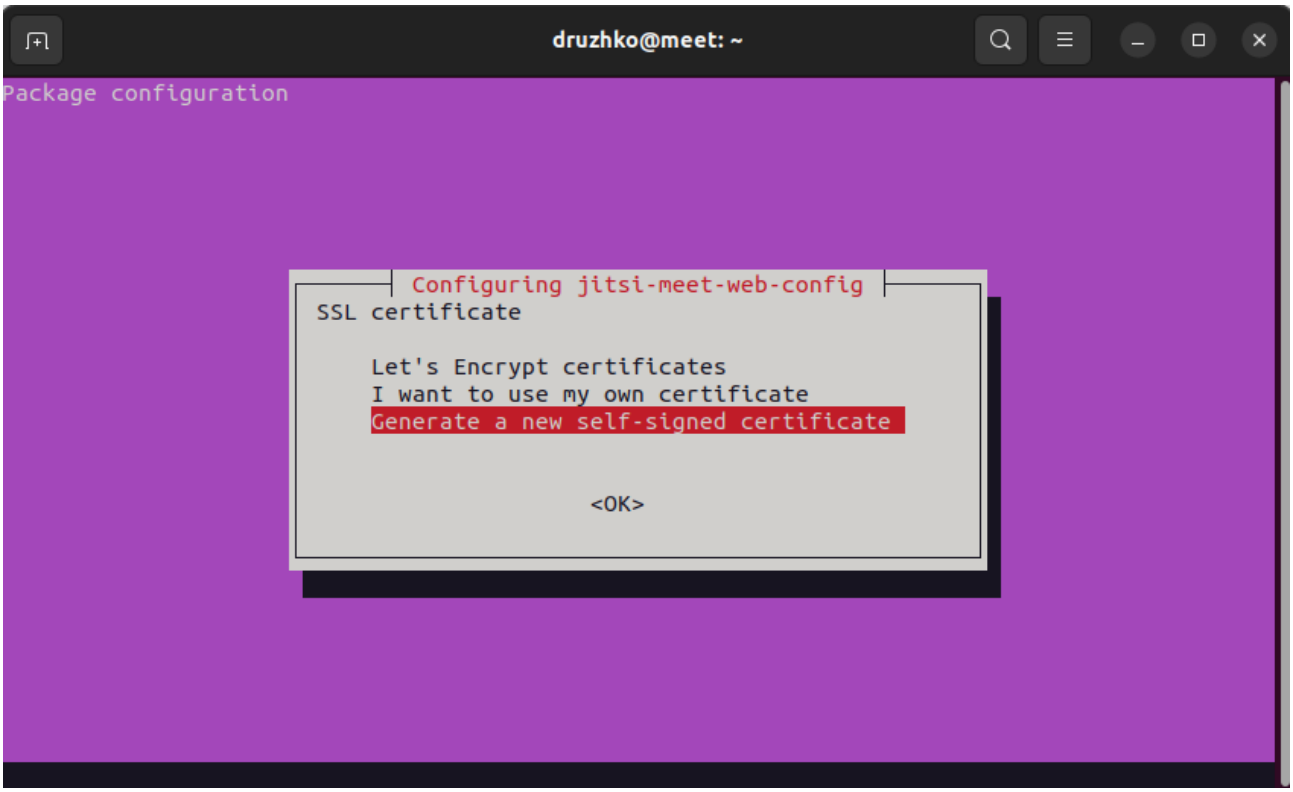

Рисунок 3.2 – Вибір опції генерування самопідписного сертифікату під час конфігурації встановленого пакету

Під час встановлення Jitsi Meet інсталятор автоматично перевіряє наявність веб-серверів Nginx або Apache і налаштовує віртуальний хост для обслуговування Jitsi Meet. Якщо на машині вже запущений Nginx на порту 443, конфігурація TURN-сервера буде пропущена, щоб уникнути конфлікту портів. Для інсталяції Jitsi Meet використовується команда sudo apt install jitsi-meet.

Після завершення встановлення Jitsi Meet варто переконатися, що інсталяція пройшла успішно та всі компоненти функціонують коректно. Для цього достатньо відкрити веб-браузер (наприклад, Firefox, Chrome або Safari) і ввести доменне ім’я або IP-адресу сервера, вказану під час налаштування. У випадку з локальним середовищем – наприклад, `http://meet.local` або `http://192.168.1.7`.

Якщо було використано самопідписний TLS-сертифікат, браузер видасть попередження про те, що з’єднання не є надійним. Це нормально в умовах тестового середовища – достатньо підтвердити довіру до сертифікату вручну. Водночас мобільні застосунки Jitsi Meet на iOS та Android, швидше за все, не зможуть під’єднатися до сервера через відсутність сертифіката від довіреного

центру сертифікації. Після підтвердження сертифікату має з’явитися головна сторінка Jitsi Meet з можливістю створити нову зустріч. Є можливість створити тестову конференцію та перевірити, чи можуть інші учасники приєднатися до неї

3.2 Реалізація DoS-атаки

Для перевірки стійкості Jitsi Meet до атак типу «відмова в обслуговуванні», було реалізовано скрипт на мові Python із використанням бібліотеки Scapy, що дозволяє генерувати та надсилати мережеві пакети. Атака імітує масове надсилання STUN Binding-запитів на порт 10000/UDP, який використовується Jitsi Videobridge для передавання медіа.

У даному прикладі IP-адреса цільового сервера встановлена як 192.168.1.7 – саме на ній локально було розгорнуто тестовий екземпляр Jitsi Meet. Загальна кількість UDP-пакетів, які надсилаються в рамках атаки, становить 50 000, що дозволяє створити значне навантаження. Для підвищення ефективності атаки передбачено запуск 20 паралельних потоків – кожен із них надсилає свою частину трафіку.

Кожен пакет містить STUN Binding Request – це типовий тип повідомлення, який використовується для визначення зовнішньої IP-адреси клієнта та ініціації з'єднання через NAT. Оскільки такі пакети відповідають очікуваному формату сигналів STUN, сервер обробляє їх як справжні, що збільшує навантаження на обробку трафіку та ресурси Jitsi Videobridge.

Функція create_stun() формує базову структуру STUN-повідомлення з випадковим вмістом, а send_packets() – виконує безперервне надсилання таких пакетів у рамках свого потоку. Завдяки використанню бібліотеки threading, усі потоки запускаються одночасно, що дозволяє змоделювати ситуацію ближчу до реальної атаки з багатьох джерел (див. Додаток А).

