МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувачка кафедри

___Катерина НЕСТЕРЕНКО
“____”_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: “Методика та програмний застосунок для запобігання кібератакам”

Виконавець: Зосимчук Олександр Русланович

Керівник: к.т.н доцент Радішевський Микола Федорович

Нормоконтролер: Кравченко Ольга Сергіївна

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувачка кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Зосимчука Олександра Руслановича

1. Тема кваліфікаційної роботи: «Методика та програмний застосунок для запобігання кібератакам»
затверджена наказом ректора від 29.09.2023 р. № 1994/ст.

2. Термін виконання кваліфікаційної роботи: з 02.10.2022 р. по 31.12.2023 р.

3. Вихідні дані до роботи: програмний застосунок розробити у вигляді додатку для браузерів Firefox, Chrome, Microsoft Edge, з використанням мови програмування Typescript.

4. Зміст пояснювальної записки:
1. Аналіз існуючих методів захисту від кіберзагроз.
2. Методика запобігання кібератакам.
3. Структура сервісів програмного застосунку.
4. Прототип програмного застосунку та перевірка його ефективності.

5. Перелік обов'язкових слайдів презентації:
1. Визначення кіберзагроз та їх вплив.
2. Існуючі методи захисту.
3. Криптографічні методи
4. Методика для протидії кібератакам
5. Структура програмного застосунку
6. Ефективність програмного застосунку

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Проведення консультації з науковим керівником щодо роботи	08.09.23 – 29.09.23	
2.	Аналіз літератури за темою кваліфікаційної роботи	01.10.23 – 06.10.23	
3.	Аналіз проблематики області дослідження	10.10.23 – 28.10.23	
4.	Аналіз архітектурних рішень для застосунку захисту від кібератак	12.10.23-16.10.23	
4.	Розробка розділу 3	04.11.23 – 15.11.23	
5.	Розробка розділу 3	15.11.23 – 11.12.23	
6.	Редагування та друк пояснювальної записки, графічного матеріалу	21.12.23	
7.	Проходження нормо-контролю	15.12.23 – 21.12.23	
8.	Передзахист.		
9.	Отримання рецензії.		
11.	Захист дипломної роботи перед ЕК		

Дата видачі завдання 02.10.2023 р.

Керівник дипломної роботи: к.т.н. доцент Микола РАДІШЕВСЬКИЙ

Завдання прийняв до виконання: Олександр ЗОСИМЧУК

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Методика та програмний застосунок для запобігання кібератакам»: 105 сторінок, 20 рисунків, 1 таблиця, 16 використаних джерел, 2 додатки.

ЗАПОБІГАННЯ КІБЕРАТАКАМ, МЕТОДИ ЗАХИСТУ ВІД КІБЕРЗАГРОЗ, ІНТЕГРАЦІЯ ЗАХОДІВ БЕЗПЕКИ, МЕРЕЖЕВА КІБЕРБЕЗПЕКА, ЗАХИСТ ПЕРСОНАЛЬНИХ ДАНИХ

Об’єкт дослідження - інтегрована методика та програмний застосунок для запобігання кібератакам у корпоративних мережах та інформаційних системах.

Мета роботи - підвищення ефективності захисту від кіберзагроз та програмний застосунку для протидії різноманітним кіберзагрозам та захисту від кібератак в організаційних середовищах.

Метод дослідження – моделювання даних веб-сайтів, аналіз випадків кібератак.

Результати дослідження мають потенціал бути використаними при створенні програмних засобів для запобігання кібератак. Нова методика та розроблений програмний застосунок можуть знайти застосування у різних галузях, де існує ризик кіберзагроз і можливість виникнення аварій або втрати конфіденційної інформації через вразливості в інформаційних системах. Проведення досліджень та розробка програми відбувалися в середовищі операційної системи, що піддається ризику - Windows. Для розробки програмного забезпечення використовувалася мова Typescript, C++. Цей підхід дозволяє створювати ефективні програмні засоби для виявлення та запобігання кібератак, забезпечуючи високий рівень безпеки в інформаційних системах

ABSTRACT

Explanatory note to the thesis "Methodology and software application to prevent cyber attacks ": 0 pages, 0 figures, 0 tables, 0 sources used.

