КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від _____, протокол №___.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«Дослідження методів та технологій захисту від DDOS-атак»

Виконав:
студент 4-го курсу
денної форми навчання
спеціальності 172 - Телекомунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»
Видрик Ігор Михайлович _____

Науковий керівник:
кандидат фіз.-мат. наук, асистент
Богданов Роман Вікторович _____

Рецензент:
кандидат технічних наук, асистент
кафедри комп'ютерної інженерії
Слюсар Євген Андрійович _____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____ Видрик Ігор

Київ – 2025

Реферат

Пояснювальна записка: 73 с., 12 рисунків, 7 таблиць, 18 джерел.

Об’єкт дослідження – процес атак типу DDoS та засоби їх виявлення і запобігання на рівні веб-застосунків.

Предмет дослідження – методи фільтрації запитів, алгоритми виявлення аномальної активності, технології обмеження трафіку, а також механізми прикладного рівня захисту.

Мета дипломного проєкту – розробка прикладного модуля фільтрації вхідного трафіку, який дозволяє виявляти аномальну активність користувачів, обмежувати кількість запитів за IP-адресою, фіксувати підозрілу поведінку та здійснювати базову валідацію користувача з метою запобігання DDoS-атакам.

У процесі виконання роботи досліджено найбільш поширені методи DDoS-атак, проаналізовано механізми захисту на прикладі рівня HTTP, розглянуто практики обмеження частоти звернень та підтвердження дій користувача. На основі отриманих результатів реалізовано серверний модуль, що виконує логування запитів, виявлення надмірної активності, занесення IP до списку підозрілих та ініціювання перевірки у разі виявлення аномалії. Система забезпечує зручне тестування функціоналу та базову інтеграцію з вебзастосунками. Розробку виконано із застосуванням PHP, HTML/CSS та MySQL без використання фреймворків, що забезпечує прозорість та контроль над кожним етапом захисту.

Актуальність теми обумовлена постійним зростанням частоти та складності DDoS-атак, які залишаються серйозною загрозою для вебресурсів різного масштабу. Реалізація ефективних засобів первинного захисту може значно знизити навантаження на інфраструктуру та мінімізувати ризики відмови в обслуговуванні.

Ключові слова: DDoS, ЗАХИСТ ВІД АТАК, HTTP FLOOD, PHP, MYSQL, CAPTCHA, ВЕБ-СЕРВЕР, IP-АДРЕСА, ЗАПИТИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DDoS – Distributed Denial of Service (Розподілена атака на відмову в обслуговуванні)

IP – Internet Protocol (Інтернет-протокол)

HTTP – HyperText Transfer Protocol (Протокол передавання гіпертексту)

URL – Uniform Resource Locator (Уніфікований локатор ресурсу)

SQL – Structured Query Language (Мова структурованих запитів)

PHP – Hypertext Preprocessor (Серверна мова сценаріїв)

MySQL – Система керування реляційними базами даних з відкритим кодом

URI – Uniform Resource Identifier (Уніфікований ідентифікатор ресурсу)

CAPTCHA – Completely Automated Public Turing test to tell Computers and Humans Apart (Автоматичний тест для розрізнення людей і комп’ютерів)

DB – Database (База даних)

GET – Метод HTTP-запиту для отримання даних із сервера

POST – Метод HTTP-запиту для надсилання даних на сервер

AJAX – Asynchronous JavaScript and XML (Асинхронний обмін даними між клієнтом і сервером)

UI – User Interface (Користувацький інтерфейс)

DoS – Denial of Service (Атака на відмову в обслуговуванні)

TTL – Time To Live (Час життя мережевого пакету або сесії)

ВСТУП

У сучасному науково-технічному середовищі значну увагу приділяють розробці ефективних механізмів забезпечення безпеки веб-систем. Однією з найбільш актуальних загроз є атаки типу DDoS (розподілена відмова в обслуговуванні), що мають на меті виведення з ладу інтернет-ресурсу шляхом надмірного навантаження на його серверну інфраструктуру. Подібні атаки можуть спричинити не лише фінансові втрати, а й повну зупинку роботи критично важливих сервісів, що робить завдання виявлення та нейтралізації DDoS-активності першочерговим у сфері інформаційної безпеки. Зі зростанням кількості загроз, спричинених автоматизованими бот-мережами та зловмисними скриптами, актуальним стає впровадження гнучких рішень на прикладному рівні, які здатні динамічно аналізувати поведінку користувачів, фіксувати аномальну активність і відповідно реагувати на потенційні атаки.

Розвиток веб-технологій створює можливості для реалізації захисних систем без необхідності використання складного або зовнішнього обладнання — безпосередньо в межах веб-застосунку. Це дозволяє адаптувати захист до конкретного проєкту, забезпечити гнучкість і масштабованість. Особливої актуальності набуває створення програмного засобу, який автоматизує фіксацію підозрілих запитів, дозволяє реагувати на них у реальному часі, впроваджує CAPTCHA-перевірку, а також веде журнал активності користувачів.

Мета роботи — дослідити сучасні методи та технології захисту від DDoS-атак, а також розробити прикладний засіб протидії таким атакам на рівні веб-застосунку, що забезпечує фіксацію аномальної активності, обмеження запитів, перевірку користувачів та ведення журналу трафіку.

Для досягнення поставленої мети в роботі сформульовано та вирішено такі основні **завдання:**

1. Провести аналіз предметної галузі у сфері DDoS-атак та їх впливу на веб-ресурси.

2. Дослідити класифікацію DDoS-атак, їх ознаки, методи реалізації та шляхи виявлення.
3. Розглянути існуючі технології захисту, зокрема обмеження частоти запитів, CAPTCHA, журналювання та валідацію сесій.
4. Обґрунтувати доцільність застосування прикладного рівня захисту для малих та середніх веб-застосунків.
5. Обрати інструменти для реалізації системи захисту (PHP, MySQL, JavaScript, HTML).
6. Спроєктувати логіку фіксації запитів, виявлення підозрілих IP, перевірки користувачів та обробки запитів.
7. Реалізувати програмні модулі системи.
8. Провести тестування функціоналу на основі симульованих DDoS-сценаріїв та оцінити ефективність запропонованого рішення.

Об’єкт дослідження – процес атак типу DDoS та засоби їх виявлення і запобігання на рівні веб-застосунків.

Предмет дослідження – методи фільтрації запитів, алгоритми виявлення аномальної активності, технології обмеження трафіку, а також механізми прикладного рівня захисту.

Методи дослідження включають аналіз предметної галузі інформаційної безпеки, вивчення природи та класифікації DDoS-атак, дослідження існуючих технологій захисту, проєктування прикладного модуля протидії атакам, реалізацію програмних рішень на основі PHP та MySQL, а також тестування працездатності системи в умовах штучного навантаження.

Практичне значення одержаних результатів полягає в можливості застосування розробленого модуля як частини захисту веб-сервісів малого та середнього масштабу. Запропоноване рішення може використовуватись для обмеження підозрілої активності, автоматичного виявлення надмірного навантаження, а також інтерактивної перевірки користувача без залучення сторонніх сервісів. Система придатна як для навчальних, так і для прикладних цілей у сфері інформаційної безпеки та розробки веб-застосунків.

1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ЇЇ ТЕОРЕТИЧНИХ АСПЕКТІВ

1.1. Введення до предметної галузі

Інформаційна безпека в сучасному цифровому світі набула статусу одного з ключових елементів стабільного функціонування організацій, підприємств, державних структур і навіть суспільства в цілому. З розвитком інтернет-технологій та розширенням IT-інфраструктур зросла і кількість потенційних векторів атак, які можуть бути використані зловмисниками для порушення нормального функціонування систем.

Однією з найнебезпечніших загроз, що ставить під сумнів доступність цифрових ресурсів, є атаки типу відмови в обслуговуванні (Denial of Service — DoS), а особливо — розподілені атаки (Distributed Denial of Service — DDoS). Основна мета таких атак полягає у створенні надмірного навантаження на цільовий ресурс з метою виведення його з ладу або унеможливлення доступу для легітимних користувачів. Відповідно до моделі CIA — концепції забезпечення конфіденційності, цілісності та доступності інформації— DDoS-атаки безпосередньо спрямовані на порушення доступності. Особливість DDoS-атак полягає у використанні великої кількості пристроїв, часто об’єднаних у ботнет-мережу, які діють синхронно та координовано, генеруючи величезну кількість запитів до цільової системи [1].

У сучасних умовах такі атаки не є поодинокими випадками — вони стали масовим явищем. За даними аналітичних агентств, щороку кількість DDoS-інцидентів зростає, причому спостерігається тенденція до їх ускладнення, використання комбінованих сценаріїв, а також спрямованості не лише на бізнес-ресурси, але й на державну інфраструктуру. На рисунку 1.1 наведено узагальнену схему функціонування типового сценарію DDoS-атаки, в якій центральну роль відіграє ботнет, що координовано надсилає запити на цільовий сервер.

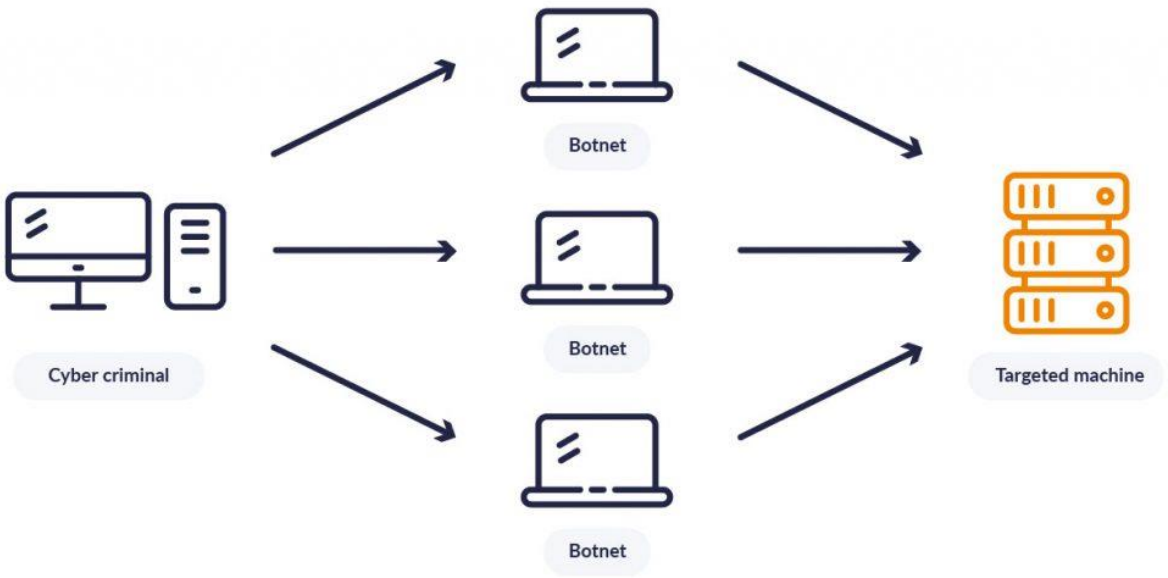

Рисунок 1.1 – Схема функціонування DDoS-атаки

У результаті цільовий сервер стає перевантаженим, що призводить до зниження продуктивності, повної недоступності сервісу або навіть до виходу з ладу мережевого обладнання. У багатьох випадках такі атаки мають катастрофічні наслідки для компаній, особливо якщо вони працюють у режимі безперервного надання онлайн-послуг. Відомі випадки, коли наслідком DDoS-атаки ставали фінансові збитки, втрата клієнтської бази, пошкодження репутації компанії та навіть перебої у роботі об'єктів критичної інфраструктури. Крім економічних і технічних збитків, такі атаки часто використовуються як інструмент політичного або конкурентного тиску. Вони можуть слугувати засобом шантажу або відволікання уваги під час інших шкідливих дій у кіберпросторі, таких як крадіжка даних чи встановлення бекдорів.

З огляду на це, дослідження DDoS-атак та відповідних методів їх класифікації, ідентифікації та нейтралізації є вкрай актуальним завданням сучасної кібербезпеки. Ґрунтовне розуміння видів, механізмів та особливостей

DDoS-атак є необхідним кроком для ефективного аналізу засобів протидії, що розглядатимуться у подальших розділах роботи [2].

1.2 Класифікація DDoS-атак за рівнями моделі OSI

У процесі аналізу DDoS-атак важливо враховувати рівень, на якому відбувається втручання у функціонування мережевої інфраструктури. Найбільш поширеним підходом до структуризації таких атак є використання семирівневої моделі OSI (Open Systems Interconnection), яка відображає логіку взаємодії між апаратним забезпеченням, протоколами передачі даних та прикладними сервісами (Рисунок 1.2).

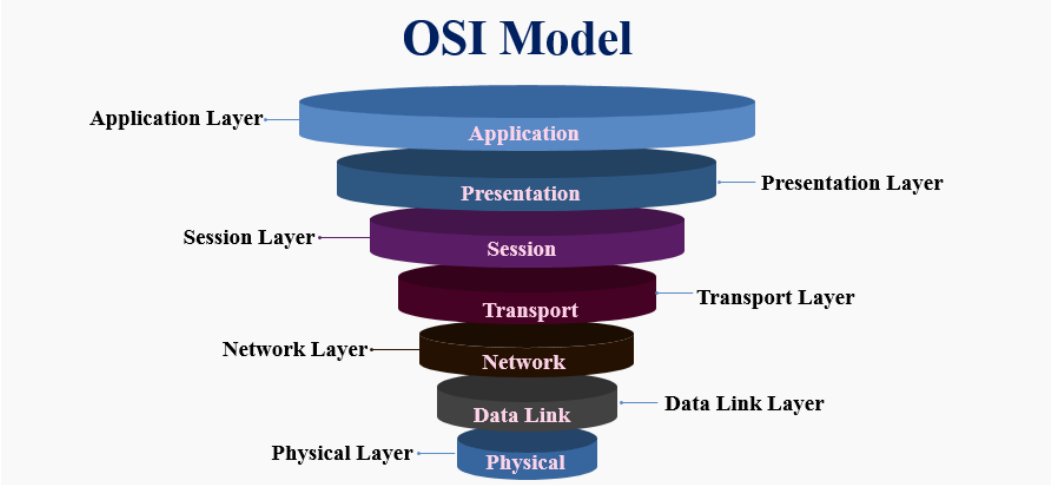

Рисунок 1.2 – Рівнева стректура моделі OSI

DDoS-атаки можуть бути спрямовані як на низькорівневу взаємодію (наприклад, мережевий чи транспортний рівень), так і на високорівневі сервіси, зокрема застосункові інтерфейси. Для точнішого розуміння механізмів порушення доступності розглянемо основні типи DDoS-атак відповідно до трьох ключових рівнів OSI-моделі: мережевого, транспортного та прикладного [3].

1.2.1 Атаки на мережевому рівні

Мережевий рівень (Network Layer) у моделі OSI відповідає за логічну адресацію та маршрутизацію даних між вузлами мережі. Він забезпечує доставку пакетів за IP-протоколом та визначає найкращі шляхи пересилання інформації. DDoS-атаки, які націлені на цей рівень, спрямовані на порушення маршрутизації, перевантаження мережевого обладнання або створення штучного трафіку, що блокує легітимну комунікацію.

Типовим прикладом втручання на цьому рівні є надсилання великої кількості ICMP-пакетів, зокрема ping-запитів, які потребують обробки з боку цільового вузла. Якщо кількість таких запитів перевищує допустиму межу, це призводить до перевантаження пропускної здатності мережі та вичерпання ресурсів пристрою. Ще одним поширеним механізмом є використання фрагментованих IP-пакетів, які ускладнюють процес їхньої обробки, вимагаючи значних обчислювальних ресурсів на їх повторне збирання. Це створює додаткове навантаження на систему та уповільнює її роботу.

Окрему загрозу становить підміна IP-адрес джерела трафіку з використанням широкомовної адресації. У таких випадках відповіді на запити спрямовуються до жертви з великої кількості пристроїв одночасно, що також призводить до мережевого перевантаження. Подібні атаки, хоч і є технічно простими у реалізації, можуть мати масштабний руйнівний ефект, особливо у випадку недостатньо захищеної інфраструктури.

Таким чином, атаки на мережевому рівні становлять значну загрозу для безперервності роботи системи, порушують логіку маршрутизації та здатні повністю заблокувати канали зв’язку між компонентами мережі [4].

1.2.2 Атаки на транспортному рівні

Транспортний рівень (Transport Layer) є четвертим у моделі OSI та відповідає за встановлення, підтримання та завершення з’єднання між

хостами. На цьому рівні працюють ключові протоколи, зокрема TCP (Transmission Control Protocol) та UDP (User Datagram Protocol), які забезпечують як надійну, так і ненадійну доставку даних. Порушення функціонування транспортного рівня шляхом DDoS-атаки призводить до дестабілізації процесу з'єднання між клієнтом і сервером, вичерпання ресурсу доступних портів або навантаження механізмів обробки сесій.

Одним з найпоширеніших векторів є зловживання процедурою встановлення TCP-з'єднання. У типовому випадку TCP використовує механізм «тристороннього рукостискання» (three-way handshake), який передбачає обмін трьома послідовними пакетами між клієнтом і сервером. DDoS-атака реалізується через надсилання великої кількості SYN-пакетів без завершення цього процесу, внаслідок чого сервер змушений резервувати ресурси для кожного непідтвердженого запиту. Це призводить до заповнення черг з'єднань і унеможливлення обслуговування легітимних користувачів.

Ще одним варіантом атаки є масове надсилання UDP-пакетів на випадкові порти цільового вузла. Через відсутність механізмів встановлення з'єднання в UDP, система змушена постійно генерувати повідомлення про недоступність портів, що створює додаткове навантаження на ресурси. Подібні атаки, хоча й не є настільки точковими, як TCP-зловживання, добре масштабуються і здатні швидко заповнити пропускну здатність або знизити продуктивність серверного обладнання.