Після завершення всіх потоків скрипт виводить загальний час, витрачений на атаку. Початкова версія скрипта виконувалася в однопоточному режимі, що суттєво обмежувало інтенсивність генерованого трафіку і не мало жодного помітного впливу на працездатність чи навантаження Jitsi-сервера. Для

досягнення більш реалістичного ефекту масової DoS-атаки до скрипта було додано підтримку багатопотоковості – зокрема, ініціалізовано 20 паралельних потоків, кожен з яких надсилав власну частину пакетів. Реалізацію такої атаки продемонстровано на рис. 3.3.

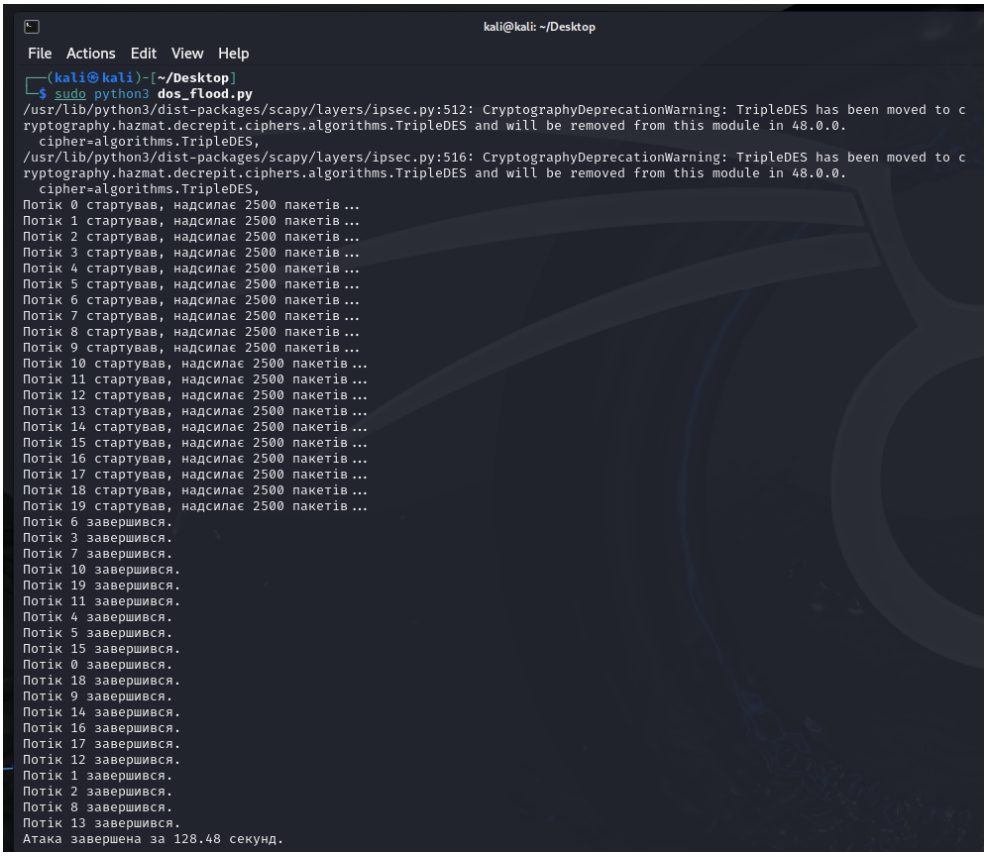

Рисунок 3.3 – Процес реалізації багатопотокової DoS-атаки

У результаті цього модифікований код досяг відчутного впливу: в нормальному стані процес Jitsi Videobridge (jvb) використовував приблизно 1.7% ресурсів процесора, тоді як під час пікового навантаження, спричиненого атакою, цей показник сягав 17.6%, що в 10 разів перевищує початкове значення (рис. 3.4). Це підтверджує ефективність використання паралельного підходу при моделюванні DoS-атак та дозволяє оцінити реакцію системи на інтенсивний вхідний трафік.

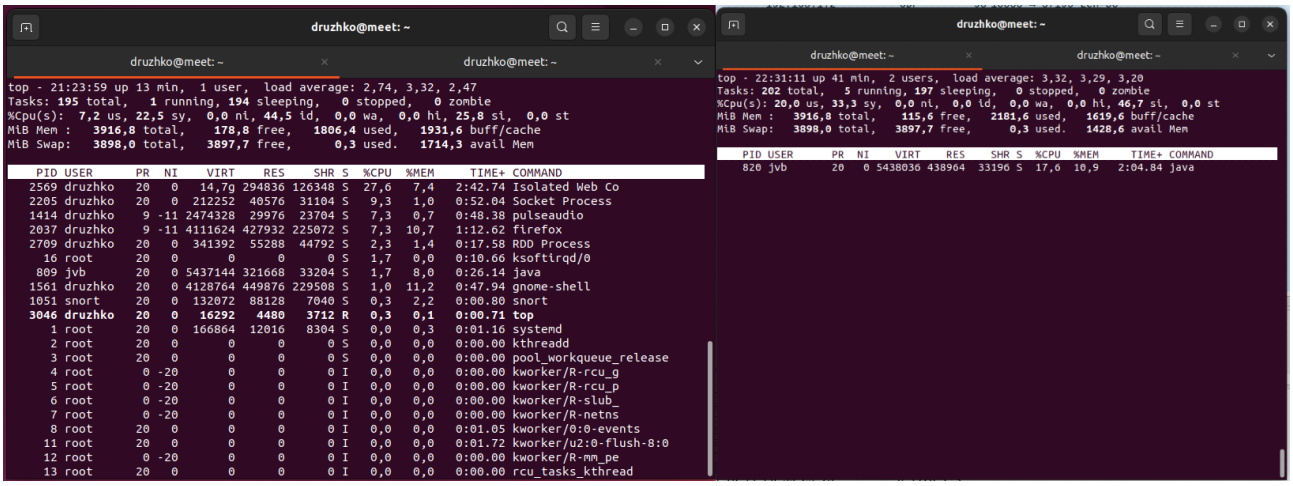

Рисунок 3.4 – Навантаження процесу jvb в штатному стані (зліва) та під час DoS-атаки (справа)

3.3 Захоплення та аналіз WebRTC-трафіку

Для виявлення потенційних вразливостей у роботі системи відеозв’язку, а також для оцінки ефективності реалізованих DoS-атак, було здійснено детальне захоплення мережевого трафіку під час сеансу відеоконференції. Це дозволило проаналізувати поведінку системи в штатному режимі та під навантаженням, простежити структуру обміну даними між клієнтами та сервером, а також зафіксувати зміни у характері передавання пакетів, що могли свідчити про порушення нормального функціонування сервісу.