PREVENTION OF CYBER ATTACKS, METHODS OF PROTECTION AGAINST CYBER THREATS, INTEGRATION OF SECURITY MEASURES, NETWORK CYBERSECURITY, PERSONAL DATA PROTECTION

Object of research – an integrated methodology and software application for preventing cyberattacks on corporate networks and information systems.

Purpose of the work – improving the effectiveness of cyber threat protection and software application to counteract various cyber threats and protect against cyber attacks in organizational environments.

Research method – modeling of website data, analysis of cyberattacks.

The results of the study have the potential to be used in the development of software tools to prevent cyberattacks. The new methodology and the developed software application can be used in various industries where there is a risk of cyber threats and the possibility of accidents or loss of confidential information due to vulnerabilities in information systems. The research and development of the program took place in the environment of the operating system at risk - Windows. Typescript and C++ were used for software development. This approach allows to create effective software tools for detecting and preventing cyberattacks, ensuring a high level of security in information system

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПДР — Правила дорожнього руху.

ТЗ — Транспортний засіб.

ВВП — Внутрішній валовий продукт.

ПЗ — Програмне забезпечення.

ІЧ — Інфрачервоний.

LAN (Local Access Network) — локальна мережа.

ARM (Advanced RISC Machine) — система описів команд для мікропроцессору.

RAM (Random Access Memory) — оперативна пам'ять.

NMS (Non maximum suppression) — принцип немаксимального подавлення значень.

CUDA (Compute Unified Device Architecture) — Архітектура паралельних вираховувань.

CPU (Central Processing Unit) — Центральний процесор.

GPU (Graphic Processing Unit) — Графічний процесор.

ROI (Region of interest) — Регіон уваги.

IOU (Intersection over union) — Пересікання через об’єднання.

ВСТУП

Створення методик та програмних застосунків для запобігання кібератак є актуальним завданням в галузі інформаційної безпеки. Сучасна інформаційна сфера переживає інтенсивний розвиток, але разом із зростанням технологічних можливостей збільшується й загроза кібератак. Кіберзлочинці використовують різноманітні методи, щоб завдати шкоди комп'ютерним системам та мережам.

Мета цієї роботи полягає у розробці ефективної методики та програмного застосунку для запобігання кібератак на сесію браузера користувача. З урахуванням швидкого розвитку кіберзагроз та постійного удосконалення атак, необхідно визначити оптимальні засоби та стратегії захисту інформаційних ресурсів.

У сучасному інформаційному середовищі кібербезпека вимагає комплексного підходу, оскільки загрози можуть виникати з різних напрямків. Розробка програмного забезпечення для виявлення, блокування та реагування на кібератаки є важливою складовою цього підходу. Методика виявлення кібератак в інтернет-просторі є важливою складовою безпечного інтернету. Суть методики полягає:

1. Санітизація всього вхідного і вихідного потоки даних з яким взаємодіє сторінка браузера
2. Превентивна заборона використання XSS ін’єкцій
3. Перевірка доменів з якими взаємодіє користувач
4. Фільтрація контенту з усіх ресурсів
5. Сувора перевірка міждоменних дозволів

Враховуючи постійне зростання кількості та складності кібератак, створення інноваційних засобів захисту стає насущною потребою.

Програмний застосунок є браузерним додатком який можна встановити навіть на телефон (середовище будь якого браузера) це робить додаток кроссплатформенним, унеможливлює атаки для всіх девайсів користувача, а ще й

допомогає вивченню методів, що використовуються кіберзлочинцями, для подальшого удосконалення систем захисту та виявленню нових типів атак.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗАХИСТУ ВІД КІБЕРЗАГРОЗ

1.1. Аналіз сучасних кіберзагроз

Сучасний стан цифрового світу надзвичайно динамічний, і це робить його більш вразливим перед кіберзагрозами. Кіберзлочинці постійно розвивають нові методи та техніки, щоб отримати доступ до конфіденційної інформації, завдати шкоди та використовувати її для власних цілей. Для ефективного захисту від кіберзагроз необхідно приділяти належну увагу вивченню та розумінню різних типів кіберзагроз та їх потенційних наслідків.