Також на транспортному рівні можуть здійснюватись атаки з використанням фальсифікованих TCP-пакетів для примусового розриву активних з'єднань. У таких випадках зловмисник імітує повідомлення типу TCP RST, яке сприймається як сигнал до завершення сеансу зв'язку. У результаті відбувається некоректне припинення обміну даними, що дестабілізує роботу застосунків або сервісів.

DDoS-атаки на транспортному рівні є особливо небезпечними в умовах, коли інфраструктура має обмежений пул доступних з'єднань або функціонує з низькими затримками. Вони відзначаються високою ефективністю,

здатністю обходити прості фільтри та необхідністю застосування спеціалізованих механізмів захисту на рівні стеку протоколів. Навіть короткочасна атака цього типу може призвести до суттєвого зниження якості обслуговування, а в окремих випадках — до повної втрати зв’язку між компонентами системи. Це створює серйозну загрозу для сервісів, що працюють у режимі реального часу або обслуговують критичні бізнес-процеси [5].

1.2.3 Атаки на прикладному рівні

Прикладний рівень (Application Layer) є найвищим у моделі OSI й забезпечує взаємодію між користувачем та програмними сервісами. Він охоплює такі протоколи, як HTTP, HTTPS, DNS, SMTP, FTP та інші, що безпосередньо відповідають за роботу вебзастосунків, електронної пошти, обміну файлами тощо. DDoS-атаки, націлені на цей рівень, є одними з найнебезпечніших і водночас найскладніших для виявлення, оскільки за своєю структурою часто імітують звичайну активність користувача. Основна мета атак на прикладному рівні полягає у вичерпанні обчислювальних ресурсів сервера, навантаженні логіки обробки запитів або перевантаженні окремих функціональних модулів застосунку. На відміну від атак на нижчі рівні, ці дії не потребують надмірного мережевого трафіку: навіть помірна кількість складних запитів може спричинити затримки в обробці та деградацію сервісу.

Найхарактернішою формою такої атаки є HTTP flood, за якого ботнет або спеціалізовані скрипти багаторазово надсилають запити GET або POST до вебсерверу. У випадку, якщо кожен запит вимагає складної логіки на стороні сервера (наприклад, звернення до бази даних), це створює надмірне навантаження й уповільнює відповідь навіть на легітимні звернення.

Ще одним прикладом є атаки типу Slowloris, які полягають у поступовому надсиланні HTTP-заголовків з розривами у часі. Це утримує з’єднання відкритим якомога довше, вичерпуючи пул доступних з’єднань без

помітного обсягу трафіку. Подібні механізми є особливо небезпечними, оскільки вони залишаються малопомітними для класичних систем моніторингу.

Окрему загрозу становлять атаки на інші прикладні протоколи, зокрема DNS. У цьому випадку сервер навантажується великою кількістю запитів на розпізнавання доменних імен, що може призвести до тимчасової втрати доступу до вебресурсів навіть у випадку фізичної працездатності основного серверного обладнання.

Прикладнорівневі атаки є особливо ефективними проти ресурсів, орієнтованих на кінцевого користувача, таких як електронна комерція, онлайн-сервіси або банківські застосунки. Їхня складність полягає не лише в шкідливій дії, а й у складності верифікації джерела запиту, що ускладнює реалізацію заходів захисту [6].

1.3 Класифікація DDoS-атак за характером трафіку

Ще одним важливим критерієм класифікації DDoS-атак є характер трафіку, який використовується зловмисником для порушення нормального функціонування системи. Відповідно до обраного механізму впливу, атаки можуть мати різну структуру, обсяг, рівень складності та поведінкові ознаки. Це визначає як їхню ефективність, так і складність виявлення. Залежно від цього виділяють три основні типи: об’ємні атаки (volumetric), протокольні (protocol-based) та атаки на рівні застосунків (application-layer). Кожен із цих підходів має власні цілі та технічні особливості, що обумовлюють методи виявлення і захисту від них.

Для комплексного розуміння механізмів реалізації DDoS-атак, а також для подальшого вибору відповідних засобів протидії, розглянемо докладніше зазначені типи атак за характером трафіку та особливостями їх впливу на IT-інфраструктуру [7].

1.3.1 Volumetric-атаки

Volumetric-атаки (об’ємні атаки) є найпоширенішим типом DDoS-атак, суть яких полягає у створенні надмірного навантаження на пропускну здатність мережі або ресурсів обладнання за рахунок надсилання великої кількості даних у найкоротший проміжок часу. У більшості випадків такі атаки досягають своєї мети шляхом генерації трафіку, що перевищує можливості обробки цільового каналу або системи. Ключова особливість volumetric-атак полягає в тому, що вони не орієнтовані на певну логіку роботи застосунку чи сервера, а спрямовані на вичерпання мережевих або інфраструктурних ресурсів, таких як пропускна здатність каналів зв’язку, потужність комутаторів або маршрутизаторів. У результаті легітимні запити не можуть бути оброблені, оскільки канал перевантажено шкідливим трафіком.

Типовими прикладами цього виду атаки є UDP flood, ICMP flood, а також атаки з використанням технік відображення і підсилення — зокрема DNS Amplification або NTP Amplification. У таких сценаріях зловмисник надсилає невеликі запити до відкритих серверів, які у відповідь генерують великі обсяги даних і перенаправляють їх на адресу жертви. Завдяки цьому атака здатна багаторазово підсилювати вихідний трафік і досягати дуже високої інтенсивності без значного навантаження на атакувальну сторону.

Volumetric-атаки відносно прості в реалізації, однак їхня ефективність особливо висока у випадках, коли захисна інфраструктура не має механізмів балансування трафіку, географічного розподілу або автоматичного масштабування. Їхня дія може охоплювати не лише цільовий сервер, а й інші елементи мережі, що робить цей тип атак надзвичайно руйнівним для загальної інфраструктури. Ураження можуть спричинити каскадні збої, що паралізують роботу пов’язаних сервісів, зокрема — внутрішніх корпоративних систем, хмарних сервісів або CDN-мереж. Саме тому volumetric-атаки вимагають не лише локального реагування, а й залучення міжмережевої координації на рівні провайдерів і дата-центрів [8].

1.3.2 Protocol-based атаки

Protocol-based атаки (протокольні атаки) ґрунтуються на зловживанні особливостями реалізації або поведінковими характеристиками мережевих протоколів, які працюють на мережевому або транспортному рівнях. На відміну від об’ємних атак, основна мета протокольних атак полягає не стільки у перевантаженні каналів зв’язку, скільки у виведенні з ладу або дестабілізації роботи пристроїв, що забезпечують функціонування мережі — серверів, маршрутизаторів, балансувальників навантаження.

Такі атаки зазвичай здійснюються через надсилання спеціально сформованих або некоректно структурованих пакетів, які змушують систему витрачати надмірні ресурси на обробку або призводять до порушення протоколів з'єднання. Одним із найвідоміших прикладів є SYN flood — атака, при якій створюється велика кількість частково відкритих TCP-з'єднань без завершення процедури «тристороннього рукостискання». У результаті сервер виділяє пам’ять і ресурси для кожного запиту, що зрештою призводить до виснаження пулу з’єднань.

Іншим прикладом є атаки на основі зловживання UDP або ICMP-протоколами, коли система змушена відповідати на запити, які не потребують повноцінного з'єднання. У багатьох випадках протокольні атаки використовують знання про специфічні налаштування або слабкі сторони цільової інфраструктури, що робить їх особливо ефективними проти старих або неправильно сконфігурованих систем.

Попри відносно невеликий обсяг шкідливого трафіку, protocol-based атаки здатні спричинити серйозні збої в роботі сервісів, особливо якщо вони застосовуються у поєднанні з іншими типами DDoS-навантаження. Для їх своєчасного виявлення та блокування потрібне глибоке розуміння мережевої поведінки та застосування спеціалізованих інструментів аналізу трафіку в реальному часі [8].

1.3.3 Application-layer атаки

Application-layer атаки, або атаки на рівні застосунків, належать до найскладніших для виявлення та протидії типів DDoS-атак. Вони спрямовані безпосередньо на порушення роботи прикладних сервісів, що взаємодіють з кінцевим користувачем, зокрема вебсерверів, API-інтерфейсів, баз даних, електронної пошти та інших елементів програмної логіки. На відміну від об’ємних або протокольних атак, application-layer атаки зазвичай не генерують значного обсягу трафіку. Їхня ефективність полягає у створенні умов, коли навіть відносно невелика кількість шкідливих запитів змушує сервер виконувати ресурсоємні операції. У такий спосіб досягається поступове вичерпання обчислювальних ресурсів, зменшення швидкості обробки легітимних запитів або повна недоступність сервісу.

Однією з типових форм цього виду атаки є HTTP flood, під час якого ботнет або розподілені скрипти багаторазово надсилають запити до вебресурсу. Особливої шкоди завдають динамічні запити, що передбачають обробку даних, звернення до бази даних або генерацію складного контенту на стороні сервера. В результаті ресурси серверної частини вичерпуються, навіть якщо пропускна здатність мережі залишається в межах норми.

Ще одним прикладом є повільні атаки типу Slowloris, що полягають у поетапному надсиланні HTTP-заголовків із тривалими паузами між ними. Це змушує сервер довгий час утримувати відкриті з’єднання, блокуючи тим самим можливість обслуговування нових запитів. Інші варіанти можуть передбачати зловживання DNS- або SMTP-протоколами, особливо у випадках, коли вони працюють без належної авторизації чи обмежень.

Особливу складність у протидії таким атакам становить їхня подібність до звичайної користувацької активності. Багато запитів у рамках атаки мають коректну структуру, використовують легітимні протоколи і не викликають підозри з боку стандартних систем захисту. Саме тому ефективне виявлення application-layer атак вимагає впровадження інтелектуальних алгоритмів

поведінкового аналізу, використання Web Application Firewall (WAF), систем оцінки навантаження та моніторингу запитів у реальному часі [8].

1.4 Класифікація DDoS-атак за рівнем автоматизації

У сучасних умовах DDoS-атаки суттєво різняться не лише за технічними характеристиками, а й за способом організації та ступенем залучення автоматизованих засобів. Саме рівень автоматизації визначає масштаб, складність координації дій зловмисника та ефективність протидії атаці з боку захисної інфраструктури. Залежно від того, якою мірою в атаці беруть участь автоматизовані засоби — скрипти, ботнети, зловмисне ПЗ або навіть системи на основі штучного інтелекту — можна виділити три основні типи: ручні атаки, частково автоматизовані та повністю автоматизовані. Кожен із цих варіантів має власну архітектуру, способи запуску та супутні ризики. Для кращого розуміння розглянемо кожен тип докладніше [9].

1.4.1 Ручні атаки

Ручні DDoS-атаки є найпростішою формою організації відмови в обслуговуванні, яка не передбачає використання автоматизованих або масштабованих засобів. Такі атаки зазвичай здійснюються окремими особами або невеликими групами зловмисників вручну, через стандартні мережеві інструменти або спеціалізовані утиліти (наприклад, LOIC — Low Orbit Ion Cannon). Основною особливістю цього типу є обмежений обсяг трафіку, що генерується, а також низька складність реалізації. При цьому ефективність ручної атаки здебільшого незначна, якщо цільова система має бодай мінімальний рівень захисту або працює в масштабованому середовищі.

Проте вразливі або ізольовані системи можуть стати легкою мішенню навіть у таких умовах. У типовій схемі ручної DDoS-атаки (Рисунок 1.3) відображено просту взаємодію: зловмисник безпосередньо надсилає запити на

цільовий сервер, імітуючи законну активність, але з надмірною частотою. Відсутність проміжного рівня автоматизації робить таку атаку легко відстежуваною, але потенційно небезпечною у випадках, коли інфраструктура є незахищеною або не передбачає обмежень на з'єднання.

Рисунок 1.3 – Схема типової ручної DDoS-атаки

Таким чином, ручні DDoS-атаки, попри свою простоту, залишаються актуальними в умовах недостатньо захищених середовищ або на початкових етапах багатовекторних атак. Їхнє вивчення дозволяє краще зрозуміти базові принципи впливу на доступність системи, які в подальшому масштабуються і ускладнюються в рамках більш автоматизованих сценаріїв впливу [10].

1.4.2 Частково автоматизовані атаки

Частково автоматизовані DDoS-атаки поєднують ручне управління з використанням спеціалізованих програмних засобів, скриптів або обмежених бот-мереж, які виконують базові інструкції зловмисника. Такий підхід дозволяє значно підвищити масштаб і ефективність атаки без повної автоматизації процесів, зберігаючи при цьому певний ступінь контролю з боку оператора. Характерною ознакою частково автоматизованих атак є залучення кількох комп'ютерів або пристроїв, які працюють за однаковим сценарієм і

підключаються до цільової системи для генерації трафіку або зловживання протоколами. Керування здійснюється централізовано, але із застосуванням простих інтерфейсів — наприклад, запуску скриптів із попередньо заданими параметрами або використання GUI-утиліт для запуску однотипних сесій з кількох пристроїв.

Такі атаки мають вищу інтенсивність порівняно з ручними, проте все ще обмежені масштабами локального середовища або кількістю доступних виконавчих пристроїв. Вони ефективні в атаках на невеликі організації, внутрішні системи або в рамках фазового впливу перед розгортанням масштабнішої DDoS-кампанії. На схемі частково автоматизованої атаки (Рисунок 1.4) зображено базову архітектуру: оператор через керуючий інтерфейс активує декілька пристроїв або скриптів, які паралельно надсилають запити до цільової інфраструктури.

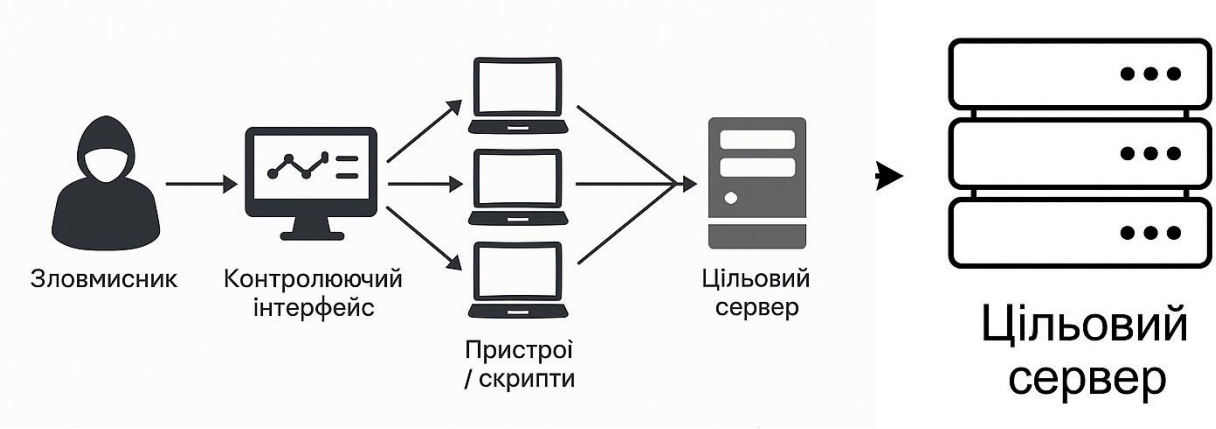

Рисунок 1.4 – Схема частково автоматизованої DDoS-атаки

Таким чином, частково автоматизовані DDoS-атаки становлять проміжну загрозу між примітивними ручними впливами та повноцінними ботнет-атаками. Вони дозволяють зловмиснику здійснювати скоординований тиск на систему з використанням мінімального набору ресурсів, і водночас залишають за ним можливість оперативного втручання та зміни сценарію атаки в реальному часі [10].

1.4.3 Повністю автоматизовані атаки

Повністю автоматизовані DDoS-атаки є найбільш небезпечною та масштабованою формою організації розподіленого навантаження на інформаційні системи. Їхня особливість полягає у повній відсутності необхідності прямої участі людини після ініціалізації процесу. Вся логіка атаки виконується через ботнети, зловмисне програмне забезпечення або інфраструктури типу «DDoS-as-a-Service», які доступні на чорному ринку у вигляді платформи за підпискою. Ці атаки здійснюються за допомогою великої кількості інфікованих пристроїв — комп'ютерів, серверів, маршрутизаторів, а останнім часом — і IoT-пристроїв. Управління здійснюється через командно-контрольні сервери, а іноді — навіть через децентралізовані протоколи, що ускладнює ідентифікацію джерела атаки. Завдяки автоматизації процесів, такі атаки здатні динамічно змінювати структуру трафіку, адаптуватися до змін у захисті й комбінувати різні техніки — від volumetric до application-layer сценаріїв.

Завдяки гнучкій архітектурі та модульному управлінню, такі атаки можуть розгортатися майже миттєво — за кілька хвилин після активації платформи. Зловмиснику не потрібно безпосередньо взаємодіяти з виконавчими пристроями, оскільки всі дії делегуються попередньо налаштованій інфраструктурі. Більше того, багато сервісів «DDoS-as-a-Service» передбачають можливість вибору типу атаки, її тривалості, цільового регіону або навіть рівня «обходу захисту», що наближає цей процес до комерційної послуги в кримінальному середовищі. У схемі повністю автоматизованої атаки (Рисунок 1.5) відображено типову архітектуру: зловмисник ініціює атаку через керуючий сервер або спеціалізовану платформу, яка в автоматичному режимі розгортає шкідливу активність тисяч ботів проти обраної цілі.