Основним інструментом для захоплення трафіку слугував Wireshark – один з найпотужніших аналізаторів мережевих протоколів [25]. Цей інструмент дозволяє не лише зберігати трафік у вигляді дампів для подальшого аналізу, а й оперативно візуалізувати потоки, фільтрувати пакети за типами протоколів (STUN, DTLS, RTP, RTCP тощо) та досліджувати структуру окремих повідомлень.

Захоплення проводилося на віртуальній машині з ОС Ubuntu 22.04 LTS, де сервер Jitsi Meet працював із самопідписаним TLS-сертифікатом. Аналіз трафіку дозволив не лише відстежити етапи встановлення з'єднання між клієнтами, а й виявити характерну поведінку системи під час атаки – наприклад, збільшення

кількості STUN-запитів або повторюваність фреймів, що можуть свідчити про аномальну активність.

Перед початком захоплення необхідно було вибрати коректний мережевий інтерфейс у Wireshark. У моєму випадку це був інтерфейс enp0s2, що відповідав підключенню віртуальної машини до локальної мережі. Важливо переконатися, що вибраний інтерфейс справді обробляє трафік, пов'язаний із взаємодією між клієнтами Jitsi Meet та сервером.

Щоб зосередитися лише на релевантному трафіку, було встановлено наступний фільтр захоплення у Wireshark: (stun || rtp || dtls || rtcp) && udp.port == 10000. Цей фільтр дозволяє захоплювати пакети, що належать до ключових протоколів, які використовуються в рамках WebRTC – а саме STUN, RTP, RTCP і DTLS – за умови, що вони передаються через UDP-порт 10000. Саме цей порт за замовчуванням використовується компонентом Jitsi Videobridge, хоча в окремих випадках він може бути змінений. Це можна перевірити у файлі конфігурації за шляхом /etc/jitsi/videobridge/jvb.conf.

Після налаштування інтерфейсу й фільтру починалося безпосереднє захоплення трафіку. Для цього запускається тестова конференція в Jitsi Meet – відкривається браузер, вводилась адреса серверу meet.local та створювалась кімната з довільною назвою. До конференції приєднується щонайменше два учасники: один з тієї ж самої віртуальної машини на ОС Ubuntu, інший - з хосту. Вмикається відео та аудіо, щоб згенерувати реальний RTP-трафік. Захоплення тривало 2–3 хвилини, протягом яких виконувались різноманітні дії, щоб охопити всі типи WebRTC-пакетів.

Таким чином було здійснено захоплення трафіку під час нормальної роботи Jitsi Meet, а також під час проведення DoS-атаки. Результати захоплення були збережені у два окремі файли: jitsi_normal_traffic.pcap для звичайного трафіку та jitsi_dos_traffic.pcap для трафіку, що виникав під час атаки. Уже на цьому етапі можна було провести первинний аналіз мережевих пакетів як візуально, використовуючи інтерфейс Wireshark, так і за допомогою його вбудованих інструментів.

Візуальний огляд показав надзвичайно високу кількість STUN-пакетів у файлі з DoS-трафіком, що свідчить про аномальну активність. Пакети мали однотипний вміст, а також розміри, які не відповідали стандартним характеристикам реального трафіку під час нормальної роботи. Це дозволило вже на початковому етапі диференціювати нормальну роботу системи від стану під атаками.

Для поглибленого аналізу збереженого трафіку було використано Python-скрипт із бібліотеками pandas та matplotlib, який дозволив виконати обчислення ключових характеристик і побудувати наочні візуалізації для порівняння нормального та DoS-трафіку. Спочатку обидва CSV-файли з трафіком було завантажено в таблиці DataFrame, після чого проведено початкову аналітику. Для кожного набору обчислювались базові метрики: загальна кількість пакетів, список використаних протоколів, кількість унікальних IP-адрес джерела та призначення, середній розмір пакета, а також частота передачі пакетів, розрахована як кількість пакетів за секунду.

Тож, після запуску скрипта parse_pcap.py було здійснено порівняльний аналіз двох типів трафіку: нормального та трафіку, що містить DoS-атаку. Результати демонструють істотні відмінності між ними за низкою ключових параметрів, що дозволяє зробити висновки щодо характеру атаки та її впливу на систему.У нормальному трафіку було зафіксовано всього 67 пакетів, що передавались між двома джерелами та двома адресами призначення. Основними використовуваними протоколами є RTCP, STUN та DTLSv1.2, що типово для сеансів WebRTC у звичайному режимі. Середній розмір одного пакета становив 123.82 байт, а частота надсилання – близько 3.22 пакета на секунду, що свідчить про помірну активність та відсутність аномальних сплесків трафіку.

Натомість у трафіку з ознаками DoS-атаки спостерігається значне навантаження: кількість пакетів зросла до 12672, що майже у 190 разів більше за нормальний обсяг. Особливо помітна частота надсилання пакетів – 859.56 пакетів на секунду, що є типовою ознакою DoS-атаки, орієнтованої на перевантаження системи обробки сигналів WebRTC. Така активність суттєво перевищує

нормальні показники і може призвести до деградації якості з’єднання, затримок, або навіть повної відмови у обслуговуванні.

Також заслуговує на увагу середній розмір пакета у DoS-трафіку – 291.92 байт, що значно більше, ніж у нормальному (123.82 байт). Це може вказувати на свідоме формування пакетів певного розміру, які мають збільшене навантаження на пропускну здатність або обробку на стороні приймача.

Далі було побудовано кругові діаграми (рис. 3.5), які відобразили розподіл протоколів у кожному з файлів. У нормальному трафіку спостерігався більш рівномірний розподіл між STUN, DTLS та RTCP, тоді як у DoS-трафіку домінував виключно STUN, що вказує на зловмисну активність.