- Однією з основних загроз є кібератаки, які можуть бути спрямовані на інфраструктуру, корпоративні мережі чи індивідуальні комп'ютери.
- Однією з основних форм кібератак є віруси та інші види зловмисного програмного забезпечення, які можуть нанести шкоду системам, вкрадення конфіденційної інформації чи вимагати викуп.
- Важливо вживати заходів для постійного оновлення антивірусного програмного забезпечення та патчів безпеки для мінімізації ризиків.
- 99% усіх атак через мережу інтернет

Фішингові атаки також є поширеним видом кіберзагроз, де зловмисники намагаються отримати конфіденційну інформацію, використовуючи підступні методи, які здавалося б, надто надійні або довірчі. Підвищення освіти користувачів та впровадження механізмів перевірки легітимності електронних повідомлень є ключовими для запобігання таким атакам. Кіберзлочинці також можуть використовувати розкриття вразливостей у програмному забезпеченні та операційних системах. Тому важливо регулярно оновлювати програмне забезпечення, використовувати надійні паролі та встановлювати необхідні заходи безпеки.

Окрім того, розвиток інтернету речей (IoT) призводить до збільшення кількості пристроїв, які можуть бути потенційно піддані кібератакам. Забезпечення безпеки

всіх підключених пристроїв та вивчення їх можливих ризиків є надзвичайно важливим. Загальною стратегією для захисту від кіберзагроз є поєднання технічних, організаційних та освітніх заходів. Ефективна кібербезпека вимагає постійного моніторингу, оновлення захисних заходів та реагування на нові та змінюючіся загрози в цифровому середовищі. Найпоширенішими типами атак є:

1. Віруси:

Статистика: За даними дослідженням компанії McAfee, щоденно виявляється близько 600 нових варіацій вірусів.

Приклад атаки: WannaCry, який вразив комп'ютери та шифрував дані користувачів, вимагаючи викуп.

2. Соціальна Інженерія:

Статистика: За даними Verizon's 2021 Data Breach Investigations Report, соціальна інженерія є однією з найпоширеніших методів для вторгнень у системи.

Приклад атаки: Зловмисники, які вигадуються представниками технічної підтримки та переконують користувачів надавати конфіденційну інформацію.

3. Фішинг:

Статистика: За даними (APWG), було зафіксовано понад 241 000 унікальних фішингових сайтів у другому кварталі 2021 року.

Рис. 1.1. Кількість фішинг атак за 2020 рік

Приклад атаки: Фішингові листи, що імітують листи від банків, щоб викликати в користувачів надання особистих даних.

4. ДДОС-атаки:

Приклад атаки: Атака на онлайн-торговельну платформу, яка призвела до її недоступності.

Статистика: В середньому, за даними Statista, в 2020 році було зареєстровано близько 4 000 ДДОС-атак на день.