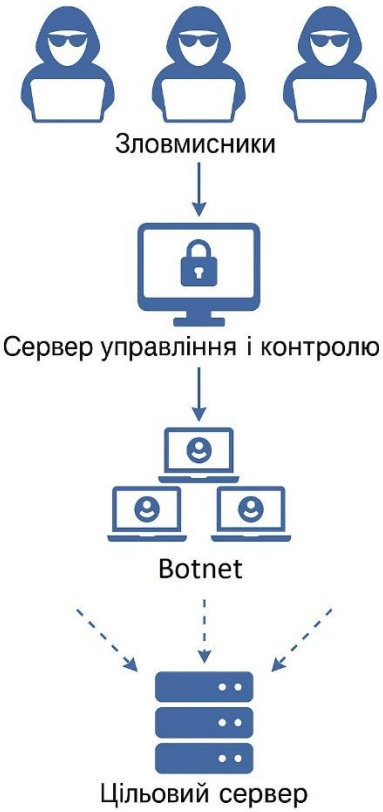

Рисунок 1.5 – Схема повністю автоматизованої DDoS-атаки

Таким чином, повністю автоматизовані DDoS-атаки є прикладом високотехнологічного впливу на інформаційні системи, що поєднує розподіленість, гнучкість і масштабованість. Їхня складність та ефективність вимагають наявності розвиненої системи кіберзахисту, а також постійного моніторингу мережевої активності на основі інтелектуального аналізу та кореляції подій [10].

1.5 Наслідки DDoS атак

DDoS-атаки становлять не лише технічну загрозу, але й мають суттєві соціальні, економічні та правові наслідки, що виводить проблему поза межі IT-сфери. З кожним роком масштаби таких інцидентів зростають, і разом із цим — обсяг завданих збитків та складність відновлення діяльності постраждалих структур.

З економічної точки зору, DDoS-атаки можуть призвести до значних фінансових втрат. Серед основних джерел збитків — простої онлайн-сервісів, втрати клієнтів через відсутність доступу до платформи, витрати на аварійне масштабування інфраструктури, а також подальші інвестиції в покращення системи захисту. У випадках атак на електронну комерцію або фінансові установи навіть кілька годин простою можуть означати втрату мільйонів доларів. Згідно з аналітичними звітами, середній розмір фінансових втрат від однієї потужної DDoS-атаки для середнього бізнесу може коливатись від десятків до сотень тисяч доларів.

Соціальні наслідки проявляються у втраті довіри з боку користувачів до цифрових сервісів, особливо якщо атака спричинила довготривалу недоступність, або ж стала публічним інцидентом у ЗМІ. DDoS-атаки на сервіси державного призначення — електронне урядування, портали публічних послуг, системи електронного голосування — можуть спричинити дестабілізацію в суспільстві, паніку або підрив авторитету державних інституцій.

З правової точки зору, проведення DDoS-атак є кримінальним правопорушенням у більшості країн світу. У той же час, розслідування таких інцидентів стикається з проблемами міжнародної юрисдикції, анонімності ботнетів, складності відстеження джерела атаки та відсутності правових механізмів для ефективної екстрадиції кіберзлочинців. Особливо це ускладнюється в умовах використання децентралізованих систем управління атаками, проксі-інфраструктур і мереж типу Tor.

Таким чином, наслідки DDoS-атак значно виходять за межі тимчасової втрати доступу до сервісів. Вони зачіпають економіку, довіру, кібербезпеку держави та актуалізують потребу у формуванні міждержавної політики кіберзахисту на глобальному рівні [11].

2. ДОСЛІДЖЕННЯ МЕТОДІВ ТА ТЕХНОЛОГІЙ ЗАХИСТУ ВІД DDOS-АТАК

2.1 Аналіз існуючих підходів до протидії DDoS-атакам

На сучасному етапі розвитку інформаційних технологій DDoS-атаки перетворились із поодиноких спроб навмисного перевантаження серверів на масовий інструмент цифрового тиску, доступний як професійним злочинним угрупованням, так і окремим ентузіастам. У зв'язку з цим актуальність пошуку дієвих засобів протидії стрімко зростає, і на практиці формується цілий спектр підходів, які відрізняються за принципом дії, рівнем впливу на трафік, ресурсною вартістю та ефективністю в різних сценаріях.

Загалом, сучасні стратегії захисту орієнтовані не лише на блокування шкідливого трафіку, а й на побудову системи раннього виявлення аномалій, гнучкої адаптації до змін у поведінці атакувального середовища, а також на здатність мінімізувати шкоду від атаки у найкоротші терміни. Різні техніки при цьому можуть працювати як на рівні мережевого обладнання, так і на рівні прикладних сервісів, або ж бути делеговані стороннім хмарним провайдерам. Умовно їх можна поділити на такі, що випереджають атаку (превентивні), виявляють її (детектувальні) або знижують її наслідки (реактивні), однак на практиці чіткі межі між цими категоріями часто розмиваються. Сучасні рішення все частіше поєднують у собі кілька механізмів. Наприклад, у рамках єдиної архітектури можуть одночасно застосовуватись обмеження частоти запитів, географічне блокування IP-адрес, аналіз поведінкових патернів, глибока перевірка пакетів, а також автоматизоване перенаправлення трафіку через так звані scrubbing-сервери, здатні «очищати» потік від зловмисного навантаження. Цей підхід, що передбачає багаторівневу фільтрацію, є більш стійким до комбінованих або тривалих атак, однак вимагає значних ресурсів як у плані обчислювальної потужності, так і у сфері адміністрування [12].

Одним з ключових трендів у цій сфері є впровадження систем, що використовують штучний інтелект і машинне навчання для аналізу нетипових змін у трафіку. Вони дозволяють здійснювати самонавчання системи на основі історичних даних, автоматично ідентифікуючи нові види атак без потреби

ручного втручання. Такі системи, хоча й потребують початкової підготовки та відповідного набору даних, демонструють високу адаптивність та потенціал до масштабування.

У контексті вебзастосунків поширення отримали такі інструменти, як веб-фаєрволи нового покоління (WAF), які інтегруються з інфраструктурою сайту і забезпечують динамічну фільтрацію трафіку на основі правил, поведінки користувача або сигнатур атак. Вони часто поєднуються з додатковими модулями автентифікації, використанням CAPTCHA, системами обмеження доступу за User-Agent або реферером, а також з CDN-платформами, що дозволяють зменшити пряме навантаження на основні сервери.

Окрему категорію складають рішення, реалізовані на рівні інтернет-провайдерів або операторів хмарних послуг. Такі підходи забезпечують фільтрацію ще до того, як трафік досягне цільової інфраструктури. Вони є ефективними у випадках масованих атак, однак часто вимагають додаткових фінансових вкладень та не завжди дозволяють швидко вносити кастомізовані зміни на рівні кінцевого користувача.

З огляду на різноманіття існуючих рішень та їх неоднакову ефективність залежно від типу і складності атаки, важливо не лише розуміти загальну класифікацію підходів, а й дослідити конкретні технічні методи, що застосовуються на практиці. Для розуміння їхньої ефективності та доцільності використання у конкретних умовах необхідно більш детально розглянути ключові технології та методи, що становлять основу практичної реалізації захисту — від фільтрації трафіку до поведінкових механізмів валідації користувача [13].

2.2 Технології фільтрації та обмеження трафіку

Одним з базових напрямів захисту від DDoS-атак є фільтрація трафіку, що надходить до серверної інфраструктури. Цей підхід ґрунтується на

попередньому аналізі вхідних пакетів і застосуванні визначених правил для їхнього дозволу, обмеження або блокування. У рамках даної технології можуть використовуватись різні фільтри — як на рівні міжмережевих екранів (firewall), так і в рамках більш гнучких рішень, таких як системи виявлення аномалій (IDS/IPS), DPI-фільтрація (Deep Packet Inspection), або навіть CDN-сервіси із вбудованим аналізом поведінки користувачів.

Обмеження трафіку — це механізм, який дозволяє зменшити навантаження на інфраструктуру за рахунок введення квот, обмежень за кількістю з'єднань з одного IP-адресу, обмеження частоти запитів або швидкості передавання даних. Такий підхід не лише знижує ймовірність перевантаження ресурсів, але й ускладнює реалізацію DDoS-атак на прикладному рівні, зменшуючи шанси на досягнення цілей зловмисника. Часто ці технології поєднуються із системами rate limiting, які реалізуються як на стороні веб-серверів, так і в межах API-шлюзів, з можливістю гнучкого налаштування порогових значень та реакцій на перевищення лімітів. Завдяки цьому вдається оперативно реагувати на аномальну активність, яка має ознаки DDoS-навантаження, а також забезпечити підтримку стабільного рівня обслуговування легітимних користувачів навіть у випадках підвищеного навантаження. Більш складні реалізації включають використання адаптивних алгоритмів, які можуть змінювати обмеження в реальному часі, враховуючи поведінку трафіку, історичні дані або контекст запитів.

На рисунку 2.1 наведено типовий приклад архітектури багаторівневої фільтрації трафіку, яка реалізує розподіл запитів за ступенем довіри. Перший рівень фільтрації — базовий firewall — відсікає некоректно сформовані або підозрілі пакети. Далі працює інтелектуальний аналізатор, який на основі історичних даних або вбудованих евристик визначає ймовірну шкідливу активність. Заключним етапом є системи контролю навантаження, які застосовують ліміти та пріоритети для трафіку [14].

Рисунок 2.1 – Архітектура багаторівневої фільтрації вхідного трафіку

Впровадження ефективних рішень фільтрації та обмеження трафіку вимагає усвідомленого вибору між різними підходами, з урахуванням їхньої специфіки, відповідності до рівня мережевої моделі, а також технічних і ресурсних обмежень організації. Кожна з технологій має свої сильні сторони й потенційні недоліки, які можуть виявлятися лише за конкретних умов використання. Тому важливо не лише знати теоретичні особливості методів, а й розуміти їх практичну доцільність, масштабованість та вплив на загальну безпеку й продуктивність системи. Для більш глибокого розуміння практичних відмінностей між цими методами доцільно порівняти ключові технології за низкою параметрів, таких як гнучкість налаштування, ефективність виявлення атак, вплив на продуктивність системи та складність впровадження. У таблиці 2.1 представлено порівняльний аналіз основних технологій фільтрації трафіку, який дозволяє оцінити переваги й обмеження кожного методу з точки зору їх застосування для запобігання DDoS-атакам.

Таблиця 2.1 – Порівняльна характеристика методів фільтрації та обмеження трафіку

Метод	Рівень OSI	Переваги	Недоліки	Вплив на прод - ність
Міжмережевий екран (Firewall)	Мережевий	Швидка фільтрація, проста конфігурація	Низька точність при складних атаках	Незначний
Системи виявлення та запобігання атак (IDS/IPS)	Усі рівні	Інт. аналіз, гнучкість	Висока потреба в ресурсах	Суттєвий
DPI (глибокий аналіз пакетів)	Прикладний	Детальний контроль трафіку	Затримка, складність впроваджень	Високий
Rate Limiting	Прикладний	Ефективність проти сплесків активності	Можливі помилкові блокування	Незначний
Geo-фільтрація	Мережевий	Захист від специфічних регіонів	Неефективна проти проксі	Незначний
CDN з анти-DDoS (Cloudflare, Akamai)	Прикладний	Захист на краю мережі	Залежність від сторонніх сервісів	Незначний

Як видно з таблиці 2.1, кожен метод має власне цільове призначення та сферу ефективного застосування. Наприклад, міжмережеві екрани забезпечують базову фільтрацію, але часто неспроможні самостійно протистояти складним атакам, які змінюють патерни трафіку. Водночас DPI і

IDS/IPS демонструють високу гнучкість і точність, однак потребують значних обчислювальних ресурсів і часто впливають на затримки в обробці запитів.

Рішення на основі rate limiting добре підходять для захисту RESTful API або вебсервісів з передбачуваним навантаженням, але їх потрібно ретельно налаштовувати для уникнення хибнопозитивних блокувань. Географічна фільтрація, хоч і обмежена у застосуванні, ефективна для організацій, що працюють у чітко визначених регіонах. Нарешті, CDN-сервіси надають готові хмарні рішення з вбудованим захистом, що дозволяє зменшити ризики без значного навантаження на локальну інфраструктуру.

Таким чином, вибір конкретної технології має базуватись на комплексній оцінці загроз, можливостей інфраструктури та вимог до доступності й часу відгуку сервісу. У наступних підрозділах будуть розглянуті додаткові механізми протидії DDoS-атакам, зокрема — технології валідації користувача та практики реалізації захисту на стороні вебзастосунків [15].

2.3 Використання методів валідації користувача

У межах прикладного рівня захисту одним із найефективніших підходів до зменшення впливу DDoS-атак є впровадження механізмів валідації користувача. Такі засоби дозволяють диференціювати запити, що надходять від реальних користувачів, від автоматизованих систем або ботів, і, відповідно, блокувати чи уповільнювати останні. Найчастіше для цього використовуються CAPTCHA, JavaScript-челленджі, перевірка cookies, обмеження за HTTP-заголовками або токенами сесії.

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) є найпоширенішим інструментом боротьби з ботами, що дозволяє оцінити здатність користувача взаємодіяти з інтерфейсом. Візуальні, звукові або інтерактивні форми CAPTCHA широко використовуються на етапі автентифікації або при підозрі на аномальну активність. Основною перевагою цього підходу є його універсальність, що забезпечує широку інтеграцію у

більшість сучасних вебплатформ. Однак сучасні ботнети здатні обходитись навіть із складними CAPTCHA, використовуючи сервіси розпізнавання або headless-браузери, що знижує ефективність методу у випадках масштабованих атак. Крім того, надмірне використання CAPTCHA може негативно впливати на користувацький досвід, особливо для мобільних користувачів або осіб з обмеженими можливостями.

JS-челленджі (JavaScript Challenge) дозволяють на стороні клієнта виконати базові обчислення або затримки, що для ботів із вимкненим JavaScript є непереборною перешкодою. Така перевірка зазвичай є прозорою для легітимного користувача, але дозволяє значно знизити навантаження на серверну частину, блокуючи найпростіші форми автоматизованого сканування чи DDoS-запитів. У деяких випадках доцільним є поєднання кількох методів: наприклад, застосування JS-челленджу перед CAPTCHA, що дозволяє зменшити кількість показів останньої лише для тих, хто не виконав перший крок. Крім того, валідація cookies або токенів сесії дозволяє визначати повторювану підозрілу активність з одного клієнта, не обтяжуючи при цьому користувацький досвід. Такі комбіновані підходи забезпечують динамічну адаптацію до змін у поведінці потенційного зловмисника та дозволяють оптимально збалансувати безпеку з продуктивністю. Особливо ефективним є застосування цих методів у високонавантажених системах або сервісах з відкритим доступом, де неможливо заздалегідь ідентифікувати всі ризики [16].

На рисунку 2.2 наведено типовий сценарій багаторівневої валідації користувача під час надходження HTTP-запиту до вебресурсу. Зображення ілюструє поетапну перевірку запиту — спочатку на наявність JavaScript-результату, потім — на коректність cookies, і лише в разі сумнівів — показ CAPTCHA.

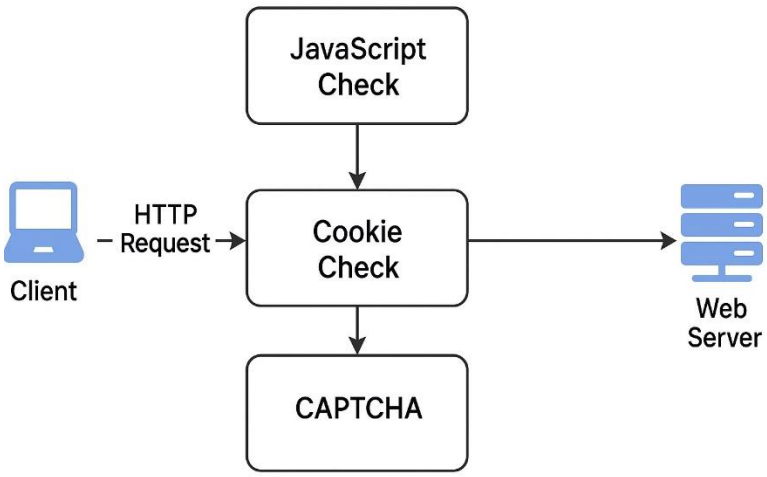

Рисунок 2.2 – Схема багаторівневої валідації користувача на прикладному рівні

У рамках аналітичного дослідження доцільно здійснити порівняльний аналіз найбільш поширених методів валідації, що застосовуються у сучасних веб-системах. Для цього охарактеризуємо їхню ефективність, зручність для користувача та типові сфери використання. Зведені дані подано у таблиці 2.2.

Таблиця 2.2 – Порівняння методів валідації користувача у контексті захисту від DDoS-атак

Метод	Ефективність проти ботів	Зручність для користувача	Типові сфери застосування
CAPTCHA	Висока	Низька	Форми, логіни, підтвердження дій
JavaScript Challenge	Середня	Висока	Захист сторінок
Перевірка cookie	Низька	Висока	Статичні ресурси, API-запити

Продовження таблиці 2.2

Токенізація сесій	Середня	Висока	Платіжні системи, REST API
Аналіз заголовків HTTP	Низька	Висока	Виявлення нетипових клієнтів, бот-сканерів

Як видно з таблиці 2.2, кожен із розглянутих методів валідації має як переваги, так і обмеження, що визначають доцільність його використання в конкретному контексті. CAPTCHA, попри свою високу ефективність у боротьбі з ботами, дедалі частіше стає джерелом незручностей для користувачів. Особливо це стосується мобільних пристроїв, де складність введення символів, вибору зображень чи проходження інтерактивних перевірок помітно ускладнює доступ до контенту. З іншого боку, CAPTCHA залишається надійним бар'єром у ситуаціях, коли потрібно захистити критичні точки доступу, наприклад, сторінки авторизації або реєстрації.

JavaScript-челленджі, які часто реалізуються через відкладене завантаження контенту або перенаправлення, дозволяють відсіювати найпримітивніші автоматизовані запити, не втручаючись у досвід реального користувача. Проте варто враховувати, що сучасні боти вже здатні емулювати базову поведінку браузера, зокрема виконання скриптів, що знижує загальний рівень захисту цього методу.