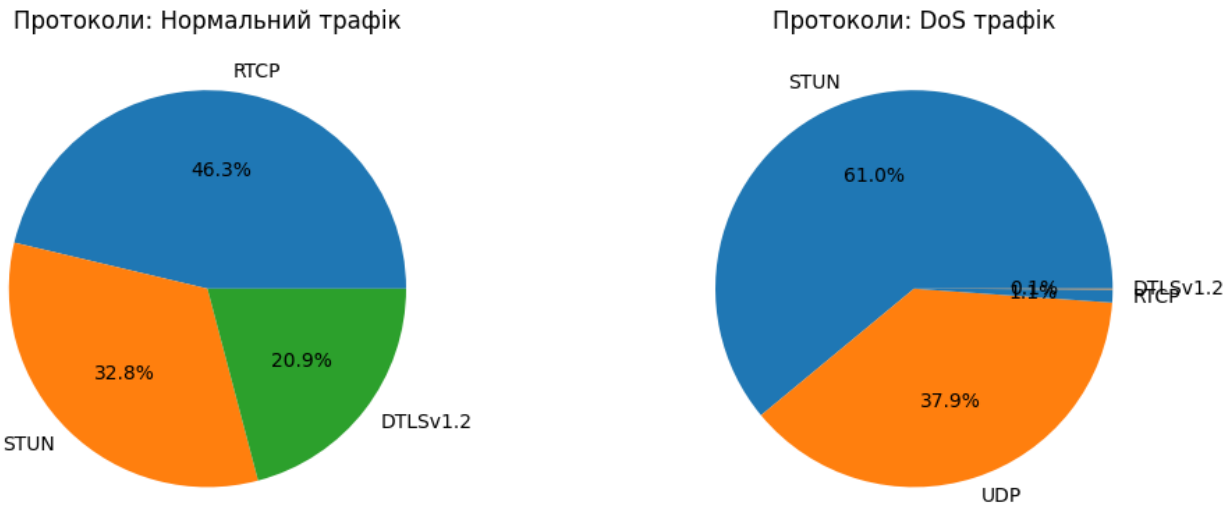

Рисунок 3.5 – Розподіл захоплених пакетів за протоколами в різних умовах функціонування

Для кількісного порівняння метрик було створено стовпчикову діаграму (рис. 3.6), яка показала зростання кількості пакетів при DoS-атаці, суттєве збільшення частоти передачі та незначну різницю у середньому розмірі пакета. Це підтверджує, що атака була спрямована не на обсяг переданих даних, а саме на навантаження мережі великою кількістю однотипних коротких STUN-повідомлень.

Рисунок 3.6 – Аналіз трафіку за основними параметрами

Окрему увагу було приділено аналізу за протоколами. Для STUN, DTLSv1.2, RTCP та UDP окремо обчислювались ті самі метрики, що й раніше, і будувались порівняльні графіки. Найбільша відмінність фіксувалась у STUN: у DoS-режимі кількість таких пакетів зросла у сотні разів, їх частота надсилання сягала кількох сотень пакетів за секунду, а вміст залишався майже незмінним. Інші протоколи в DoS-трафіку реагували по різному: деякі демонстрували зниження активності, як DTLSv1.2, деякі підвищення, як RTCP.

Ці результати дозволяють зробити висновок, що DoS-атака реалізовувалась саме через навмисне перенавантаження WebRTC-сервера STUN-пакетами з високою частотою, що може бути використано для побудови правил фільтрації або як ознака для системи виявлення вторгнень.

3.4 Розробка системи виявлення вторгнень

Для моніторингу стану WebRTC-з’єднання та виявлення потенційно шкідливого трафіку було розроблено систему виявлення вторгнень, яка працює в режимі реального часу та орієнтована на аналіз трафіку STUN, DTLSv1.2 та RTCP-протоколів. Система реалізована мовою програмування Python із

використанням інструмента tshark – консольного аналога Wireshark, що дозволяє ефективно фільтрувати та зчитувати мережеві пакети [26].

Основною структурною одиницею аналізу в десктопному застосунку є клас Packet, який зберігає ключову інформацію про кожен перехоплений пакет: час надходження, IP-адреси джерела та призначення, протокол, розмір та службову інформацію. Збір даних здійснюється у фоновому потоці за допомогою утиліти tshark, яка конфігурується на перехоплення UDP-трафіку порту 10000 — типової транспортної основи для медіа-компонентів WebRTC. Всі отримані пакети динамічно відображаються у графічному інтерфейсі користувача в таблиці з останніми записами.

Зібрані пакети аналізуються у реальному часі за допомогою декількох евристичних перевірок. Для агрегації статистики використовується список peer_list, який містить екземпляри PeerData для кожної унікальної IP-адреси джерела. Ці об’єкти зберігають стан активності, кількість отриманих пакетів, час останнього виявлення відхилення та список причин зміни стану. Виявлена підозріла активність автоматично оновлюється у GUI через окрему таблицю, а статусна панель змінює повідомлення залежно від ступеня загрози (див. рис. 3.7).