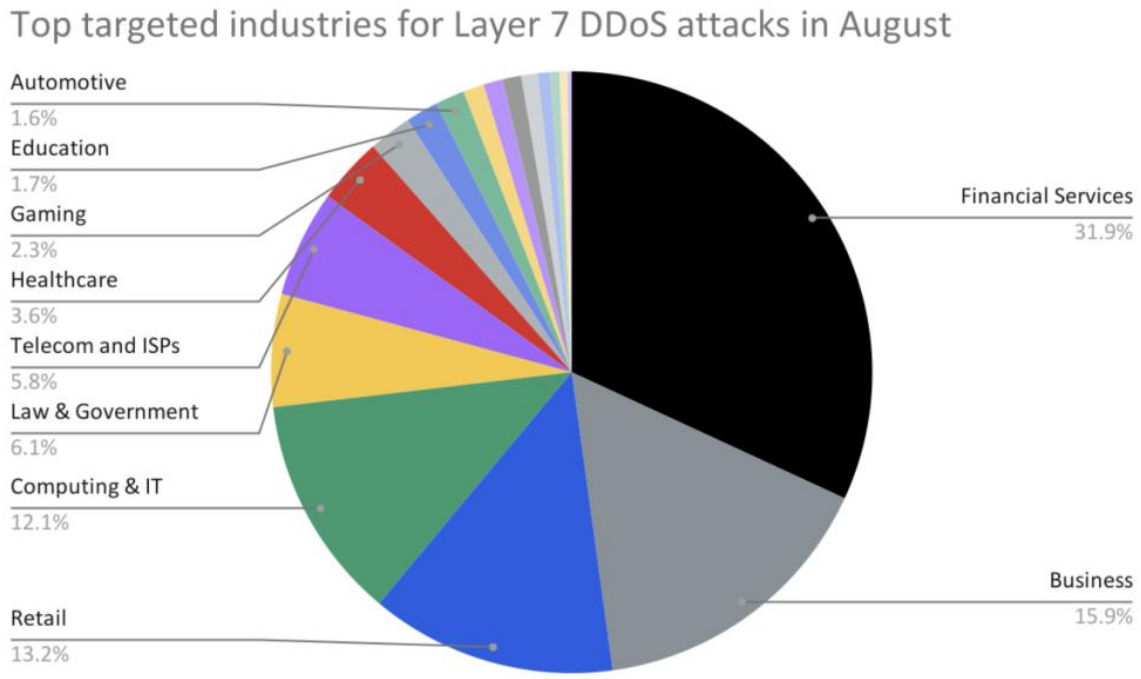

Рис. 1.2. Розподілення кількості DDOS атак за сферами діяльності 2020 рік

5. Рейнджом (Ransomware):

Приклад атаки: Атака WannaCry була серією кібератак, здійснених глобальним кіберзлочинним угрупованням, яке використовувало ботнет для розповсюдження шкідливого програмного забезпечення шифровальника WannaCry. WannaCry використовував уразливість у програмному забезпеченні Microsoft Windows, щоб отримати доступ до комп’ютерів і шифрувати їхні файли, а потім вимагати викуп у розмірі 300 доларів США у біткойнах.

Статистика: За даними Cybereason, в 2020 році було зафіксовано 350% зростання рейнсомвар атак порівняно з попереднім роком.