Методи, засновані на перевірці HTTP-заголовків або наявності cookie, не потребують взаємодії з користувачем і можуть бути ефективними для фільтрації найпростіших загроз, зокрема сканерів або некоректних запитів. Однак у випадку цілеспрямованих DDoS-атак ці механізми є недостатніми, оскільки сучасні інструменти дозволяють легко підробляти відповідні поля або імітувати поведінку реального браузера. Окрему увагу слід приділити токенізації сесій — механізму, що передбачає динамічне створення унікальних маркерів доступу, які потрібно подавати разом із кожним запитом. Цей підхід ефективний у довготривалих сесіях, однак вимагає належної організації на

рівні бекенду та інфраструктури, а також додаткової валідації життєвого циклу токена.

У підсумку, жоден із механізмів не є універсальним. Для досягнення високої стійкості до DDoS-атак доцільним є комбінування різних типів валідації залежно від контексту використання, навантаження на систему та критичності доступу. Такий підхід дозволяє не лише зменшити ризик проникнення ботів, але й зберегти прийнятний рівень користувацького досвіду, що є особливо важливим для публічних вебзастосунків [17].

2.4 Метод виявлення DDoS-атак на основі аналізу аномального трафіку

Один із найперспективніших підходів до виявлення DDoS-атак ґрунтується на ідеї аналізу аномального трафіку. На відміну від класичних методів, які покладаються на статичні правила чи обмеження (наприклад, за кількістю запитів на IP), системи цього типу орієнтуються на динамічний профіль нормальної поведінки користувачів. Атака виявляється тоді, коли вхідний трафік починає відхилятися від цього профілю за певними критеріями: частотою, обсягом, структурою HTTP-запитів, географією джерел тощо.

Системи виявлення на основі аномалій часто реалізуються як частина IDS/IPS (Intrusion Detection/Prevention Systems) і можуть бути побудовані на базі машинного навчання, статистичних методів або комбінованих евристик. До найбільш поширених підходів належать аналіз тимчасових рядів (Time Series Analysis), кластеризація запитів за схожістю, використання нейронних мереж для виявлення складних патернів трафіку. Сигнатурні методи, які використовують попередньо відомі шаблони DDoS-атак, поступаються за гнучкістю поведінковим моделям, але часто застосовуються в парі з ними — для покриття як відомих, так і нових загроз. Завдяки цьому системи виявлення на основі аномалій мають високу ефективність у виявленні Zero-Day атак та адаптивних ботнетів, які змінюють свою поведінку під час атаки.

На рисунку 2.3 схематично зображено принцип роботи методу виявлення DDoS-атак на основі аналізу аномального трафіку. Потік вхідних запитів аналізується через модулі збору метрик, аномалійного детектора, і далі — передається до систем прийняття рішень та блокування або маркування підозрілих сесій.

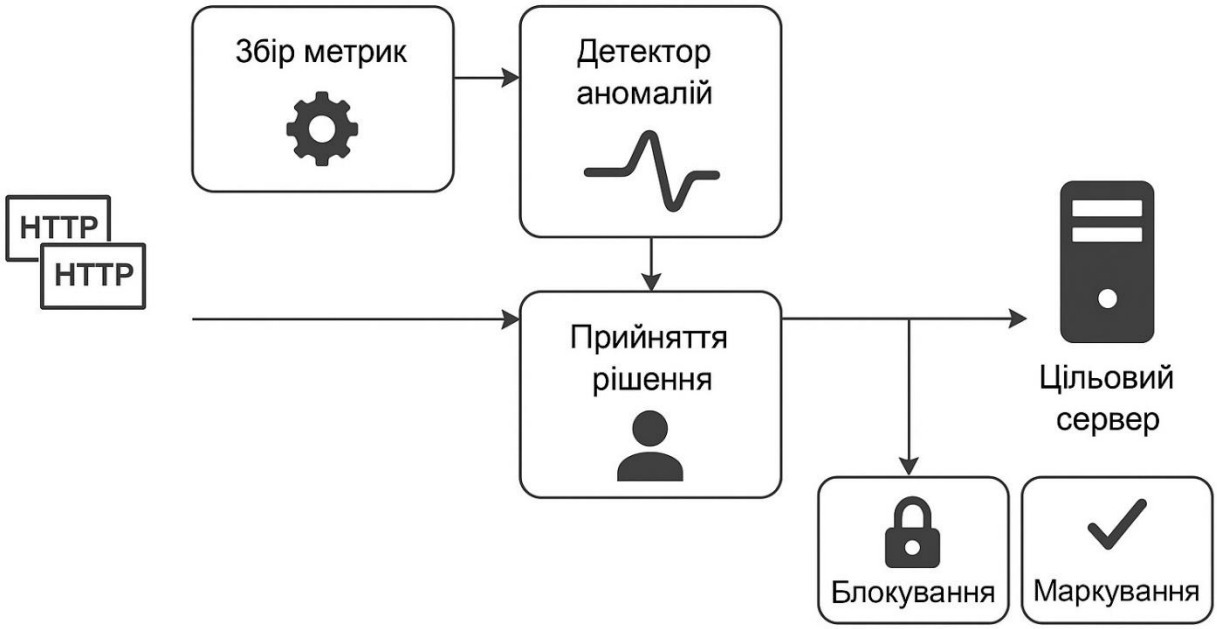

Рисунок 2.3 – Структура методу виявлення DDoS на основі аналізу аномального трафіку

Для забезпечення ефективного виявлення аномального трафіку у межах DDoS-атак важливо розуміти, які підходи доступні на практиці, та як саме вони відрізняються за своїм функціональним призначенням, точністю, ресурсомісткістю й адаптивністю. Кожен метод має свою сферу застосування — від швидкої реакції на відомі шаблони до виявлення складних відхилень у поведінці клієнтів, що не можуть бути виявлені за допомогою статичних фільтрів.

Для кращого розуміння переваг і обмежень сучасних підходів до виявлення DDoS-атак на основі аномального трафіку наведемо порівняльну таблицю, яка систематизує основні характеристики популярних методів (Таблиця 2.3).

Таблиця 2.3 – Порівняння методів виявлення аномального трафіку

Метод	Переваги	Недоліки
Сигнатурний аналіз	Швидка реакція на відомі атаки	Неефективний проти нових атак
Аналіз тимчасових рядів	Виявлення раптових відхилень у навантаженні	Вимагає багато історичних даних
Кластеризація запитів	Групування схожих шаблонів для аналізу	Складність інтерпретації результатів
Рішення на базі ML	Адаптивність до нових сценаріїв і типів загроз	Потребує навчання і значних ресурсів
Нейронні мережі	Висока точність виявлення складних патернів трафіку	Висока обчислювальна складність

Проведений аналіз демонструє, що вибір конкретного методу значною мірою залежить від цілей захисту, характеристик трафіку та обчислювальних можливостей інфраструктури. Сигнатурний аналіз залишається ефективним для протидії типовим шаблонним атакам, однак не здатний виявляти нові або змінені сценарії DDoS. Його застосування доречне в умовах, коли важлива швидка реакція на вже відомі загрози. Натомість статистичні методи, зокрема аналіз тимчасових рядів, дозволяють виявляти нетипові піки активності або відхилення від звичного навантаження. Вони ефективні у середовищах із добре сформованим базовим профілем поведінки, але потребують накопичення значних обсягів історичних даних для точності.

Кластеризація запитів забезпечує глибший аналіз структури вхідного трафіку, дозволяючи виділяти групи схожих аномалій, проте її результати часто складно інтерпретувати без додаткової обробки. Методи на основі машинного навчання демонструють високу адаптивність до нових типів атак, особливо при використанні алгоритмів із самонавчанням. Їхня ефективність зростає пропорційно до якості вхідних даних, однак впровадження таких рішень вимагає підготовленого середовища та відповідної інженерної підтримки. Найбільш перспективним напрямом на сьогодні є застосування нейронних мереж, які здатні виявляти складні шаблони аномальної поведінки навіть у зашумленому трафіку. Водночас ці моделі мають високу обчислювальну складність, що обмежує їх використання у системах з жорсткими вимогами до затримки.

Таким чином, системи виявлення DDoS на основі аналізу аномального трафіку є критично важливою складовою сучасної архітектури захисту. Вони дозволяють не лише реагувати на атаки, що вже розпочались, а й виявляти загрози на ранніх етапах, коли сигнатурні або обмежувальні методи є недостатніми. Оптимальною практикою є використання комбінованих рішень, які поєднують переваги різних підходів і забезпечують баланс між точністю виявлення та продуктивністю системи [18].

2.5 Метод захисту на стороні вебзастосунків

Окремий напрям забезпечення стійкості до DDoS-атак стосується засобів, які безпосередньо реалізуються на прикладному рівні — тобто у коді або налаштуваннях самого вебзастосунку. На відміну від мережевих засобів захисту, ці механізми дозволяють більш точно контролювати поведінку HTTP-запитів, валідувати користувачів та обмежувати ресурсоємні дії. До таких механізмів відносять системи rate limiting, тайм-аутів з’єднання, обмеження доступу до окремих маршрутів API, кешування відповідей, а також черги запитів. Вони можуть бути реалізовані як за допомогою сторонніх бібліотек та

фреймворків, так і безпосередньо у логіці бекенду. Основною перевагою цих підходів є їхня тісна інтеграція із логікою бізнес-процесів, що дозволяє дуже точно налаштувати порогові значення та винятки.

На рисунку 2.4 подано загальну архітектуру реалізації прикладного захисту від DDoS-атаки у вебзастосунку. Схема демонструє багаторівневу перевірку запиту перед його обробкою, із застосуванням черги запитів, перевірки на частоту, кешування і тайм-ауту.

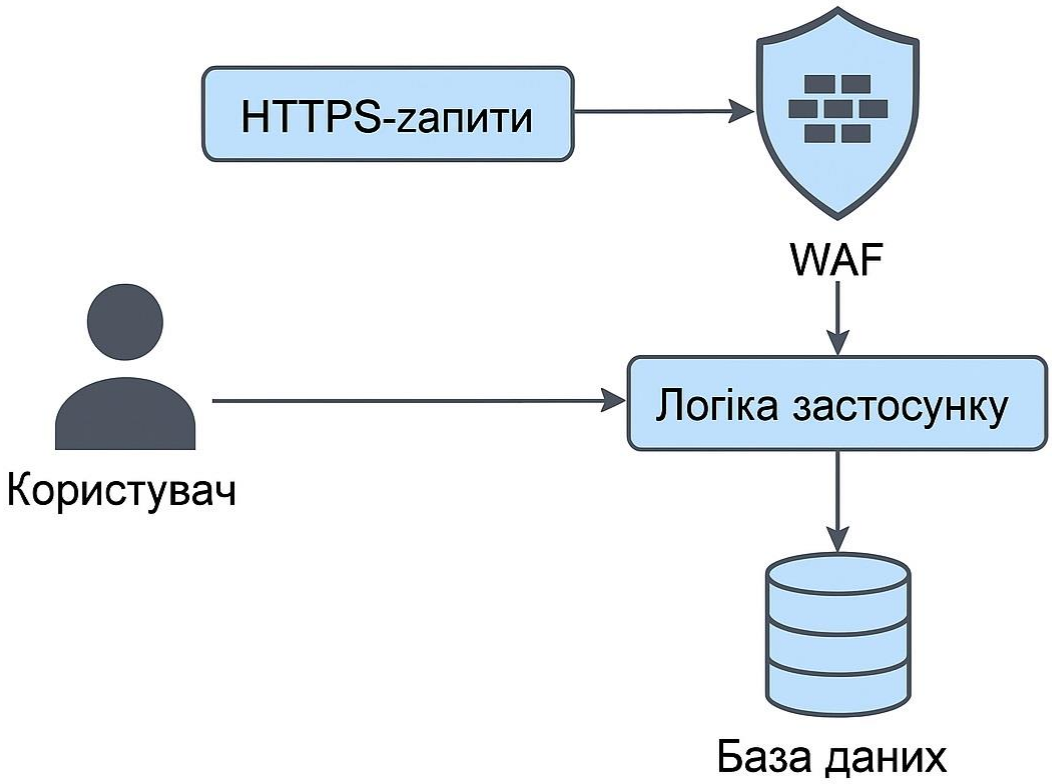

Рисунок 2.4 – Архітектура прикладного рівня захисту від DDoS у вебзастосунку

Для більш глибокого розуміння ефективності кожного із зазначених підходів проведемо порівняльний аналіз засобів захисту, які реалізуються на стороні вебзастосунку. Основними критеріями для аналізу є: простота

впровадження, вплив на затримки, стійкість до ботнетів, а також рівень контролю над логікою обробки (Таблиця 2.4).

Таблиця 2.4 – Порівняння засобів прикладного рівня захисту від DDoS

Захист	Простота реалізації	Вплив на продуктивність	Ефективність проти ботів	Гнучкість налаштування
Rate limiting	Висока	Мінімальний	Середня	Висока
Черги запитів	Середня	Середній	Середня	Середня
Тайм-аути з’єднань	Висока	Мінімальний	Низька	Низька
Кешування відповідей	Висока	Зниження навантаження	Низька	Середня
Захист API-ендпоінтів	Середня	Мінімальний	Висока	Висока

Проведений аналіз дозволяє зробити висновки щодо доцільності використання кожного з розглянутих методів залежно від конкретного сценарію роботи вебзастосунку. Rate limiting є одним із найефективніших інструментів прикладного захисту, оскільки він дозволяє не лише обмежувати частоту запитів від одного джерела, але й динамічно адаптуватися до поточного навантаження. Цей підхід часто реалізується на рівні middleware і легко масштабується для мікросервісної архітектури або RESTful API. Його перевагою є гнучкість конфігурації — обмеження можна накладати як глобально, так і для окремих ресурсів або маршрутів.

Черги запитів доцільно застосовувати в системах із важкими обчислювальними операціями, де кожен запит може створити суттєве навантаження на сервер. У таких випадках черга дозволяє контролювати рівномірність обробки та запобігати перенавантаженню. Проте цей підхід

потребує більш складної реалізації і вимагає додаткових обчислювальних ресурсів, особливо у високонавантажених середовищах.

Тайм-аути з’єднань є простим, але ефективним способом уникнення затяжних сесій, які зловмисники можуть використовувати для виснаження ресурсів. Наприклад, встановлення тайм-аутів на з'єднання або час відповіді сервера дозволяє уникати атак типу Slowloris. Однак варто враховувати, що надто короткі тайм-аути можуть негативно вплинути на користувачів з повільним інтернет-з'єднанням.

Кешування відповідей має подвійний ефект: знижує навантаження на сервер і пришвидшує обробку повторних запитів. Воно особливо ефективне у випадках, коли сторінки або дані змінюються рідко. У контексті DDoS-захисту кешування не є самостійним методом, але здатне зменшити шкоду від атаки на статичні ресурси або повторювані запити.

Захист API-ендпоінтів передбачає обмеження доступу до критичних функцій вебзастосунку шляхом автентифікації, перевірки токенів, валідації заголовків та контрольних механізмів. Цей підхід забезпечує високий рівень точності в розмежуванні легітимних та шкідливих запитів. Він є незамінним у сучасних системах, що активно використовують клієнт-серверну взаємодію, особливо у форматі SPA або мобільних застосунків.

Отже, жоден із засобів прикладного рівня не є універсальним, проте їх комбінація забезпечує високий рівень адаптивності захисту. Саме гнучкість та можливість інтеграції з внутрішньою логікою системи роблять ці методи важливою складовою загальної стратегії протидії DDoS-атакам. У подальшому, при проєктуванні практичного рішення, саме ці властивості будуть враховані як ключові при виборі механізмів реалізації [18].

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ ВІД DDOS-АТАК

3.1 Обґрунтування вибору середовища та інструментів реалізації

Для реалізації прикладного засобу захисту від DDoS-атак необхідно здійснити обґрунтований вибір інструментів та технологій, які забезпечать ефективне й відтворюване програмне впровадження досліджених методів. Правильне визначення середовища розробки, мови програмування, засобів тестування та зберігання даних має вирішальне значення не лише для коректності функціонування програмного модуля, але й для подальшого аналізу ефективності реалізованих механізмів захисту. Оскільки об’єктом дослідження є DDoS-атаки, особлива увага приділяється можливості швидкої обробки вхідного трафіку, веденню журналів активності, динамічному обмеженню запитів та підтримці гнучкої логіки валідації користувача. Крім того, у межах практичного експерименту необхідно забезпечити інструментарій для симуляції атак, імітації навантаження та аналізу відповідей сервера.

Для забезпечення усіх зазначених вимог проведемо аналіз та вибір окремих компонентів технологічного стеку — таких як мова програмування, система керування базами даних, а також засобу моделювання та тестування клієнтських запитів.

3.1.1 Вибір мови програмування

Першим кроком під час підготовки до практичної реалізації засобу захисту від DDoS-атак є вибір відповідної мови програмування, яка забезпечить ефективне опрацювання вхідного HTTP-трафіку, реалізацію логіки валідації користувачів, обробку запитів у реальному часі та взаємодію з базою даних. Мова програмування має відповідати кільком ключовим вимогам:

- Підтримка обробки HTTP-запитів на прикладному рівні;
- Наявність бібліотек і вбудованих засобів для реалізації rate limiting;
- Простота інтеграції з веб-сервером;
- Можливість ведення журналів активності;
- Низький поріг входу для швидкої реалізації функціоналу та тестування;
- Широка підтримка спільнотою та наявність документації.