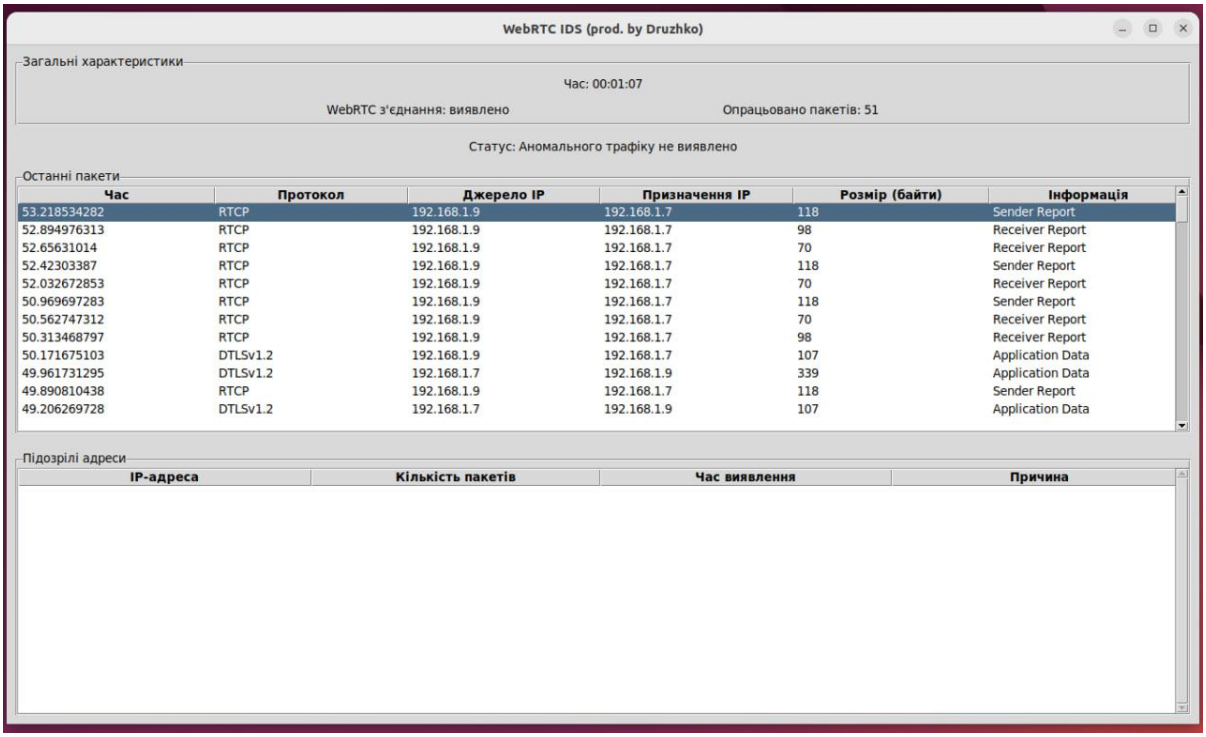

Рисунок 3.7 – Демонстрація десктопної версії додатку

Система використовує заздалегідь задані норми поведінки для кожного протоколу – зокрема, середню частоту надсилання пакетів та середній розмір. Для STUN-протоколу також контролюється співвідношення запитів та відповідей, що дозволяє виявити ознаки атаки через відсутність відповідей (наприклад, при фальсифікації джерел) (див. Додаток В).

На основі зібраної статистики кожні 10 секунд проводиться перевірка, чи не перевищено порогові значення, задані у вигляді коефіцієнтів до норми. Для кожного показника система має два рівні тригерів:

- Попередження – при незначному перевищенні середнього значення параметру;
- Загроза – при значному перевищенні параметру (у 3-5 разів – залежно від протоколу та типу параметра).

Виявлення аномалій базується на таких ключових критеріях:

- Частота передачі пакетів – фіксується кількість пакетів, отриманих за останні 10 секунд.
- Середній розмір пакета – дозволяє виявити надмірне навантаження за рахунок великих обсягів.
- Кількість STUN-запитів без відповіді – індикатор спроб сканування або DoS-атаки.

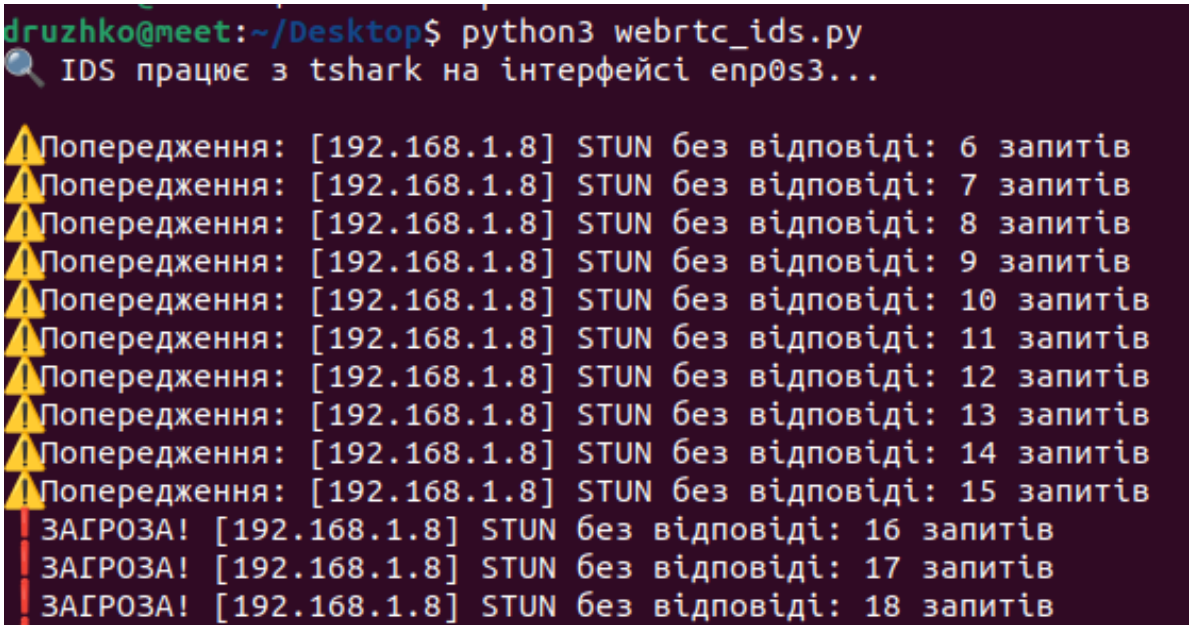

Рисунок 3.8 – Реагування програми на аномальні параметри трафіку

У випадку перевищення одного з контрольних показників система виводить повідомлення до терміналу, класифікуючи подію як «Попередження» або «Загрозу», як наведено на рис. 3.8.

Таким чином, розроблена система дозволяє оперативно виявляти підозрілу активність, зокрема пов’язану з DoS-атаками, і може бути інтегрована в більш широку систему безпеки WebRTC-додатків. Її гнучкість дозволяє адаптувати порогові значення та правила виявлення під конкретні сценарії використання або особливості мережевого середовища.

Висновки за розділом 3

У третьому розділі було детально розглянуто процес розробки системи виявлення вторгнень, спрямованої на захист технології WebRTC від атак типу «відмова в обслуговуванні». На першому етапі було створено тестове середовище, що дозволило моделювати реальний WebRTC-трафік та проводити експерименти у контрольованих умовах.

Після цього було здійснено збір та попередній аналіз захопленого трафіку за допомогою інструмента tshark. У результаті були зафіксовані суттєві відмінності між нормальним і аномальним трафіком, зокрема в частоті надсилання пакетів, середньому розмірі та кількості STUN-запитів без відповідей.

На завершальному етапі було реалізовано власну систему IDS, яка в реальному часі аналізує WebRTC-трафік, виявляє відхилення від типових характеристик і класифікує події за рівнем загрози. Система виявилась здатною ефективно розпізнавати DoS-атаки за допомогою простих, але дієвих евристичних правил, що базуються на частоті, об’ємі трафіку та наявності відповідей.

Таким чином, результати розділу підтверджують можливість створення ефективного засобу первинного захисту WebRTC-комунікацій від DoS-атак з мінімальними витратами на обчислювальні ресурси та без потреби у складних методах машинного навчання.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було комплексно досліджено питання забезпечення захисту технології WebRTC від DoS-атак – однієї з найпоширеніших загроз для систем реального часу. WebRTC, будучи основою сучасних браузерних аудіо- та відеокомунікацій, має низку вбудованих механізмів безпеки, проте все ще залишається вразливою до навмисного перевантаження трафіком. Проведене дослідження дозволило поглибити розуміння структури WebRTC, оцінити потенційні вектори атак і створити ефективну систему виявлення вторгнень, здатну реагувати на підозрілу активність у режимі реального часу. Практичні результати цієї роботи можуть бути застосовані для підвищення надійності систем зв’язку в різних WebRTC-додатках.