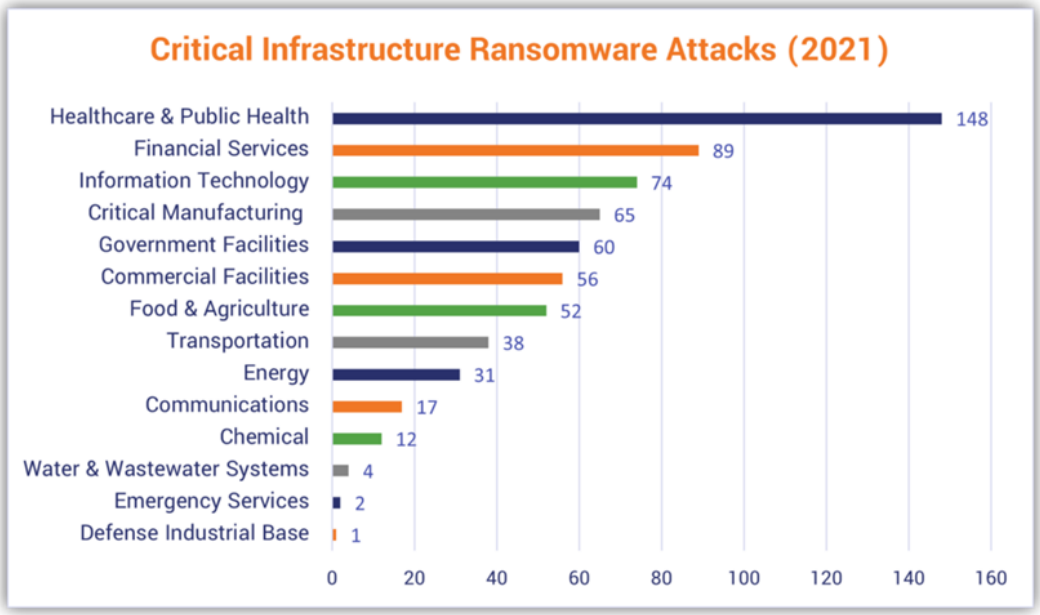

Рис. 1.3. Кількість Ransomware атак на критичну інфраструктуру США (2021 рік)

6. Витоки даних:

Статистика: За даними Statista, у 2020 році було зафіксовано понад 1000 витоків даних з загальною кількістю порядку 155 мільйонів порушених записів. Зазначені витоки даних можуть мати різноманітні причини, включаючи кібератаки, недоліки в безпеці програмного забезпечення та соціально інженернінг.

Приклад атаки: Витік даних з соціальної мережі TikTok, який призвів до розголошення особистої інформації користувачів. Повний дамп бази данних в 250GB був розміщений на просторах інтернету, будь який користувач міг увійти в аккаунт жертви

7. Зловмисна Програма (Malware):

Статистика: За даними Symantec's Internet Security Threat Report 2021, було зафіксовано понад 5 мільйонів нових варіацій зловмисного програмного забезпечення у 2020 році.

Приклад атаки: Поширення шпигунського програмного забезпечення SpyEye для викрадення конфіденційної інформації. Вона була активною у 2010-2012 роках і стала однією з найнебезпечніших загроз для фінансових установ та інших організацій SpyEye має широкий спектр функцій, спрямованих на крадіжку конфіденційних інформаційних даних. Це включає в себе викрадення банківських

реквізитів, логінів та паролів, персональних даних користувачів та іншої конфіденційної інформації.