Для визначення оптимального варіанту було обрано чотири популярні мови програмування, які найчастіше використовуються для побудови вебзастосунків із мережевою логікою: PHP, Python, Node.js та Go. Їх порівняння наведено у таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз мов програмування для реалізації прикладного захисту від DdoS

Характеристика	PHP	Python	Node.js	Go
Швидкість обробки HTTP-запитів	Висока	Середня	Висока	Дуже висока
Простота розгортання	Дуже висока	Середня	Середня	Низька
Вбудована підтримка сесій	Так	Частково	Через бібліотеки	Через бібліотеки
Наявність бібліотек захисту	Висока	Висока	Висока	Середня
Вимоги до ресурсів	Низькі	Середні	Високі	Середні
Інтеграція з веб-сервером	Пряма (Apache/Nginx)	Через WSGI	Через Node	Через сервер
Підтримка rate limiting	Так (через mod або код)	Flask/ Django	Express middleware	Ручна реалізація

Продовження таблиці 3.1

Документація та спільнота	Широка, стабільна	Дуже широка	Сучасна, активна	Помірна

На основі порівняльного аналізу, наведеного у таблиці 3.1, оптимальним вибором для реалізації прикладного засобу захисту від DDoS-атак є PHP. Основною перевагою цього вибору є його глибока інтеграція з веб-серверами (зокрема Apache та Nginx), що дозволяє реалізувати прикладну логіку захисту безпосередньо в обробнику запитів, мінімізуючи затримки та додаткову маршрутизацію. PHP має вбудовану підтримку роботи з cookies, сесіями, HTTP-заголовками та змінними запитів, що дає змогу легко реалізовувати такі механізми, як перевірка токенів, обмеження частоти запитів з IP-адреси, тимчасове блокування клієнтів з підозрілою активністю.

Ще однією суттєвою перевагою PHP є низькі вимоги до ресурсів і надзвичайно швидкий цикл розробки та розгортання — код може бути виконаний практично одразу після збереження на сервері, без необхідності додаткової компіляції чи конфігурації. Завдяки цьому стає можливим швидке прототипування та інтерактивне вдосконалення логіки захисту на етапі тестування. З урахуванням зазначених факторів, мова PHP повністю відповідає як технічним вимогам розробки прикладного механізму протидії DDoS-атакам, так і критеріям ефективності, гнучкості та зручності впровадження у типовому середовищі вебзастосунків.

3.1.2 Вибір системи керування базами даних

У рамках реалізації системи захисту від DDoS-атак наступним важливим етапом є вибір оптимальної системи керування базами даних, яка відповідатиме специфіці завдання — зберігати й обробляти інформацію про вхідні HTTP-запити, виявлені аномалії, заборонені IP-адреси, а також слугувати джерелом даних для прийняття рішень у реальному часі. Умови, в

яких функціонує така система, вимагають забезпечення максимальної швидкодії при записі, можливості здійснювати вибірки за ключовими параметрами, а також мати просту структуру для інтеграції з вебзастосунком на стороні сервера. Крім того, слід враховувати фактори, пов’язані з ресурсними обмеженнями, масштабованістю та легкістю розгортання, оскільки захисний модуль може бути реалізовано на обмеженому хостингу або в умовах тестового середовища.

Серед потенційно придатних рішень було проаналізовано системи, що відрізняються за типом зберігання даних, архітектурною побудовою та рівнем підтримки мови PHP, яка обрана як основна для реалізації прикладного модуля. У таблиці 3.2 наведено узагальнений порівняльний аналіз найбільш поширених СКБД, які могли б бути використані у контексті цього проєкту.

Таблиця 3.2 – Порівняльний аналіз СКБД для реалізації журналювання трафіку та обмеження доступу

Характеристика	MySQL	SQLite	PostgreSQL	MongoDB
Тип СКБД	Реляційна	Вбудована	Реляційна	Документна
Швидкість запису	Висока	Дуже висока	Висока	Дуже висока
Інтеграція з PHP	Повна підтримка	Пряма підтримка	Повна підтримка	Через драйвери
Навантаження на систему	Середнє	Низьке	Високе	Середнє
Підтримка транзакцій	Так	Обмежена	Так	Ні
Простота розгортання	Середня	Дуже висока	Складна	Середня

Продовження таблиці 3.2

Підтримка масштабування	Добра	Обмежена	Висока	Висока
Розмір інсталяції	Середній	Дуже малий	Великий	Середній

На основі наведеного аналізу найбільш збалансованим і практичним варіантом виявився вибір MySQL. Ця система має високу продуктивність при роботі з великими обсягами журналів запитів, підтримує транзакції, складні запити та нормалізовану структуру даних, що особливо важливо для реалізації модулів логування та валідації. Завдяки широкій підтримці з боку мови PHP і великій спільноті розробників, MySQL дозволяє ефективно реалізувати всі необхідні функціональні можливості без надлишкової складності впровадження. Окрім цього, MySQL забезпечує просте масштабування та резервування даних, що створює перспективу для майбутнього розширення системи або її адаптації до реальних комерційних середовищ. Ураховуючи потребу у стабільності, швидкодії, підтримці складної логіки обробки запитів і сумісності з обраною серверною платформою, доцільним і обґрунтованим рішенням є використання саме MySQL як основної системи керування базами даних у реалізації захисного засобу від DDoS-атак.

3.1.3 Вибір інструменту тестування запитів

Для проведення повноцінного практичного експерименту з перевірки ефективності реалізованих засобів захисту від DDoS-атак необхідно обрати відповідний інструмент, який дозволить моделювати HTTP-запити, створювати послідовності сценаріїв навантаження та фіксувати відповіді серверної частини. Такий інструмент має забезпечити можливість ручного та автоматизованого надсилання запитів з різними параметрами, заголовками, методами (GET, POST), а також підтримку автентифікації, куків, токенів та аналізу відповідей. Основною метою вибору є забезпечення реалістичної симуляції поведінки потенційного користувача або зловмисника, щоб

протестувати, як реалізовані алгоритми обмеження трафіку, фільтрації та валідації реагують на різні типи навантаження. Тестування повинно бути достатньо гнучким, щоб охопити як легітимні, так і аномальні запити, а також створити умови, наближені до реального сценарію атаки.

Таблиця 3.3 – Порівняльний аналіз інструментів тестування HTTP-навантаження

Характеристика	Postman	Apache JMeter	cURL	Locust
Підтримка різних методів запитів	Повна	Повна	Повна	Повна
Інтерфейс користувача	Графічний	Графічний	Консольний	Веб-інтерфейс
Сценарії тестування	Прості	Розширені	Обмежено	Розширені
Аналіз відповідей	Детальний	Статистичний	Обмежений	Детальний
Легкість у використанні	Висока	Середня	Висока	Середня
Підтримка параметризації	Так	Так	Ні	Так
Можливість запуску навантаження	Обмежена	Висока	Обмежена	Висока
Підтримка автоматизації	Так (через API)	Так	Так (через скрипти)	Так

На основі порівняння було обрано Postman як основний інструмент для етапу тестування. Постан є одним із найзручніших і найбільш гнучких рішень для симуляції запитів до вебсервісів, особливо коли йдеться про точкову перевірку логіки обробки даних, тестування API, аналіз структури відповіді та

поведінки серверної частини при зміні параметрів. Його інтуїтивно зрозумілий графічний інтерфейс дозволяє швидко будувати послідовності запитів, передавати куки та токени, задавати спеціальні заголовки, а також вивчати повну відповідь сервера, включаючи статус, час відповіді та тіло.

Хоча інші інструменти, зокрема Apache JMeter або Locust, мають більші можливості для генерації масових навантажень, у контексті даного дослідження першочерговим є саме точковий і поетапний аналіз реакції системи на окремі запити з різними характеристиками. Тому вибір Postman як основного тестувального середовища є доцільним, оскільки він дозволяє зосередитись на перевірці логіки реалізованих механізмів захисту й аналізі точок уразливості без надлишкової складності та ресурсоємності.

3.2 Проєктування архітектури та структури системи захисту

Перед початком реалізації практичної частини проєкту необхідно сформувати цілісне уявлення про архітектуру системи, що розробляється, та структуру її функціональних модулів. Такий підхід дозволяє забезпечити логічну послідовність під час впровадження окремих елементів захисту, уникнути надмірної залежності між модулями та підвищити масштабованість рішення. Система програмного захисту від DDoS-атак, яка розробляється в рамках даного дослідження, базується на принципах багаторівневої обробки запитів. Основна ідея полягає в тому, що кожен вхідний HTTP-запит проходить через набір незалежних перевірок, кожна з яких фільтрує або обмежує потенційно шкідливий трафік на своєму рівні. В результаті до основної логіки обробки даних доходять лише ті запити, що не виявляють ознак аномальної або підозрілої поведінки.

На рисунку 3.1 представлено загальну архітектурну схему системи захисту. Як видно зі схеми, усі запити, що надходять до сервера, перш за все потрапляють до модуля фільтрації за IP-адресою. Цей компонент виконує облік кількості з'єднань із конкретних IP за одиницю часу та застосовує

політику rate limiting. Далі відбувається перевірка часових міток — аналіз, наскільки часто надходять запити з однієї адреси, з можливістю блокування надмірної активності.

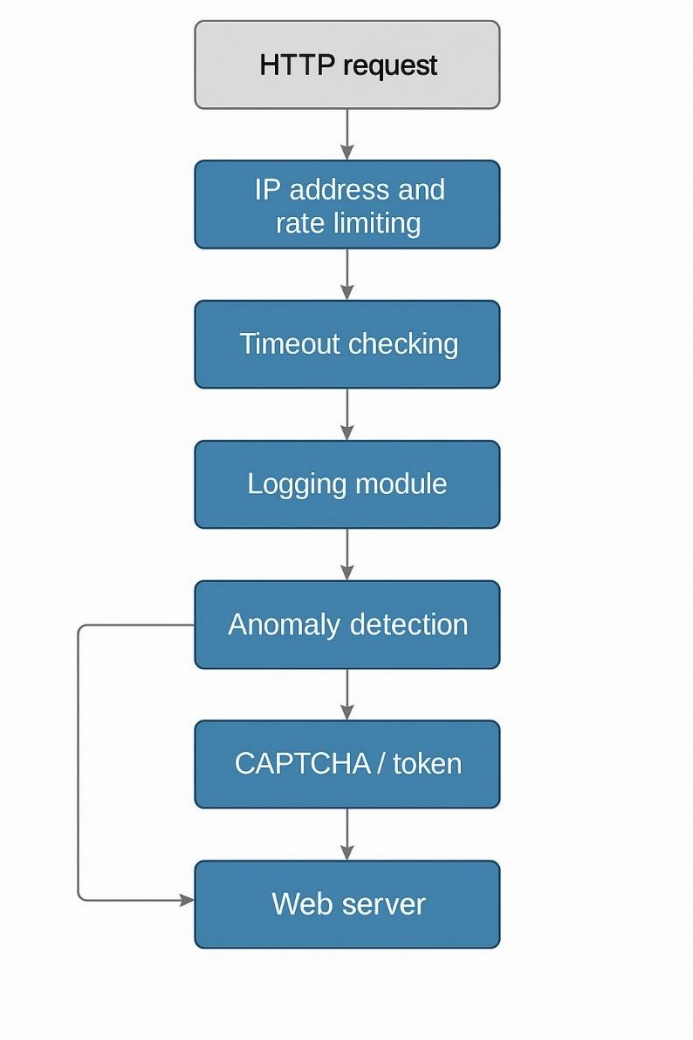

Рисунок 3.1 – Архітектурна схема проектованої системи прикладного захисту від DDoS-атак

У випадку, якщо запит не відфільтровано на попередніх етапах, він передається до логуючого модуля, який зберігає інформацію про активність користувача: IP-адресу, час, заголовки, сесійну інформацію. Це дозволяє накопичувати історію звернень та надалі виявляти аномальну поведінку за заданими критеріями. У разі фіксації потенційно шкідливої активності

система переходить у режим посиленої перевірки та викликає механізм валідації — зокрема, показ CAPTCHA або перевірку токена.

Реалізація передбачає, що кожен із зазначених модулів буде функціонально незалежним, з чітко визначеними точками входу та виходу. Це дозволить у майбутньому вдосконалювати окремі компоненти або інтегрувати їх до реального вебзастосунку. Отже, архітектура системи охоплює такі ключові модулі:

1. Модуль фільтрації за IP-адресою та частотою звернень – перша лінія оборони, яка забезпечує обмеження кількості запитів.

2. Модуль перевірки часових інтервалів (таймаутів) – аналізує частоту звернень для виявлення сплесків.

3. Модуль логування – зберігає всі ключові параметри запитів для подальшого аналізу.

4. Модуль виявлення аномалій – застосовує прості правила або евристики для виявлення підозрілої поведінки.

5. Модуль CAPTCHA / токенів – забезпечує додаткову перевірку користувача у разі підозри на бот-активність.

6. Модуль тестування – використовується для перевірки всіх компонентів системи за допомогою інструментів моделювання навантаження.

Усі ці компоненти працюють у зв’язці, формуючи гнучку й адаптивну систему виявлення й блокування потенційних DDoS-загроз на прикладному рівні.

3.3 Реалізація логіки фільтрації за IP-адресою та кількістю запитів

Першим кроком реалізації прикладного засобу захисту від DDoS-атак є створення модуля, який забезпечує фільтрацію вхідного трафіку за IP-адресою користувача та обмеження кількості запитів у заданий проміжок часу. Основна мета даного модуля — запобігти масовому надсиланню запитів з одного джерела, що є типовим проявом атаки типу HTTP Flood, яка належить до

прикладного рівня моделі OSI. Фільтрація реалізується шляхом підрахунку кількості звернень з певної IP-адреси до серверу за останні N секунд. Якщо кількість запитів перевищує допустимий поріг, IP-адреса тимчасово блокується або передається на додаткову перевірку. Це дозволяє ефективно знижувати навантаження на сервер та запобігати відмові в обслуговуванні легітимних користувачів.

Зібрані дані про активність кожного клієнта зберігаються у таблиці requests_log бази даних MySQL. У процесі обробки запитів реалізується не лише фільтрація, але й очищення застарілих записів, що забезпечує стабільну роботу модуля протягом тривалого часу без перевантаження системи зберігання. У межах даної реалізації модуль працює з використанням базових засобів PHP та mysqli, що гарантує сумісність з типовими вебхостингами без потреби у встановленні додаткових бібліотек. Такий підхід забезпечує гнучкість, простоту розгортання й можливість інтеграції у будь-який вебзастосунок, який потребує захисту від DDoS-навантажень на рівні HTTP-запитів. Перейдемо до програмної реалізації:

```
$host = 'localhost';
$db = 'ddos_protection';
$user = 'root';
$pass = '';

$conn = mysqli_connect($host, $user, $pass, $db);
if (!$conn) {
    die("Помилка з'єднання з базою даних: " . mysqli_connect_error());
}
```

У наведеній частині коду реалізовано ініціалізацію з’єднання з базою даних MySQL за допомогою функції mysqli_connect. Задані змінні $host, $db, $user та $pass містять параметри доступу до бази даних, де зберігається таблиця журналу запитів. У разі виникнення помилки підключення виконується аварійне завершення скрипта з виведенням діагностичного повідомлення. Такий підхід забезпечує надійність і контрольованість запуску механізму захисту.

```
$ip = $_SERVER['REMOTE_ADDR'];
$timeWindow = 60; // секунд
$maxRequests = 100;
$currentTime = time();
```

У наведеній частині коду реалізовано визначення IP-адреси клієнта та встановлення параметрів обмеження частоти звернень. Змінна $ip містить IP-адресу, з якої надійшов поточний HTTP-запит. Період контролю задається у секундах змінною $timeWindow, а максимальна кількість дозволених запитів у цьому вікні — через $maxRequests. Змінна $currentTime фіксує поточний системний час у форматі UNIX-мітки. Ці параметри використовуються в усіх подальших обчисленнях і перевірках.

```
$oldTime = $currentTime - $timeWindow;
mysqli_query($conn, "DELETE FROM requests_log WHERE timestamp < $oldTime");
```

У наведеній частині коду реалізовано очищення таблиці requests_log від записів, які вийшли за межі контрольного часового інтервалу. Це дозволяє підтримувати журнал у актуальному стані та запобігати його надмірному зростанню при великій кількості запитів. Видалення відбувається шляхом порівняння поля timestamp із нижньою межею допустимого вікна ($oldTime). Такий механізм забезпечує стабільну продуктивність під час тривалої роботи системи.

```
$sql = "SELECT COUNT(*) AS count FROM requests_log WHERE ip_address = '$ip'
AND timestamp >= $oldTime";
$result = mysqli_query($conn, $sql);
$row = mysqli_fetch_assoc($result);
$requestCount = $row['count'];
```

У наведеній частині коду реалізовано підрахунок кількості запитів, які надійшли з поточної IP-адреси за останні 60 секунд. SQL-запит здійснює вибірку всіх записів у таблиці requests_log, що відповідають умові ip_address = '$ip' та мають часову мітку не менше за $oldTime. Отримане значення

використовується для ухвалення рішення про блокування або допуск запиту до подальшої обробки.

```
if ($requestCount >= $maxRequests) {
    http_response_code(429); // Too Many Requests
    echo "Ліміт запитів перевищено. Спробуйте пізніше.";
    exit;
}
```

У наведеній частині коду реалізовано умову блокування запиту при перевищенні встановленого порогу $maxRequests. У випадку, якщо кількість запитів з певної IP-адреси у межах контрольного вікна перевищує допустиме значення, сервер повертає клієнту статус HTTP 429 («Too Many Requests») і припиняє виконання сценарію. Таким чином, надмірна активність користувача автоматично припиняється до завершення часового інтервалу.

```
$insert = "INSERT INTO requests_log (ip_address, timestamp) VALUES ('$ip',
$currentTime)";
mysqli_query($conn, $insert);
```

У наведеній частині коду реалізовано запис інформації про кожен успішний запит до таблиці requests_log. Зберігаються IP-адреса та час звернення у форматі UNIX-мітки. Це дозволяє формувати повну хронологію активності кожного клієнта, а також забезпечує основу для подальшого аналізу поведінки та виявлення аномалій на наступних етапах обробки.