У першому розділі було проаналізовано фундаментальні принципи функціонування WebRTC: вивчено його основне призначення – забезпечення прямого однорангового мультимедійного обміну між клієнтами без необхідності в серверному посереднику, а також структуру компонентів, таких як PeerConnection, MediaStream та DataChannel. Детально розглянуто механізми сигналізації та встановлення з’єднання через STUN, TURN і ICE, а також криптографічні протоколи, зокрема DTLS та SRTP, які забезпечують конфіденційність і цілісність даних. Було також наведено переваги (низька затримка, пряме з’єднання) та недоліки (складність забезпечення безпеки) WebRTC, що стало теоретичною основою для подальшого аналізу вразливостей.

Другий розділ присвячено загрозам безпеці в контексті WebRTC, зокрема атакам типу «відмова в обслуговуванні». Проведено систематизацію видів DoS-атак, розглянуто їх особливості у WebRTC-середовищі, а також механізми, що частково захищають від них, зокрема перевірку STUN-запитів, регуляцію сигналінгу та обмеження кількості з’єднань. Описано типові вектори атак: флудинг STUN/TURN-серверів, перевантаження каналу DataChannel, зловживання механізмами NAT traversal. Аналіз наявних підходів до запобігання

атакам (фільтрація трафіку, таймаути, CAPTCHA, виявлення аномалій) засвідчив необхідність створення окремих IDS-рішень, які враховують специфіку WebRTC-протоколів.

У третьому розділі було реалізовано практичну частину – побудовано тестове середовище на базі WebRTC-додатка з підключенням STUN-серверів та моделюванням DoS-атак через автоматичну генерацію навантаження. За допомогою інструменту tshark проводився перехоплення, збереження та аналіз трафіку, що проходив між клієнтами. На основі отриманих даних розроблено систему виявлення вторгнень, яка оцінює частоту, обсяг і повторюваність пакетів – основні індикатори DoS-атаки.

Виходячи із поставленої мети кваліфікаційної роботи, були виконані наступні завдання:

- Проаналізовано архітектуру, принципи функціонування та протоколи, що використовуються в технології WebRTC.
- Ідентифіковано основні вразливості WebRTC-додатків до атак типу «відмова в обслуговуванні» та проаналізовано існуючі підходи до захисту від них.
- Підготовлено багатофункціональне тестове середовище для моделювання DoS-атак, збору та дослідження трафіку WebRTC-додатків.
- Виконано захоплення та аналіз WebRTC-трафіку з метою виявлення характерних ознак атак типу DoS.
- Розроблено систему виявлення DoS-атак у WebRTC-додатках, що здійснює моніторинг мережевого трафіку в реальному часі з використанням мови програмування Python і модулю Wireshark..

Апробація роботи відбулася на VIII Міжнародній науково-практичній конференції «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSICS) 11 квітня 2025 року.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Technologies enabling remote work and collaboration: WebRTC, Intelliswift [Електронний ресурс] – Режим доступу до ресурсу: https://www.intelliswift.com/insights/blogs/technologies-enabling-remote-work-and-collaboration-webrtc (дата звернення: 03.06.2025).

2. Top 100 Open Source WebRTC Projects for 2023, WebRTC Developers [Електронний ресурс] – Режим доступу до ресурсу: https://www.webrtc-developers.com/webrtc-top-100-open-source-projects-for-2023/#top-complete-webrtc-solutions (дата звернення: 03.06.2025).

3. WebRTC stack architecture and layers, Altanai, Telecom R & D [Електронний ресурс] – Режим доступу до ресурсу: https://telecom.altanai.com/2013/07/31/webrtc/ (дата звернення: 03.06.2025).

4. WebRTC: Security in Depth, WebRTC Security [Електронний ресурс] – Режим доступу до ресурсу: https://webrtc-security.github.io/ (дата звернення: 03.06.2025).

5. Overview: Real-Time Protocols for Browser-Based Applications: RFC 8825, M. Alvestrand, Internet Engineering Task Force (IETF) [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc8825 (дата звернення: 03.06.2025).

6. What are STUN, TURN, and ICE?, LiveSwitch [Електронний ресурс] – Режим доступу до ресурсу: https://developer.liveswitch.io/liveswitch-server/guides/what-are-stun-turn-and-ice.html (дата звернення: 03.06.2025).

7. Розуміння протоколу передачі даних у реальному часі (RTP), CQR Company [Електронний ресурс] – Режим доступу до ресурсу: https://cqr.company/ua/wiki/protocols/understanding-the-real-time-transport-protocol-rtp-how-it-works-and-its-importance-in-media-streaming/ (дата звернення: 03.06.2025).

8. OpenVidu: Open-source WebRTC platform, OpenVidu Community [Електронний ресурс] – Режим доступу до ресурсу: https://openvidu.io/ (дата звернення: 03.06.2025).

9. Understanding Janus: A WebRTC Server, Meetecho [Електронний ресурс] – Режим доступу до ресурсу: https://janus.conf.meetecho.com/ (дата звернення: 03.06.2025).

10. Jitsi Meet, Jitsi Community [Електронний ресурс] – Режим доступу до ресурсу: https://jitsi.org/jitsi-meet/ (дата звернення: 03.06.2025).

11. WebRTC Video Calling with Twilio, Twilio [Електронний ресурс] – Режим доступу до ресурсу: https://www.twilio.com/en-us/video (дата звернення: 03.06.2025).

12. Real-time communication with Agora, Agora [Електронний ресурс] – Режим доступу до ресурсу: https://www.agora.io/en/ (дата звернення: 03.06.2025).

13. Distributed Denial of Service Attacks Prevention, Detection and Mitigation – A Review, U. Rahamathullah, E. Karthikeyan, SSRN Electronic Journal [Електронний ресурс] – Режим доступу до ресурсу: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3852902 (дата звернення: 03.06.2025).

14. Що таке DDoS-атака?, CIP Ukraine [Електронний ресурс] – Режим доступу до ресурсу: https://cip.gov.ua/ua/faqs/sho-take-ddos-ataka (дата звернення: 03.06.2025).

15. DoS and DDoS Attacks, MEVspace [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mevspace.com/en/articles/articles-content/dos-and-ddos-attacks (дата звернення: 03.06.2025).

16. DDoS history, StormWall Network [Електронний ресурс] – Режим доступу до ресурсу: https://stormwall.network/resources/blog/ddos-history (дата звернення: 03.06.2025).