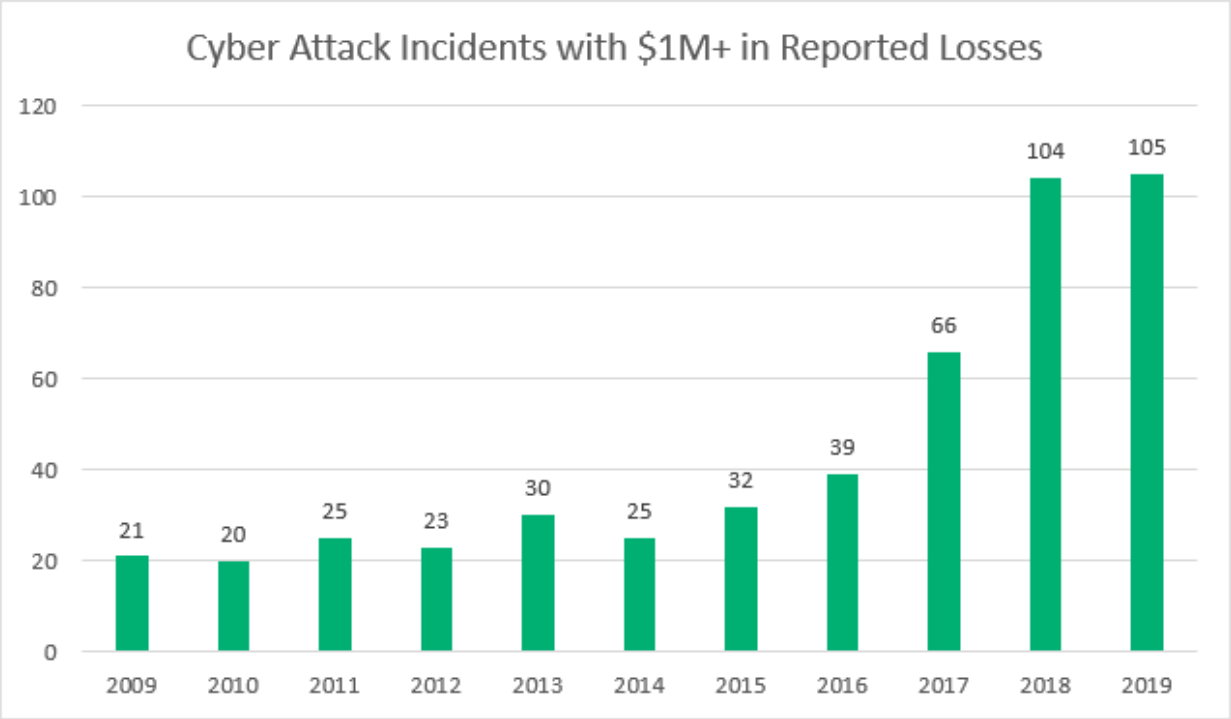

Рис. 1.4. Статистика випадків кібератак в результаті яких було втрачено більше 1 мільйона долларів

У результаті аналізу сучасних кіберзагроз виділяються кілька ключових аспектів. Віруси, фішинг, ДДОС-атаки, рейнсомвар, соціальна інженерія, витоки даних та зловмисна програма (Malware) є основними загрозами для інформаційної безпеки в сучасному цифровому світі. Статистика показує, що ці кіберзагрози не тільки існують, але й набувають наступницьких форм і стають все більш вдосконаленими. Кількість варіацій вірусів, фішингових сайтів та нових видів зловмисного програмного забезпечення постійно зростає. Далі, ДДОС-атаки стають більш масштабними і здатними завдати серйозних збитків онлайн-сервісам та компаніям. Рейнсомвар стає дедалі більш поширеним, а атаки соціальною інженерією залишаються важливим методом для вторгнень. Витоки даних стали набагато більшими та серйознішими за останні роки, розкриваючи особисту інформацію мільйонів осіб та підіймаючи питання про приватність та безпеку.

Зловмисна програма (Malware) залишається популярним інструментом для кіберзлочинців у здійсненні різних злочинних дій, від крадіжок даних до шпигунства.

Отже, аналіз сучасних кіберзагроз вказує на необхідність постійного удосконалення заходів кібербезпеки. Організації та індивіди повинні бути завжди на варті, слідкувати за новими трендами у сфері кібербезпеки, навчати персонал усвідомленим підходам до безпеки, а також вживати заходів для захисту своїх систем, даних і конфіденційної інформації. Кібербезпека стає невід'ємною частиною сучасного цифрового життя і вимагає постійного вдосконалення та відповідального підходу.

1.2. Існуючі методи захисту від кібератак

Відомі кібератаки, такі як WannaCry, Equifax Data Breach, SolarWinds Cyberattack і інші, наголошують на необхідності постійної боротьби зі зростаючими загрозами у цифровому просторі. У світі, який все більше цифрується та стає залежним від інтернету захист від кібератак - це завдання, яке стає більш складним і вимагає не тільки технічних знань, але й свідомості та усвідомлення ризиків. Захищати дані та інформаційні ресурси потрібно на всіх рівнях - від особистого користувача до великих корпорацій та урядових структур. Кібербезпека стає не тільки викликом, але і обов'язковою складовою для збереження конфіденційності, цілісності та доступності даних. Незалежно від вашого досвіду та ресурсів, існують шляхи для ефективного захисту від кібератак, і їх варто вивчити та впровадити.

1. Фільтрація трафіку - є важливим шаром захисту мережі та інфраструктури. Брандмауер блокує спроби несанкціонованого доступу до важливих ресурсів, це дозволяє зменшити вірогідність несанкціонованого доступу на 60% або більше.
2. Антивірусне програмне забезпечення (Antivirus Software)

Виявлення та блокування будь якого вірусу, який намагається виконати зловмисні операції неможливе без антивірусного ПЗ тому воно є ефективним у виявленні та блокуванні вірусів, але не завжди може захистити від всіх

кіберзагроз. Дослідження від AV-Test показують, що найкращі антивірусні програми виявляють понад 99% від усіх вірусів та атак на систему користувача, Найкращі антивіруси це:

- Symantec Endpoint Protection
- McAfee VirusScan
- Norton 360
- Microsoft Security Essentials
- Comodo Antivirus
- Avira Free Antivirus

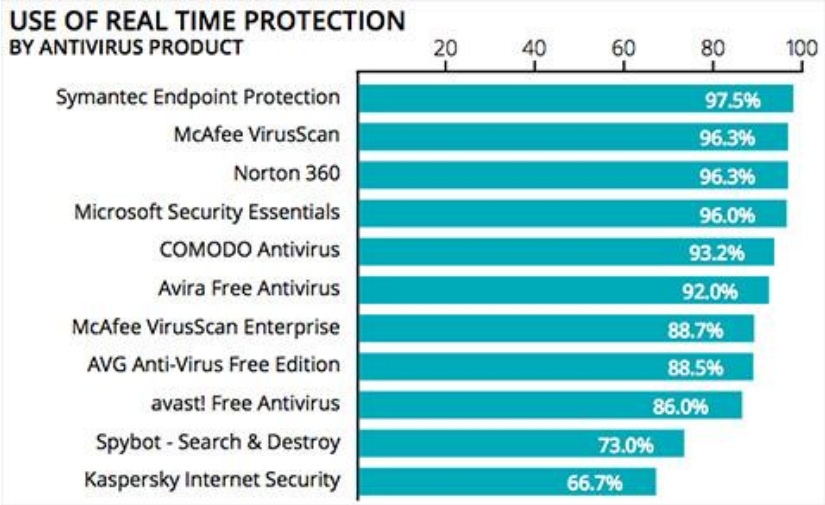

Рис. 1.5. Рейтинг антивірусів за якістю захисту в режимі реального часу

3. Шифрування - перетворює дані в такий формат, який може бути прочитаний лише за допомогою відповідного ключа. Це робить важким або неможливим доступ до інформації без відповідного авторизованого доступу. Використання шифрування може значно знизити ризик витоків конфіденційної інформації. Навіть якщо атакувач отримає доступ до зашифрованих даних, вони залишаються нечитабельними без правильного ключа.
4. Створення резервних копій (Backup) - За даними Acronis, 95% компаній, які регулярно створюють резервні копії, успішно відновлюють дані після кібератаки для цього використовують спеціальні програмні рішення або сервіси для управління резервними копіями, які дозволяють забезпечити

надійність і доступність резервних копій, а також виконувати їх регулярне оновлення.

5. Оновлення програмного забезпечення - За даними CVE Details, більшість експлойтів використовують вразливості, які вже були виправлені патчами. Оновлення програмного забезпечення є критично важливим для зменшення ризику використання вразливостей кіберзлочинцями.
6. Слідкування за безпекою (Security Monitoring) - За даними Ponemon Institute, середній час виявлення порушення безпеки складає 207 діб. Моніторинг безпеки дозволяє вчасно виявляти інциденти та зменшувати їхні наслідки. Виявлення аномальних активностей в мережі та реагування на них

Ці методи захисту є лише частиною комплексної стратегії кібербезпеки. Ефективність будь-якого методу може варіюватися залежно від контексту і рівня захисту, і рекомендується їх комбінувати для максимальної надійності та протидії атакам. Важливо враховувати те, що це не лише сукупність технічних заходів, але і система, що включає в себе людський фактор, процеси, технології та найкращі практики.

1.3. Вразливості програмного забезпечення

Вразливості програмного забезпечення є однією з основних точок входу для кіберзлочинців, які намагаються здійснити кібератаки. Ці вразливості можуть бути використані для виконання коду, який викрадає дані, завдає шкоди системі або навіть отримує несанкціонований доступ до комп'ютера чи сервера. Поширеними вразливостями є:

SQL Injection (SQLi) - це атака, при якій зловмисник вставляє SQL-код у вхідні дані, які передаються до бази даних. Це може призвести до незаконного доступу до даних у базі та витоку конфіденційної інформації. Приклад механізму дії, зловмисник вводить SQL-код у веб-форму або параметри URL. Наприклад, замість введення імені користувача, зловмисник вводить ' OR '1'='1' --, що

призводить до вибору всіх записів у таблиці. Статистика використання SQL Injection залишається однією з найпоширеніших вразливостей програмного забезпечення. Вона використовується у багатьох атаках, і за даними OWASP, вона часто потрапляє до списку найбільш поширених вразливостей.