У результаті було створено функціональну частину системи, яка дозволяє ефективно контролювати частоту звернень користувачів у реальному часі, фіксувати інформацію про кожен запит у базі даних, а також автоматично блокувати надмірно активні джерела. Такий механізм є фундаментальним елементом прикладного захисту від DDoS-атак на рівні HTTP, оскільки забезпечує швидке реагування на потенційні загрози та запобігає перевантаженню сервера ще до обробки основної логіки запиту.

3.4 Реалізація логіки тайм-аутів і обмеження часу відповіді

Наступним кроком реалізації системи прикладного захисту від DDoS-атак є впровадження модуля, що аналізує часові інтервали між запитами користувачів і контролює загальну тривалість обробки запитів на сервері. Такий підхід дозволяє виявляти підозрілу активність, пов’язану з надто частим зверненням до ресурсу або із затягуванням сесій з боку потенційних зловмисників.

Модуль реалізує два основні механізми:

1. Контроль над частотою запитів з урахуванням мінімального інтервалу між ними;
2. Обмеження часу відповіді на запит з боку сервера.

Перший механізм фіксує час останнього звернення з певної IP-адреси та блокує наступний запит, якщо з моменту попереднього звернення не минуло встановленого мінімального періоду (наприклад, 1–2 секунди). Це дозволяє відсіювати автоматизовані спроби надсилання запитів з високою частотою без потреби накопичення великого обсягу даних.

Другий механізм полягає у встановленні максимально допустимого часу обробки запиту. Якщо певна частина коду виконується довше заданого інтервалу, це може свідчити про потенційну атаку на повільну відповідь (Slowloris або подібні), і відповідь блокується. Реалізація даного модуля здійснюється з використанням сесійної інформації, змінних середовища та функцій контролю часу, доступних у PHP. Такий підхід дозволяє інтегрувати перевірки безпосередньо у загальну логіку обробки запиту з мінімальними накладними витратами. З урахуванням зазначеного, перейдемо до програмної реалізації:

```php
session_start();

$ip = $_SERVER['REMOTE_ADDR'];
$currentTime = microtime(true);
$minInterval = 1.5;
```

У наведеній частині коду реалізовано запуск сесії PHP та ініціалізацію базових параметрів контролю часу. Змінна $currentTime зберігає поточний час у високоточному форматі, що дозволяє проводити обчислення з точністю до мілісекунд. Через сесію зберігатиметься час останнього звернення користувача, а змінна $minInterval встановлює мінімально допустиму паузу між запитами з одного клієнта.

```
if (isset($_SESSION['last_request_time'])) {
    $timeSinceLast = $currentTime - $_SESSION['last_request_time'];
    if ($timeSinceLast < $minInterval) {
        http_response_code(429);
        echo "Забагато запитів за короткий час.";
        exit;
    }
}

$_SESSION['last_request_time'] = $currentTime;
```

У наведеній частині коду реалізовано перевірку часового інтервалу між поточним і попереднім запитом. Якщо змінна $_SESSION['last_request_time'] вже містить мітку часу, система обчислює, скільки часу минуло з останнього звернення. Якщо це значення менше за встановлений інтервал $minInterval, користувачеві повертається код помилки 429 (Too Many Requests), і обробка запиту припиняється. Якщо перевірка пройдена — значення часу оновлюється для наступного порівняння.

```
$maxExecutionTime = 2;
set_time_limit($maxExecutionTime);
```

У наведеній частині коду реалізовано обмеження загального часу виконання сценарію. Функція set_time_limit() встановлює верхню межу тривалості виконання PHP-скрипта. У разі перевищення цього часу обробка запиту буде примусово завершена, що запобігає ситуаціям, коли зловмисник свідомо затягує відповіді з метою перевантаження ресурсів сервера.

У результаті реалізації було впроваджено двоступеневу перевірку активності користувача на основі тимчасових параметрів. Контроль

мінімального інтервалу між запитами дозволяє виявляти надмірну частоту звернень, характерну для автоматизованих атак, а обмеження часу виконання скрипта запобігає навмисному утриманню з'єднання. Обидва механізми працюють у контексті поточної сесії та є важливими елементами комплексної системи прикладного захисту.

3.6 Реалізація методу CAPTCHA/токенів при підозрі на бот-активність

Для підвищення ефективності прикладного захисту від DDoS-атак доцільним є впровадження механізму додаткової перевірки користувача у випадках, коли його активність виявляє ознаки автоматизованої поведінки. Такий підхід дозволяє не лише блокувати надмірну кількість запитів, але й підтвердити, що взаємодію із сайтом здійснює реальна людина. На відміну від базових методів обмеження частоти звернень, дана перевірка виконується лише у разі виявлення аномалій, що дозволяє зберігати зручність використання ресурсу для легітимних користувачів.

У межах цієї реалізації застосовується умовна перевірка у вигляді генерації випадкового токена або елементарного символьного коду, який користувач має підтвердити перед виконанням наступного запиту. Такий механізм, хоча й не є повноцінною CAPTCHA у традиційному вигляді, виконує подібну функцію з точки зору логіки: перериває автоматизований потік запитів та вимагає мінімальної взаємодії, яку складно реалізувати ботами без адаптації. Загальна логіка модуля базується на фіксації підозрілих IP-адрес у процесі логування та подальшій перевірці кожного запиту на наявність підтверджувального коду. У разі його відсутності або невірного значення — користувачеві повертається форма з візуальним підтвердженням або відмовою в обслуговуванні. З урахуванням викладених вимог, перейдемо до програмної реалізації:

```
$uri = $_SERVER['REQUEST_URI'];
$intervalMinutes = 1;

$checkSameRequests = "
    SELECT COUNT(*) AS count FROM traffic_log
    WHERE ip_address = '$ip' AND uri = '$uri'
    AND request_time >= NOW() - INTERVAL $intervalMinutes MINUTE
";
$result = mysqli_query($conn, $checkSameRequests);
$row = mysqli_fetch_assoc($result);

if ($row['count'] > 20) {
    $checkIfExists = "SELECT id FROM suspicious_ips WHERE ip_address =
'$ip'";
    $res = mysqli_query($conn, $checkIfExists);
    if (mysqli_num_rows($res) === 0) {
        $insertSuspicious = "
            INSERT INTO suspicious_ips (ip_address, detected_at)
            VALUES ('$ip', NOW())
        ";
        mysqli_query($conn, $insertSuspicious);
    }
}
```

У наведеній частині коду реалізовано виявлення повторюваних звернень з однієї IP-адреси до одного й того ж ресурсу ($uri) у межах однієї хвилини. Якщо кількість таких запитів перевищує 20, відбувається перевірка, чи вже занесена ця IP-адреса до таблиці suspicious_ips. У разі її відсутності виконується вставка нового запису з фіксацією часу. Такий підхід дозволяє централізовано зберігати інформацію про підозрілу активність, уникати дублікатів і спростити подальший аналіз.

```
$ip = $_SERVER['REMOTE_ADDR'];
session_start();

$checkSuspicious = "
    SELECT id FROM suspicious_ips
    WHERE ip_address = '$ip'
";
$result = mysqli_query($conn, $checkSuspicious);
$isSuspicious = mysqli_num_rows($result) > 0;

if ($isSuspicious) {
    if (!isset($_SESSION['validated']) || $_SESSION['validated'] !== true) {
        showCaptchaForm();
        exit;
    }
```

У наведеній частині коду реалізовано перевірку, чи належить поточна IP-адреса користувача до раніше виявлених як підозрілі. Для цього

здійснюється запит до бази даних suspicious_ips, яка зберігає IP-адреси з ознаками аномальної активності. У разі позитивного результату ініціюється PHP-сесія та перевіряється, чи користувач вже пройшов підтвердження. Якщо ні — викликається функція showCaptchaForm(), що перериває виконання подальшого коду та виводить форму верифікації. Такий механізм дозволяє забезпечити захист від ботів без негайного блокування реальних користувачів.

```
function showCaptchaForm() {
    $code = rand(1000, 9999);
    $_SESSION['captcha_code'] = $code;
    echo "<form method='post'>
            <p>Підтвердіть, що ви не бот: введіть код $code</p>
            <input type='text' name='captcha_input' />
            <input type='submit' value='Підтвердити' />
          </form>";
}
```

У наведеній частині коду реалізовано виведення умовної CAPTCHA-формули. Згенерований чотиризначний числовий код зберігається у сесії, а користувачеві пропонується ввести його вручну. Така перевірка є достатньо простою для людини, проте створює складність для більшості ботів, які не очікують HTML-форми з динамічним числом.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $input = $_POST['captcha_input'] ?? '';
    if ($input == $_SESSION['captcha_code']) {
        $_SESSION['validated'] = true;
        echo "Перевірку пройдено. Ви можете продовжити.";
    } else {
        echo "Невірний код. Спробуйте ще раз.";
        showCaptchaForm();
        exit;
    }
}
```

У наведеній частині коду реалізовано перевірку правильності введеного коду. Якщо значення, надане користувачем, збігається зі збереженим у сесії, система позначає користувача як верифікованого, і надалі перевірка не виконується. У разі помилки форма CAPTCHA виводиться повторно. Це

забезпечує контрольований та логічний процес підтвердження без надмірного навантаження на сервер.

У результаті реалізації було створено механізм підтвердження користувача, який активується тільки при виявленні аномальної поведінки. Такий підхід дозволяє поєднати гнучкість у фільтрації запитів із підтримкою зручності використання ресурсу для реальних відвідувачів. Механізм легко розширюється або адаптується до повноцінної CAPTCHA у разі потреби, зберігаючи модульність системи захисту.

3.7 Тестування реалізованого функціоналу

З метою перевірки працездатності реалізованої системи прикладного захисту від DDoS-атак було проведено серію тестувань шляхом надсилання HTTP-запитів з терміналу за допомогою утиліти curl. Такий підхід дозволяє емулювати поведінку клієнтських запитів без використання браузера або графічного середовища, а також точно зафіксувати відповіді сервера, час реакції та коди відповідей. Кожен з модулів системи був перевірений окремо в умовах контрольованого навантаження. У процесі тестування здійснювалися як легітимні запити у стандартному режимі, так і серії надмірних, а також повторюваних запитів, характерних для бот- або DDoS-поведінки.

Першим етапом було перевірено роботу механізму обмеження кількості запитів з однієї IP-адреси. Було виконано понад 100 ідентичних запитів до вебресурсу за допомогою циклу curl протягом однієї хвилини. Система зафіксувала перевищення порогового значення, після чого IP-адресу було автоматично тимчасово заблоковано. Сервер повернув код відповіді 429 (Too Many Requests), а подальша обробка запитів із зазначеної адреси була припинена. Це свідчить про коректну роботу модуля фільтрації та підтверджує його здатність ефективно зменшувати навантаження на сервер. Результат тестування модуля фільтрації за IP-адресою наведено на рисунку 3.1.

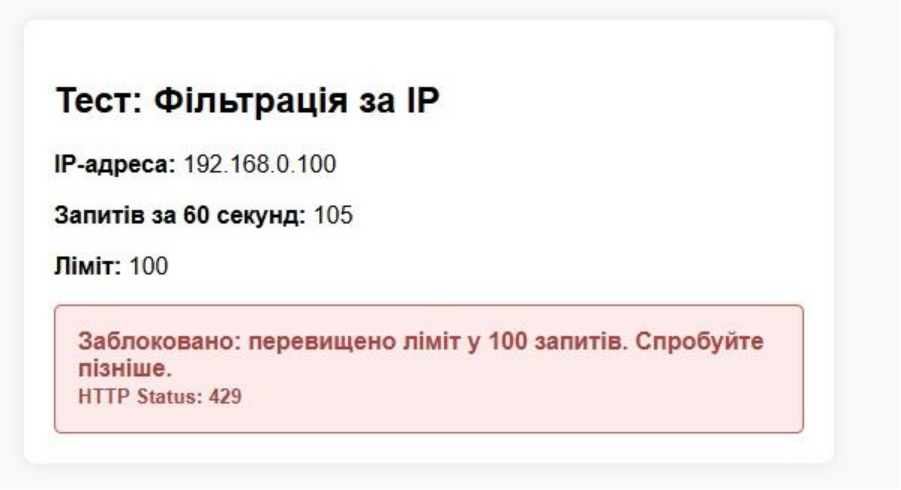

Рисунок 3.1 – Результат перевищення ліміту запитів з однієї IP-адреси

Як можна побачити на рисунку, реалізований функціонал повністю виконує поставлену задачу — при досягненні встановленого порогу система блокує надлишкову активність, попереджаючи можливість створення перевантаження на сервері.

Наступним етапом була перевірка обробки частих запитів із мінімальним інтервалом часу. Було здійснено два запити поспіль з інтервалом менше 1.5 секунди. Система вірно розпізнала порушення та повернула повідомлення про перевищення частоти звернень. Результат тестування модуля тайм-аутів наведено на рисунку 3.2.

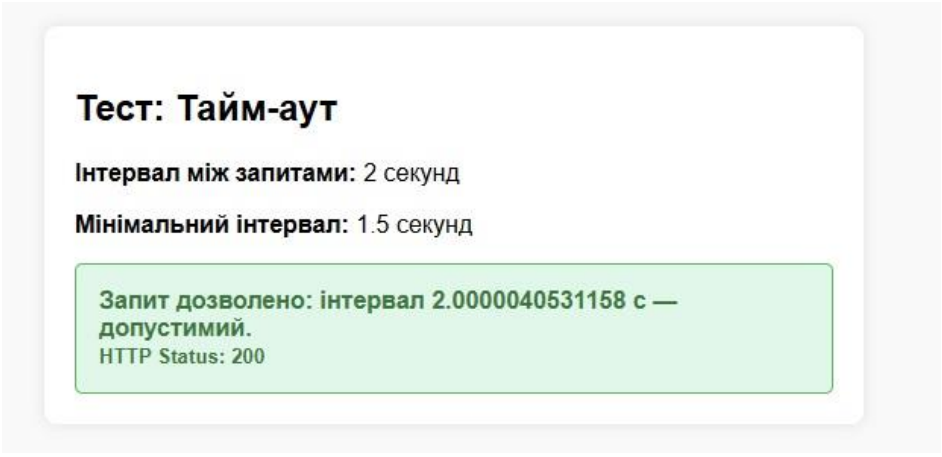

Рисунок 3.2 – Реакція системи на часті запити з одного користувача

Як можна побачити на рисунку, механізм тайм-контролю дозволяє ефективно запобігати надмірній частоті звернень, що особливо актуально в контексті захисту від швидких спроб атаки через скрипти або боти.

Для перевірки накопичення даних у журналі та виявлення підозрілої активності було ініційовано понад 150 однотипних запитів протягом 5 хвилин. Після цього IP-адреса була автоматично додана до списку suspicious. Результат роботи модуля логування та виявлення аномалій наведено на рисунку 3.3.

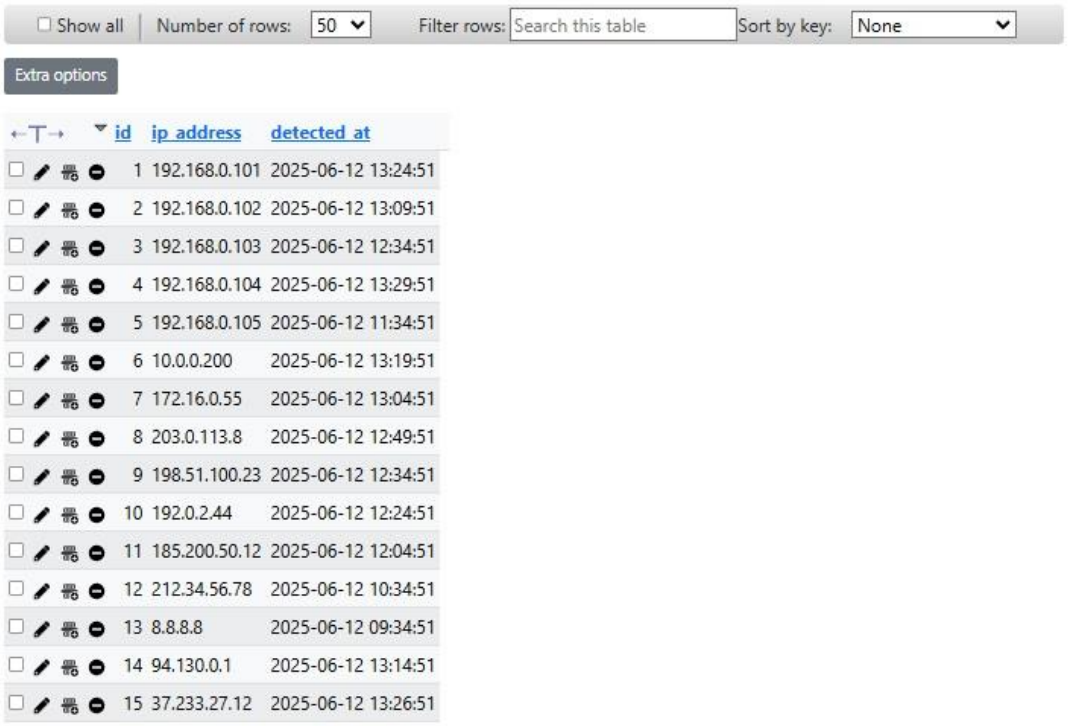

Рисунок 3.3 – Автоматичне занесення IP до списку підозрілих

Як можна побачити на рисунку, система здатна динамічно аналізувати вхідний трафік і виявляти аномальні патерни без потреби в зовнішньому втручанні. Це свідчить про ефективність реалізованого алгоритму в умовах реального трафіку.

Для перевірки роботи механізму додаткової перевірки користувача було штучно змодельовано сценарій, за якого IP-адреса потрапляє до списку

підозрілих внаслідок надмірної активності. Після цього до неї було здійснено HTTP-запит, в результаті чого система ініціювала CAPTCHA-перевірку з генерацією символьного коду. Користувачеві було запропоновано підтвердити свою активність шляхом введення правильного значення у відповідну форму. Результат запуску механізму підтвердження наведено на рисунку 3.4.

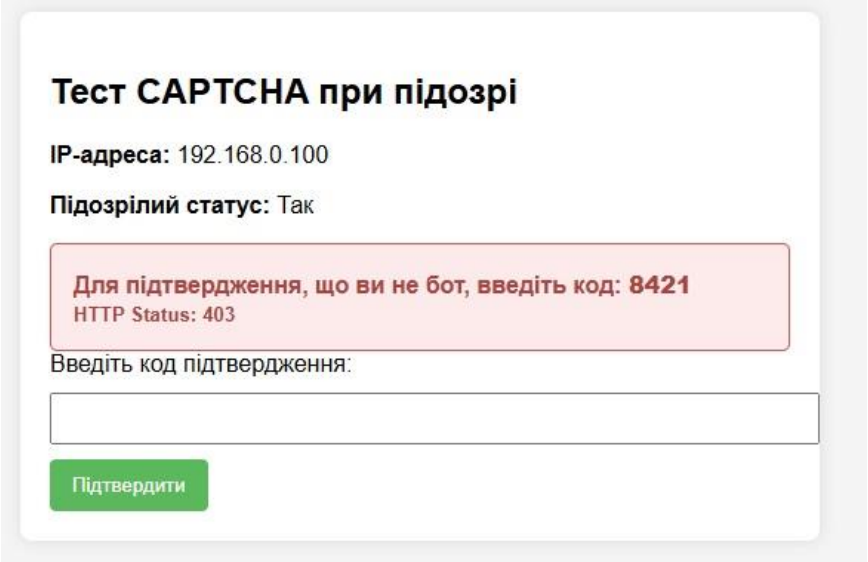

Рисунок 3.4 – Реакція системи на підозрілу активність з ініціацією CAPTCHA

Як можна побачити на рисунку, реалізований механізм здатний коректно відокремлювати користувачів з потенційно автоматизованою поведінкою, запобігаючи доступу до ресурсу до моменту проходження перевірки. Це дозволяє зменшити навантаження на сервер та підвищити безпеку без негативного впливу на легітимних користувачів.

Усі етапи тестування показали, що реалізовані модулі системи прикладного захисту працюють узгоджено та ефективно виконують свої функції. Виявлення аномальної активності, блокування підозрілих запитів, динамічне застосування CAPTCHA і обмеження частоти звернень забезпечують надійний захист сервера на прикладному рівні. Отримані результати підтверджують готовність системи до інтеграції у реальні вебресурси, а також її здатність протидіяти базовим формам DDoS-атак.

3.8 Висновки до розділу та оцінка ефективності реалізованих рішень

У наведеному розділі було виконано поетапну реалізацію та тестування системи прикладного захисту від DDoS-атак, орієнтованої на обробку HTTP-запитів на стороні сервера. На основі попереднього теоретичного аналізу було визначено набір найбільш доцільних механізмів захисту, зокрема: обмеження частоти звернень за IP-адресою, контроль інтервалів між запитами, фіксацію аномальної активності, динамічну реакцію на перевищення навантаження та додаткову перевірку через умовну CAPTCHA. Реалізація здійснювалась засобами мови PHP із використанням реляційної бази даних MySQL для зберігання журналів запитів та ведення списку підозрілих IP-адрес. Кожен модуль було реалізовано як самостійний функціональний блок, що дозволяє легко масштабувати або інтегрувати їх до існуючих вебзастосунків без втрати узгодженості логіки. Значну увагу було приділено точності виявлення аномалій, мінімізації навантаження на сервер та забезпеченню прозорості для легітимних користувачів.

Результати тестування в умовах змодельованого навантаження підтвердили коректність функціонування всіх компонентів захисту. Система успішно блокувала надмірні запити, виявляла повторювану активність, обмежувала частоту звернень і активувала перевірку лише для підозрілих джерел. Таким чином, розроблений засіб може бути використаний як основа для побудови більш комплексної системи захисту, або як легковаговий прикладний модуль для захисту невеликих вебресурсів.

Загалом, реалізоване рішення забезпечує базовий, але ефективний рівень протидії прикладним DDoS-атакам та демонструє потенціал використання простих алгоритмів на рівні серверного коду без залучення сторонніх сервісів або апаратних засобів.

4. Практична реалізація Анти-DDOS захисту на основі веб-сервісу візуального контенту: приклад реверс-проксі з перевіркою активності користувача

4.1 Розгортання тестового сайту

Для підготовки до практичного впровадження системи анти-DDoS захисту було попередньо розгорнуто тестовий вебсайт за доменною адресою https://testmydiplom.com. Домен було придбано через сторонній реєстратор, а обслуговування DNS-записів передано до платформи **Cloudflare**. Це дало змогу безкоштовно активувати сертифікат SSL/TLS та забезпечити підтримку протоколу **HTTPS**, що є важливою умовою для безпечної взаємодії користувача з сайтом.

Як стартовий лендинг-сайт було використано готовий односторінковий шаблон, взятий з відкритого репозиторію на платформі **GitHub**. Цей шаблон було адаптовано та розміщено на сервері для забезпечення початкового функціоналу. Основною метою цього етапу було створення базового середовища для подальшого тестування та інтеграції власного рішення з реверс-проксі і захистом від DDoS-атак.

Такий підхід дозволив заздалегідь перевірити працездатність HTTPS, стабільність домену та коректність маршрутизації через Cloudflare, що є важливою передумовою для коректної роботи механізму верифікації токенів та перенаправлення трафіку у реальному середовищі

4.2 Створення реверс-проксі з вбудованим захистом від DDoS-атак

На фінальному етапі було реалізовано повнофункціональний сервер на основі Node.js, який виконує роль **реверс-проксі з вбудованим захистом від DDoS-атак**. Його основною задачею є фільтрація запитів до бекенд-сервера,

перевірка користувача через візуальний challenge та збереження статистики мережевої активності.