17. Security Considerations for WebRTC: RFC 8826, E. Rescorla, Internet Engineering Task Force [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc8826 (дата звернення: 03.06.2025).

18. WebRTC Security: Best Practices and Key Risks Explained, Digital Samba [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalsamba.com/blog/webrtc-security (дата звернення: 03.06.2025).

19. What is Datagram Transport Layer Security (DTLS), Wallarm [Електронний ресурс] – Режим доступу до ресурсу: https://www.wallarm.com/what/what-is-datagram-transport-layer-security-dtls (дата звернення: 03.06.2025).

20. SRTP Deep Dive, Twilio [Електронний ресурс] – Режим доступу до ресурсу: https://www.twilio.com/en-us/blog/srtp-deep-dive (дата звернення: 03.06.2025).

21. WebRTC IP Address Handling Requirements: RFC 8827, G. Salgueiro, Internet Engineering Task Force (IETF) [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc8827 (дата звернення: 03.06.2025).

22. WebRTC and Man-in-the-Middle Attacks, S. Fogel, WebRTC Hacks [Електронний ресурс] – Режим доступу до ресурсу: https://webrtchacks.com/webrtc-and-man-in-the-middle-attacks/ (дата звернення: 03.06.2025).

23. Дружко Д., Бабенко Ю. Дослідження вразливості технології WebRTC до DoS-атак // Проблеми кібербезпеки інформаційно-комунікаційних систем : матеріали VIII Міжнар. наук.-практ. конф., м. Київ, 2025. – С. 140–141.

24. Jitsi Meet Handbook: DevOps Guide – Quickstart, Jitsi Community [Електронний ресурс] – Режим доступу до ресурсу: https://jitsi.github.io/handbook/docs/devops-guide/devops-guide-quickstart/ (дата звернення: 03.06.2025).

25. Wireshark User’s Guide, Wireshark Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://www.wireshark.org/docs/wsug_html_chunked/ (дата звернення: 03.06.2025).

26. Wireshark Tshark Documentation, Wireshark Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html (дата звернення: 03.06.2025).

ДОДАТКИ

Додаток А

Лістинг коду реалізації DoS-атаки

```
from scapy.all import *
import random
import threading
import time

target_ip = "192.168.1.7"
target_port = 10000
num_requests = 50000
threads = 5

def random_bytes(length):
    return bytes(random.getrandbits(8) for _ in range(length))

def create_stun():
    return b'\x00\x01\x00\x00\x21\x12\xa4\x42' + random_bytes(12)

def send_packets(thread_id, count):
    print(f"Потік {thread_id} стартував, надсилає {count} пакетів...")
    for _ in range(count):
        pkt = IP(dst=target_ip) / UDP(sport=RandShort(), dport=target_port) / Raw(load=create_stun())
        send(pkt, verbose=False)
    print(f"Потік {thread_id} завершився.")

start_time = time.time()
threads_list = []
requests_per_thread = num_requests // threads
```

Продовження додатку А

```
for i in range(threads):
    t = threading.Thread(target=send_packets, args=(i, requests_per_thread))
    threads_list.append(t)
    t.start()

for t in threads_list:
    t.join()

print(f"Атака закінчена за {time.time() - start_time:.2f} секунд.")
```

Додаток Б

Лістинг коду аналізу WebRTC -трафіку

```
import pandas as pd
import matplotlib.pyplot as plt

normal_df = pd.read_csv('normal.csv')
dos_df = pd.read_csv('dos.csv')

def analyze_traffic(df, name):
    print(f"\nАналіз трафіку: {name}")
    print(f'Кількість пакетів: {len(df)}")
    print(f'Протоколи: {df['Protocol'].unique()}")
    print(f'Джерела: {df['Source'].nunique()}")
    print(f'Призначення: {df['Destination'].nunique()}")
    print(f'Середній розмір пакета: {df['Length'].mean():.2f} байт")
    time_duration = df['Time'].max() - df['Time'].min()
    packet_rate = len(df) / time_duration if time_duration > 0 else 0
    print(f"Частота пакетів: {packet_rate:.2f} пакетів/сек")

analyze_traffic(normal_df, "Нормальний трафік")
analyze_traffic(dos_df, "DoS трафік")

def plot_protocol_distribution(normal_df, dos_df):
    normal_protocols = normal_df['Protocol'].value_counts()
    dos_protocols = dos_df['Protocol'].value_counts()

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    ax1.pie(normal_protocols, labels=normal_protocols.index, autopct='%1.1f%%',
colors=['#1f77b4', '#ff7f0e', '#2ca02c'])
    ax1.set_title('Протоколи: Нормальний трафік')
```

```
        ax2.pie(dos_protocols, labels=dos_protocols.index, autopct='%1.1f%%',
colors=['#1f77b4', '#ff7f0e'])
        ax2.set_title('Протоколи: DoS трафік')
        plt.show()

    plot_protocol_distribution(normal_df, dos_df)

    # ===== Порівняльні графіки для всього трафіку =====
    def plot_general_comparison(normal_df, dos_df):
        def get_stats(df):
            duration = df['Time'].max() - df['Time'].min()
            return {
                'count': len(df),
                'rate': len(df) / duration if duration > 0 else 0,
                'avg_size': df['Length'].mean()
            }

        normal_stats = get_stats(normal_df)
        dos_stats = get_stats(dos_df)

        labels = ['Кількість пакетів  (за 20 сек.)', 'Частота (пакети/сек.)', 'Середній розмір
(байти)']
        normal_values = [normal_stats['count'], normal_stats['rate'], normal_stats['avg_size']]
        dos_values = [dos_stats['count'], dos_stats['rate'], dos_stats['avg_size']]

        x = range(len(labels))
        plt.figure(figsize=(10, 6))
        plt.bar(x, normal_values, width=0.4, label='Нормальний', align='center')
        plt.bar([p + 0.4 for p in x], dos_values, width=0.4, label='DoS', align='center')

        for i, v in enumerate(normal_values):
            plt.text(i, v + max(normal_values + dos_values) * 0.01, f"{v:.1f}", ha='center',
va='bottom')
```

```
        for i, v in enumerate(dos_values):
            plt.text(i + 0.4, v + max(normal_values + dos_values) * 0.01, f"{v:.1f}", ha='center',
va='bottom')

        plt.xticks([p + 0.2 for p in x], labels)
        plt.ylabel('Значення')
        plt.title('Загальні метрики трафіку')
        plt.legend()
        plt.grid(True, linestyle='--', alpha=0.6)
        plt.show()

    plot_general_comparison(normal_df, dos_df)

    # ===== Аналіз по протоколах =====
    def plot_per_protocol(normal_df, dos_df, protocols):
        for proto in protocols:
            norm_p = normal_df[normal_df['Protocol'] == proto]
            dos_p = dos_df[dos_df['Protocol'] == proto]

            if norm_p.empty and dos_p.empty:
                continue  # Протокол відсутній у обох

            def get_stats(df):
                if df.empty:
                    return {'count': 0, 'rate': 0, 'avg_size': 0}
                duration = df['Time'].max() - df['Time'].min()
                return {
                    'count': len(df),
                    'rate': len(df) / duration if duration > 0 else 0,
                    'avg_size': df['Length'].mean()
                }

            norm_stats = get_stats(norm_p)
            dos_stats = get_stats(dos_p)
```