```
const express = require('express');
const { createProxyMiddleware } = require('http-proxy-middleware');
const cookieParser = require('cookie-parser');
const config = require('./config.json');

const app = express();
app.disable('x-powered-by');
app.use(express.json());
app.use(cookieParser());

```

На початку реалізації серверу імпортуються необхідні Node.js-бібліотеки: express — для створення HTTP-сервера, http-proxy-middleware — для обробки проксі-запитів, cookie-parser — для роботи з cookie, а також config.json, у якому зберігаються всі ключові параметри конфігурації (порт, секретний ключ, поріг безпеки тощо).

Далі створюється сервер за допомогою express(), вимикається заголовок X-Powered-By (для безпеки), підключаються парсери JSON та cookie.

```
setInterval(() => {
  const now = Date.now();
  for (const [ip, data] of stats.requestCounts.entries()) {
    if (now - data.lastRequest > 60000) {
      stats.requestCounts.delete(ip);
    }
  }
}, 60000);

setInterval(() => {
  const now = Date.now();
  stats.rps = Math.round(stats.totalRequests / ((now - stats.lastMinute) / 1000));
  stats.lastMinute = now;
  stats.totalRequests = 0;

  if (stats.rps > config.security.highSecurityRpsThreshold) {
    stats.securityLevel = 'high';
    stats.highSecurityStartTime = now;
  } else if (stats.securityLevel === 'high') {
    const highSecurityDuration = config.security.highSecurityDurationMinutes * 60 * 1000;
    if (now - stats.highSecurityStartTime >= highSecurityDuration && stats.rps < config.security.highSecurityRpsThreshold) {
      stats.securityLevel = 'low';
      stats.highSecurityStartTime = null;
    }
  }

  console.clear();
  console.log(`[AntiDDoS Count] Unique IPs: ${stats.uniqueIPs.size} | Challenges: ${stats.challengeShown.size} | Passed: ${stats
}, 1000);
```

Щоб адаптивно реагувати на зміну мережевого навантаження, система відстежує основні метрики — кількість унікальних IP-адрес, загальну кількість запитів, кількість успішно авторизованих запитів і RPS (requests per second — запити за секунду). Усі ці параметри зберігаються у змінній stats.

Кожну секунду відбувається оновлення статистики: обчислюється новий RPS, при необхідності змінюється рівень безпеки (low ↔ high). Якщо RPS перевищує поріг, зазначений у config.security.highSecurityRpsThreshold, система автоматично переходить у режим “високої безпеки”. Через вказаний час (config.security.highSecurityDurationMinutes) вона повертається до нормального режиму, якщо навантаження спадає.

```
function generateToken(ip) {
  const timestamp = Date.now();
  const data = `${ip}:${timestamp}`;
  const signature = [...data + config.security.secret].reduce((a, b) => a * 31 + b.charCodeAt(0), 0);
  return Buffer.from(`${data}:${signature}`).toString('base64');
}

function validateToken(token, ip) {
  try {
    const decoded = const timestamp: any e64').toString('utf8');
    const [tokenIp, timestamp, signature] = decoded.split(":");
    if (tokenIp !== ip) return false;

    const tokenTime = parseInt(timestamp);
    const currentTime = Date.now();
    if (currentTime - tokenTime > config.security.tokenExpirationMinutes * 60 * 1000) return false;

    const data = `${tokenIp}:${timestamp}`;
    const expectedSignature = [...data + config.security.secret].reduce((a, b) => a * 31 + b.charCodeAt(0), 0);
    return signature === expectedSignature.toString();
  } catch {
    return false;
  }
}

function getRealIP(req) {
  const forwarded = req.headers['x-forwarded-for'];
  return (forwarded ? forwarded.split(',')[0] : null) ||
         req.headers['x-real-ip'] ||
         req.connection.remoteAddress ||
         req.socket.remoteAddress ||
         'unknown';
}
```

У рамках захисту від автоматизованих атак сервер використовує механізм **одноразових токенів**, які видаються лише перевіреним користувачам. Кожен токен містить IP-адресу клієнта, мітку часу та цифровий

підпис, згенерований на основі секретного ключа, заданого у config.security.secret.

Генерація токена (generateToken)

Коли користувач успішно проходить перевірку (challenge), сервер викликає функцію generateToken(ip), яка створює підписаний токен та відправляє його в cookie.

Перевірка токена (validateToken)

Кожен запит перевіряється функцією validateToken(token, ip), яка:

• декодує токен з base64;

• перевіряє, чи IP у токені збігається з поточним IP;

• перевіряє, чи токен не прострочений (наприклад, більше ніж 10 хвилин);

• обчислює цифровий підпис і порівнює з очікуваним.

Якщо токен некоректний, користувач отримує відмову в доступі.

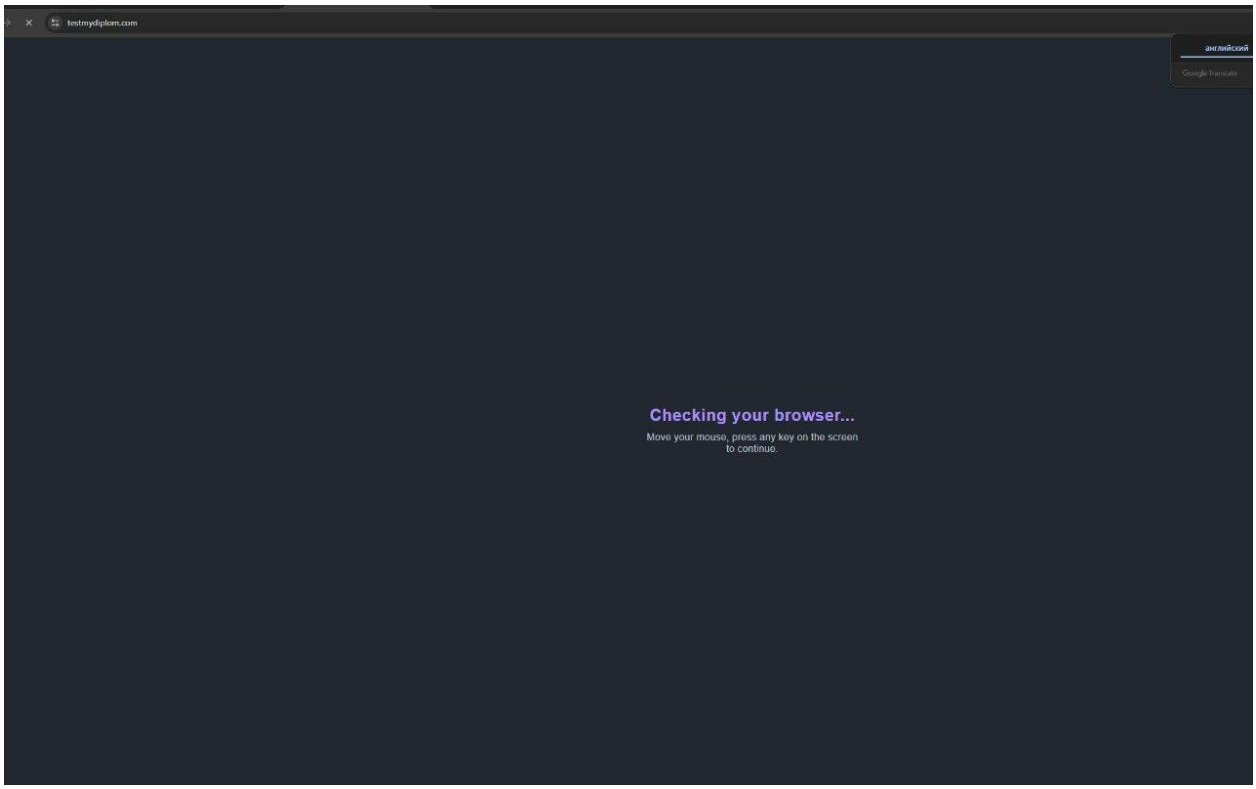

Рисунок 4.1 – Реакція системи на підозрілу активність

У режимі підвищеної безпеки сервер не дозволяє одразу проксирувати запит, а натомість відповідає спеціальною **challenge-сторінкою**. Її основна мета — перевірити, що користувач є реальною людиною, а не ботом.

HTML-сторінка реалізована у вигляді повноекранного темного інтерфейсу з повідомленням *“Checking your browser…”*, а також з інтерактивною анімацією (loader). Після затримки користувачу потрібно **здійснити активність** — рух миші, натиснення клавіші або торкання екрана.

Після цього браузер автоматично перенаправляється на /verify, додаючи до запиту токен, у якому вказані IP-адреса і час.

Цей механізм не лише блокує ботів, а й дає користувачеві **зрозумілий візуальний інтерфейс**. Він нагадує аналогічні перевірки у сервісах Cloudflare чи Google (*“just a moment…”*), що підвищує довіру та юзабіліті.

```
app.get('/verify', (req, res) => {
  const ip = getRealIP(req);

  try {
    const { r: token } = req.query;

    if (!token) {
      return res.status(400).send('Missing verification token');
    }

    const decodedToken = Buffer.from(token, 'base64').toString('utf8');
    const tokenData = JSON.parse(decodedToken);

    if (tokenData.ip !== ip) {
      return res.status(403).send('Bad Token');
    }

    const expirationTime = new Date(Date.now() + config.security.tokenExpirationMinutes * 60 * 1000);
    const newToken = generateToken(ip);

    const cookieOptions = {
      ...config.cookies.options,
      expires: expirationTime
    };

    res.cookie(config.cookies.name, newToken, cookieOptions);
    console.log(`✅ Verified user from ${ip}`);

    const originalUrl = req.query.redirect || '/';
    res.redirect(originalUrl);

  } catch (error) {
    console.error('Verification error:', error);
    res.status(400).send('Bad Request');
  }
```

Після взаємодії з challenge-сторінкою користувач автоматично перенаправляється на маршрут /verify, передаючи у параметрі r зашифрований токен із даними про IP та час. Сервер декодує токен, перевіряє його дійсність і, у разі успіху, видає **новий авторизаційний токен** у вигляді cookie.

Алгоритм перевірки:

1. Отримати токен з параметру ?r=....

2. Декодувати його з base64 та розпарсити JSON.

3. Перевірити, чи IP у токені збігається з реальним IP користувача.

4. Згенерувати новий токен і встановити його у cookie (res.cookie(...)) з терміном дії, визначеним у config.security.tokenExpirationMinutes.

Після цього користувача редиректить назад на вихідну адресу (/ або іншу вказану).

Цей механізм виконує роль фінального шлюзу: **тільки після підтвердження активності та IP** користувач отримує право доступу до основного контенту.

```
const transparentProxy = createProxyMiddleware({
  target: BACKEND_SERVER,
  changeOrigin: true,
  ws: true,
  timeout: config.proxy.timeout,
  proxyTimeout: config.proxy.proxyTimeout,
  onProxyReq: (proxyReq, req, res) => {
    const ip = getRealIP(req);
    proxyReq.setHeader('X-Forwarded-For', ip);
    proxyReq.setHeader('X-Real-IP', ip);
  },
  onProxyRes: (proxyRes, req, res) => {
  },
  onError: (err, req, res) => {
    console.error('Proxy error:', err.message);
    res.status(502).send('Backend server unavailable');
  }
});

app.use(async (req, res, next) => {
  const ip = getRealIP(req);
  stats.uniqueIPs.add(ip);
  stats.totalRequests++;

  const token = req.cookies[config.cookies.name];
  const isVerified = token && validateToken(token, ip);

  if (isVerified) {
    stats.passedRequests++;
  }

  next();
});
```

Перевірені користувачі (які мають дійсний токен у cookie) автоматично перенаправляються на бекенд-сервер за допомогою **reverse proxy**. Цей функціонал реалізовано через бібліотеку http-proxy-middleware, яка передає HTTP-запити на задану адресу, визначену у config.server.backend.

Проксі автоматично додає до кожного запиту заголовки X-Forwarded-For та X-Real-IP, щоб основний сервер знав реальну IP-адресу клієнта.

Також передбачено:

• підтримку WebSocket-з’єднань (ws: true);

• встановлення таймаутів;

• обробку помилок підключення до основного сервера (onError).

Цей компонент забезпечує гнучке й безпечне підключення до бекенду лише для **перевірених клієнтів**, і при цьому залишається прозорим для браузера — користувач бачить лише результат.