Продовження додатку Б

```
        labels = ['Кількість (за 20 сек.)', 'Частота  (пакети/сек.)', 'Середній розмір  (байти)']
        norm_values = [norm_stats['count'], norm_stats['rate'], norm_stats['avg_size']]
        dos_values = [dos_stats['count'], dos_stats['rate'], dos_stats['avg_size']]

        x = range(len(labels))
        plt.figure(figsize=(10, 5))
        plt.bar(x, norm_values, width=0.4, label='Нормальний', align='center')
        plt.bar([p + 0.4 for p in x], dos_values, width=0.4, label='DoS', align='center')

        for i, v in enumerate(norm_values):
            plt.text(i, v + max(norm_values + dos_values) * 0.01, f"{v:.1f}", ha='center',
va='bottom')

        for i, v in enumerate(dos_values):
            plt.text(i + 0.4, v + max(norm_values + dos_values) * 0.01, f"{v:.1f}", ha='center',
va='bottom')

        plt.xticks([p + 0.2 for p in x], labels)
        plt.ylabel('Значення')
        plt.title(f'Протокол {proto}: Порівняння метрик')
        plt.legend()
        plt.grid(True, linestyle='--', alpha=0.6)
        plt.show()

    protocols = ['STUN', 'DTLSv1.2', 'RTCP', 'UDP']
    plot_per_protocol(normal_df, dos_df, protocols)
```

Додаток В

Лістинг коду системи виявлення DoS-атак у WebRTC-додатках

```
import subprocess
import time
from collections import defaultdict, deque

class Packet:
    def __init__(self, fields):
        self.time = float(fields[0]) if fields[0] else 0.0
        self.src_ip = fields[1]
        self.dst_ip = fields[2]
        self.protocol = fields[3]
        self.length = int(fields[4]) if fields[4].isdigit() else 0
        self.info = fields[5]

    def __repr__(self):
        return (f"<Packet {self.time:.6f}s | {self.protocol} | {self.src_ip} → {self.dst_ip} | "
                f"{self.length} bytes | Info: {self.info}>")

NORMAL_VALUES = {
    'STUN': {'freq': 2, 'size': 140},
    'DTLSv1.2': {'freq': 1, 'size': 206},
    'RTCP': {'freq': 10, 'size': 100},  # розмір умовний
}

THRESHOLDS = {
    'STUN': {'freq': [50, 100], 'size': [2, 5], 'unanswered': [1, 3]},
    'DTLSv1.2': {'freq': [2, 5], 'size': [2, 3]},
    'RTCP': {'freq': [2, 5], 'size': [2, 5]},
}

stats = defaultdict(lambda: defaultdict(lambda: {
    'packets': 0,
```

```
    'sizes': [],
    'times': deque(),
    'last_time': 0,
    'stun_requests': 0,
    'stun_responses': 0,
}))

def analyze_packet(packet):
    proto = packet.protocol
    if proto not in NORMAL_VALUES:
        return

    sender = packet.src_ip
    now = time.time()
    data = stats[sender][proto]

    # Update stats
    data['packets'] += 1
    data['sizes'].append(packet.length)
    data['times'].append(now)
    data['last_time'] = now

    while data['times'] and now - data['times'][0] > 10:
        data['times'].popleft()

    if proto == 'STUN':
        if "Binding Request" in packet.info:
            data['stun_requests'] += 1
        elif "Binding Success" in packet.info:
            response_data = stats[packet.dst_ip][proto]
            response_data['stun_responses'] += 1

    if packet.time >= 10:
            check_thresholds(sender, proto)
```

```
        def check_thresholds(ip, proto):
            data = stats[ip][proto]
            normal = NORMAL_VALUES[proto]
            thresh = THRESHOLDS[proto]

            duration = max(1, data['times'][-1] - data['times'][0])  # сек
            freq = len(data['times']) / duration
            check_level(freq, normal['freq'], thresh['freq'], f"[{ip}] Частота {proto}: {freq:.2f}
pkt/s")

            avg_size = sum(data['sizes']) / len(data['sizes']) if data['sizes'] else 0
            check_level(avg_size, normal['size'], thresh['size'], f"[{ip}] Розмір {proto}:
{avg_size:.1f} байт")

            if proto == 'STUN':
                unanswered = data['stun_requests'] - data['stun_responses']
                check_level(unanswered, 5, thresh['unanswered'], f"[{ip}] STUN без відповіді:
{unanswered} запитів")

        def check_level(value, norm, thresholds, label):
            warn_k, danger_k = thresholds
            if value > norm * danger_k:
                print(f" ❗ ЗАГРОЗА! {label}")
            elif value > norm * warn_k:
                print(f" ⚠️ Попередження: {label}")

        def start_tshark_interface(interface='enp0s3'):
            tshark_cmd = [
                "tshark",
                "-i", interface,
                "-Y", "(stun || rtp || dtls || rtcp) && udp.port == 10000",
                "-T", "fields",
                "-e", "frame.time_relative",
```

```
            "-e", "ip.src",
            "-e", "ip.dst",
            "-e", "_ws.col.Protocol",
            "-e", "frame.len",
            "-e", "_ws.col.Info"
        ]

        process = subprocess.Popen(tshark_cmd, stdout=subprocess.PIPE,
stderr=subprocess.DEVNULL, text=True)
        print(f"🔍 IDS працює з tshark на інтерфейсі {interface}...\n")

        for line in process.stdout:
            try:
                fields = line.strip().split('\t')
                if len(fields) != 6:
                    continue
                packet = Packet(fields)
                analyze_packet(packet)
            except Exception as e:
                continue

    if __name__ == '__main__':
        start_tshark_interface('enp0s3')
```

Додаток Г

Апробація результатів дослідження

Дружко Денис, Юрій Бабенко. Дослідження вразливості технології WebRTC до DoS-атак. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSICS)». 2025, Київ, Україна. С. 140 – 141.