```
pp.use((req, res, next) => {
 const ip = getRealIP(req);

 if (req.path === '/verify') {
   return next();
 }

 const token = req.cookies[config.cookies.name];
 const isValidSession = token && validateToken(token, ip);

 if (isValidSession) {
   stats.passedRequests++;
   return transparentProxy(req, res, next);
 }

 if (!isValidSession) {
   const now = Date.now();
   const ipData = stats.requestCounts.get(ip) || { count: 0, lastRequest: now };

   if (now - ipData.lastRequest > 60000) {
     ipData.count = 0;
   }

   ipData.count++;
   ipData.lastRequest = now;
   stats.requestCounts.set(ip, ipData);

   if (ipData.count > 3) {
     return res.status(403).send('Too many requests');
   }
 }
```

Окрім глобального захисту на основі RPS, сервер реалізує **локальний захист на рівні кожного IP-адресу**. Якщо користувач без дійсного токена надсилає понад 3 запити протягом 60 секунд, сервер блокує його подальші звернення, повертаючи відповідь 403 Too Many Requests.

Це реалізовано за допомогою об’єкта stats.requestCounts, у якому зберігається:

• кількість запитів від конкретного IP (count);

• час останнього запиту (lastRequest).

Якщо між запитами проходить понад 60 секунд — лічильник обнуляється. Якщо ліміт перевищено — користувач отримає заборону доступу до ресурсу навіть у “low” режимі безпеки.

Це рішення запобігає надмірній кількості “тестових” або автоматичних запитів, навіть у період, коли захист неактивний, і гарантує стабільність сервера під час фонових навантажень. Також це можливо протестувати в режимі реального часу за доменною адресою testmydiplom.com

4.3 Створення стресеру для перевірки анти-DDos

Щоб переконатися в дієвості реалізованого анти-DDoS reverse proxy, було прийнято рішення провести **н**авантажувальне тестування з метою перевірки реакції системи на пікові запити. Для цього ми розробили власний скрипт stress.js, який дозволяє згенерувати високе мережеве навантаження на сервер за протоколом HTTPS.

Скрипт написаний мовою JavaScript з використанням Node.js, що дозволяє ефективно працювати з мережею на низькому рівні та створювати паралельні процеси за допомогою модуля cluster.

Скрипт запускається з командного рядка із зазначенням:

• цільової адреси сайту (testmydiplom.com),

• тривалості тесту (в секундах),

• кількості запитів на секунду,

• кількості паралельних потоків (worker’ів).

Для маскування джерел запитів використовується список проксі-серверів (у форматі ip:port) з файлу proxy.txt. Це дозволяє симулювати багатьох користувачів із різних IP-адрес.

Скрипт встановлює захищене з’єднання через проксі з використанням TLS. Потім поверх нього ініціюється HTTP/2-з’єднання, яке дозволяє надсилати декілька запитів у межах одного TCP-сеансу.

Задано набір реалістичних заголовків запитів:

• User-Agent (наприклад, Chrome),

• Accept, Accept-Encoding, Referer,

• X-Forwarded-For, sec-ch-* тощо.

Це робить трафік максимально подібним до звичайної активності користувача, що ускладнює його відрізнення від легітимного трафіку.

Для масштабування навантаження скрипт використовує Node.js cluster. Master-процес створює декілька worker-процесів, кожен з яких паралельно генерує запити. Таким чином, тест можна виконати з тисячами запитів на секунду.

4.4 Перевірка працездатності анти-DDos під навантаженням

Рисунок 4.2 – Запуск стресера для перевірки анти-DDOS

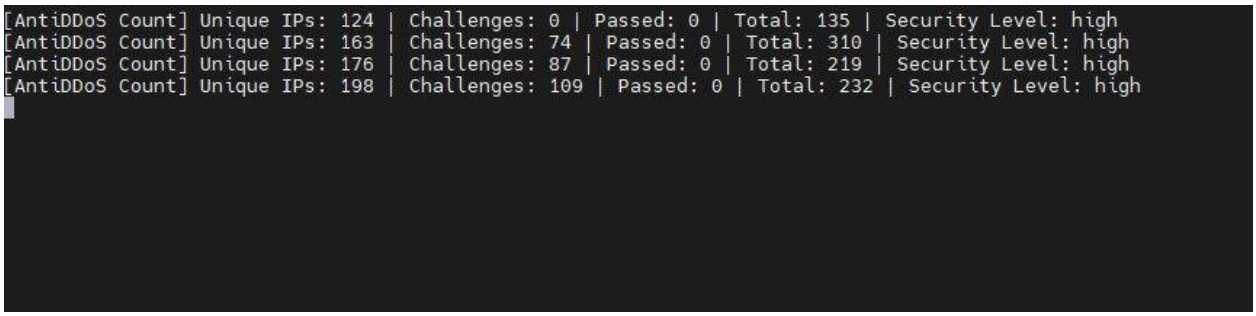

Рисунок 4.2 – Реакція системи анти-DDOS під час тестового навантаження

Видно, що зростає кількість унікальних IP-адрес та викликів перевірки (Challenge).

Рівень безпеки автоматично переходить у стан high, що означає активацію додаткового захисту — показу challenge-сторінки та блокування неперевірених запитів.

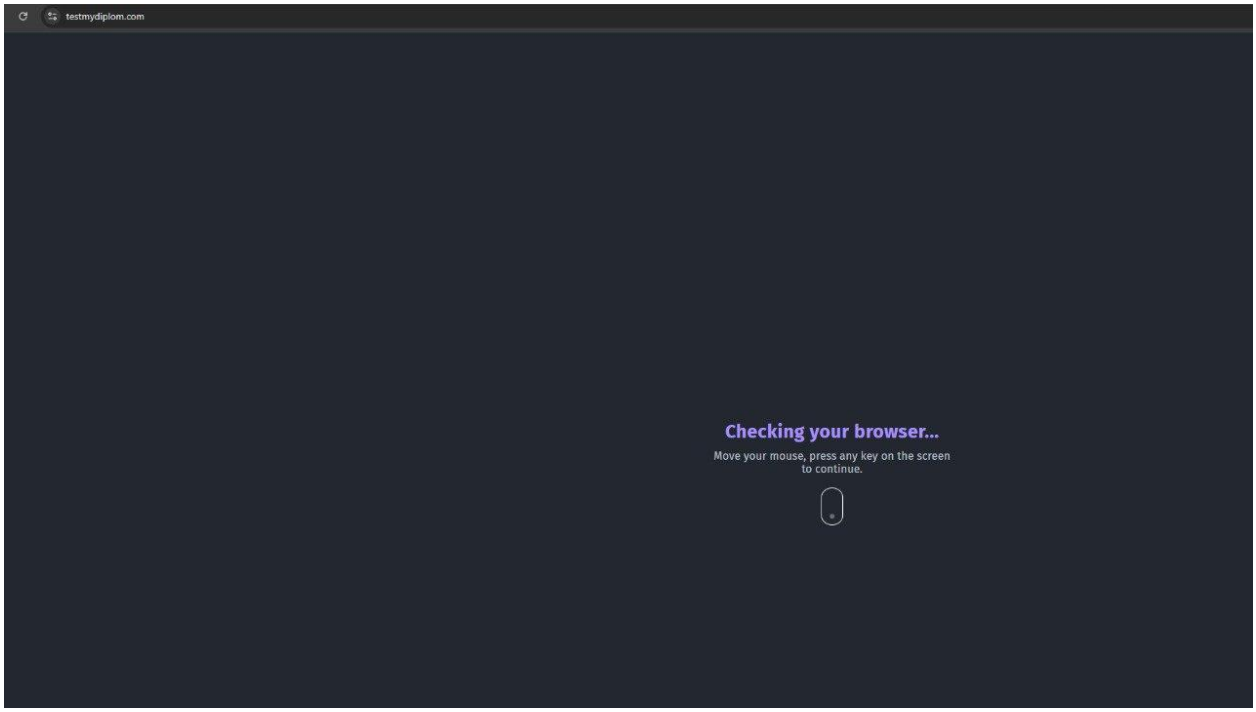

Рисунок 4.3 – Challenge-сторінка перевірки користувача, показана системою захисту

Під час режиму підвищеної безпеки reverse proxy автоматично перенаправляє неперевірених користувачів на цю сторінку. Для проходження

перевірки достатньо здійснити просту дію — рух мишки або натискання клавіші. Після цього видається токен доступу, який дозволяє користувачеві потрапити на сайт.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено і реалізовано систему захисту вебресурсу від DDoS-атак, побудовану на основі технології reverse proxy. Головна мета проєкту полягала в створенні гнучкого рішення, яке автоматично реагує на зміну мережевого навантаження, забезпечуючи доступ до ресурсу лише перевіреним користувачам, при цьому не порушуючи легітимний трафік. Реалізована система виконує безперервний моніторинг кількості вхідних запитів, визначає поточний рівень загрози і, у разі перевищення критичного порогу, активує режим підвищеного захисту. У цьому режимі користувачі, що не мають дійсного токена доступу, перенаправляються на спеціальну challenge-сторінку, де відбувається перевірка їхньої взаємодії з сайтом. Після проходження перевірки система видає унікальний токен, який дозволяє отримати доступ до основного ресурсу.

З метою перевірки працездатності системи було розроблено власний інструмент генерації навантаження — stress.js. Він дозволяє здійснювати багатопотокову генерацію HTTP/2-запитів через TLS-з’єднання з використанням проксі-серверів та максимально наближених до реальних заголовків запитів. Це дозволило протестувати поведінку сервера в умовах, максимально наближених до справжніх атак, включаючи високе навантаження, часту зміну IP-адрес та імітацію реального браузерного трафіку. Проведені експерименти підтвердили стабільність роботи системи, коректну зміну рівнів захисту, успішне блокування неперевірених запитів та збереження доступності ресурсу для легітимних користувачів.

Розроблене рішення вирізняється простотою інтеграції, відсутністю залежності від сторонніх сервісів, підтримкою TLS і HTTP/2, та можливістю масштабування під конкретні потреби вебпроєкту. Такий підхід дозволяє адаптувати систему під будь-який тип ресурсу, незалежно від його

функціонального призначення, кількості користувачів чи технічної архітектури. Таким чином, у межах дипломної роботи було не лише реалізовано актуальну і технічно складну задачу, але й досягнуто результату, який має практичну цінність і може бути використаний як основа для побудови надійних та стійких до атак систем у реальних умовах.

СПИСОК ЛІТЕРАТУРИ

1. Mirkovic J., Reiher P. DDoS Defense: Perspectives, Trends, and Challenges / Jelena Mirkovic, Peter Reiher. – Wiley, 2004. – 288 p.
2. Zargar S. T., Joshi J., Tipper D. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks / Sherif Zargar, Jeremy Joshi, David Tipper. – Computer Networks, 2013. – 50 p.
3. Antonatos S., Akritidis P., Markatos E. P. Defending Against Hitlist Worms Using Stackpiices / Stylianos Antonatos, Periklis Akritidis, Evangelos P. Markatos. – Springer, 2005. – 195 p.
4. Kumar S., Goyal N. PHP Web Security: Essential Training / Stefano Maffulli. – Packt Publishing, 2008. – 164 p.
5. Haug T. J. MySQL Administrator's Bible / Sheeri K. Cabral, Keith Murphy, Jon M. Janssens. – Wiley, 2006. – 812 p.
6. Kalmanowicz M., Pinter M. Web Application Security: PHP / Michal Kalmanowicz, Marcin Pinter. – Packt Publishing, 2011. – 226 p.
7. Viega J., Messier M. Secure Coding: Principles and Practices / Mark G. Graff, Kenneth R. van Wyk. – O'Reilly Media, 2003. – 368 p.
8. Kaufman C., Perlman R., Speciner M. Network Security: Private Communication in a Public World / Charlie Kaufman, Radia Perlman, Mike Speciner. – 2nd ed., Prentice Hall, 2002. – 560 p.
9. Hovav Shacham, Dan M. Boneh. Fast Public-Key Cryptography for Low-Power Devices / Dan Boneh, Hovav Shacham. – Wiley, 2000. – 428 p.
10. Al-Shaer E., Boutaba R. Configuring Network Security / Ehab Al-Shaer, Raouf Boutaba. – Morgan Kaufmann, 2004. – 384 p.
11. Han H., Liu H. Distributed Denial of Service Attacks and Defense Strategies / Hongyu Han, Hui Liu. – Springer, 2017. – 210 p.
12. Tammer Z. PHP and MySQL Web Development / Luke Welling, Laura Thomson. – 4th ed., Addison-Wesley, 2008. – 736 p.

13. Stuttard D., Pinto M. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws / Dafydd Stuttard, Marcus Pinto. – 2nd ed., Wiley, 2011. – 936 p.
14. Schmidt A. DDoS Protection and Network Security / Andreas Schmidt. – Syngress Publishing, 2014. – 480 p.
15. Mehani O., Vanbever L. Flow-Level Anomaly Detection in Network Traffic / Oliver Mehani, Laurent Vanbever. – Springer, 2019. – 152 p.
16. Goodwin P. Linux Server Security: Hack and Defend / Chris Binnie, Paul Goodwin. – 2nd ed., No Starch Press, 2008. – 550 p.
17. Zwicky E. D., Cooper S., Chapman D. Building Internet Firewalls / Elizabeth D. Zwicky, Simon Cooper, D. Brent Chapman. – 2nd ed., O'Reilly Media, 2000. – 350 p.
18. Roschke S., Feng D., Meinel C. Classification of DDoS Attacks and Common Countermeasures / Stefan L. C. Roschke, Duan Qiu Feng, Christoph Meinel. – IEEE International Conference on Computer Communications, 2010. – 12 p.

ДОДАТКИ

ДОДАТОК А

Програмний код

ddos_protection.php:

```php
<?php
$host = 'localhost';
$db = 'ddos_protection';
$user = 'root';
$pass = '';

$conn = mysqli_connect($host, $user, $pass, $db);
if (!$conn) die("Connection failed: " . mysqli_connect_error());

$ip = $_SERVER['REMOTE_ADDR'];
$uri = $_SERVER['REQUEST_URI'];
$timestamp = date('Y-m-d H:i:s');

$insertLog = "INSERT INTO traffic_log (ip_address, uri, request_time) VALUES
('$ip', '$uri', '$timestamp')";
mysqli_query($conn, $insertLog);

$limit = 100;
$intervalSeconds = 60;

$query = "
    SELECT COUNT(*) AS count FROM traffic_log
    WHERE ip_address = '$ip'
    AND request_time >= NOW() - INTERVAL $intervalSeconds SECOND
";
$result = mysqli_query($conn, $query);
$row = mysqli_fetch_assoc($result);

if ($row['count'] > $limit) {
    http_response_code(429);
    echo "Too Many Requests";
    exit;
}
?>
```

timeout_protection.php:

```php
<?php
session_start();

$timeoutSeconds = 10;

if (!isset($_SESSION['last_activity'])) {
    $_SESSION['last_activity'] = time();
} else {
    $elapsed = time() - $_SESSION['last_activity'];
    if ($elapsed > $timeoutSeconds) {
        http_response_code(408);
        echo "Request Timeout: Ви були неактивні понад $timeoutSeconds
секунд.";
        session_destroy();
        exit;
    }
}

$_SESSION['last_activity'] = time();
?>
```

log_anomaly.php:

```php
<?php
$host = 'localhost';
$db = 'ddos_protection';
$user = 'root';
$pass = '';

$conn = mysqli_connect($host, $user, $pass, $db);
if (!$conn) die("Connection failed: " . mysqli_connect_error());

$ip = $_SERVER['REMOTE_ADDR'];
$uri = $_SERVER['REQUEST_URI'];
$timestamp = date('Y-m-d H:i:s');

$insertLog = "INSERT INTO traffic_log (ip_address, uri, request_time) VALUES
('$ip', '$uri', '$timestamp')";
mysqli_query($conn, $insertLog);

$intervalMinutes = 1;
$checkQuery = "
    SELECT COUNT(*) AS count FROM traffic_log
    WHERE ip_address = '$ip' AND uri = '$uri'
    AND request_time >= NOW() - INTERVAL $intervalMinutes MINUTE
";
$result = mysqli_query($conn, $checkQuery);
$row = mysqli_fetch_assoc($result);

$anomalyThreshold = 20;
if ($row['count'] > $anomalyThreshold) {
    $insertSuspicious = "
        INSERT INTO suspicious_ips (ip_address, detected_time)
        VALUES ('$ip', NOW())
        ON DUPLICATE KEY UPDATE detected_time = NOW()
    ";
    mysqli_query($conn, $insertSuspicious);
}
?
```

captcha_check.php:

```php
<?php
session_start();

$host = 'localhost';
$db = 'ddos_protection';
$user = 'root';
$pass = '';

$conn = mysqli_connect($host, $user, $pass, $db);
if (!$conn) die("Connection failed: " . mysqli_connect_error());

$ip = $_SERVER['REMOTE_ADDR'];

$query = "SELECT * FROM suspicious_ips WHERE ip_address = '$ip'";
$result = mysqli_query($conn, $query);

if (mysqli_num_rows($result) > 0) {
    if (!isset($_SESSION['validated']) || $_SESSION['validated'] !== true) {
        if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
isset($_POST['captcha'])) {
            if ($_POST['captcha'] == $_SESSION['captcha_code']) {
                $_SESSION['validated'] = true;
                echo "CAPTCHA пройдено. Доступ дозволено.";
            } else {
                echo "Невірний код CAPTCHA. Спробуйте ще раз.";
            }
        } else {
            $_SESSION['captcha_code'] = rand(1000, 9999);
            echo '
                <form method="POST">
                    Введіть код: <strong>' . $_SESSION['captcha_code'] .
'</strong><br><br>
                    <input type="text" name="captcha" required>
                    <button type="submit">Підтвердити</button>
                </form>
            ';
            exit;
        }
    }
}
?>
```