Cross-Site Scripting (XSS):

XSS - це атака, під час якої зловмисник вставляє веб-скрипти (наприклад, JavaScript) у вміст сторінок, які потім виконуються на браузері користувача.Приклад механізму дії, зловмисник вставляє скрипти у ввідні поля або URL-параметри. Браузер користувача виконує ці скрипти, що може призвести до крадіжки сесійних файлів або інших атак. Статистика XSS залишається дуже поширеною вразливістю серед всіх інших. За даними різних джерел, вона складає від 20% до 40% від усіх виявлених вразливостей програмного забезпечення.

Вразливості в сторонніх бібліотеках та компонентах:

Використання сторонніх бібліотек та компонентів може вносити вразливості у програмне забезпечення, які не завжди очевидні для розробників.Приклад механізму дії, зловмисник може експлуатувати вразливості у сторонніх бібліотеках, які використовуються в програмі. Наприклад, вразливість у бібліотеці для обробки зображень може призвести до виконання зловмисного коду при завантаженні зображення. Вразливості сторонніх бібліотек і компонентів є дуже поширеними. Вони часто використовуються зловмисниками для атак на програмне забезпечення.

Використання застарілих або вразливих бібліотек:

Використання застарілих або вразливих бібліотек у програмному забезпеченні може призвести до вразливостей. Зловмисники можуть атакувати ці бібліотеки, знаючи їхні вразливості. Приклад механізму дії, якщо розробник не оновлює застарілу бібліотеку з відомою вразливістю, зловмисники можуть використовувати цю вразливість для виконання атак. Використання застарілих бібліотек є досить

поширеним явищем. Наприклад, застаріла версія бібліотеки OpenSSL була використана в атаках типу Heartbleed.

Вразливості в аутентифікації та управлінні сеансами:

Вразливості, пов'язані з аутентифікацією та управлінням сеансами, можуть дозволити зловмисникам обходити авторизацію або крадіжку сесій користувачів. Наприклад, недостатньо безпечний механізм сесій може дозволити зловмиснику підміняти сесійні ідентифікатори та виходити на чужий обліковий запис. Вразливості в аутентифікації та управлінні сеансами часто використовуються в атаках, пов'язаних з вторгненнями в системи.

Вразливості відмови в обслуговуванні (DoS) і атаки на переповнення буфера:

Атаки DoS спрямовані на перевантаження системи, роблячи її недоступною для законних користувачів. Атаки на переповнення буфера використовують вразливості, щоб вивести програму з ладу або виконати зловмисний код. Наприклад, атака DoS може надсилати велику кількість запитів на сервер, перевантажуючи його ресурси та призводячи до відмови обслуговування. Атаки DoS і атаки на переповнення буфера стали дуже популярними. Вони можуть використовувати різні методи і навіть використовувати ботнети для розподіленого перевантаження.

Вразливості взаємодії з користувачем:

Деякі вразливості можуть виникнути внаслідок недостатньої валідації користувацького введення або недостатньої обробки даних, введених користувачем. Це може призвести до атак, таких як впровадження зловмисних файлів або кросс-сайтового скриптингу. Зловмисник може вставити зловмисний код у текстове поле або завантажити файл з шкідливим вмістом, який може бути виконаний на стороні сервера або клієнта. Вразливості взаємодії з користувачем можуть бути використані у різних типах атак, і вони залишаються актуальними для зловмисників.

Вразливості в роботі з файлами:

Використання небезпечних функцій для роботи з файлами, недостатня перевірка шляхів або недостатні обмеження можуть призвести до атак типу Local File Inclusion (LFI) або Remote File Inclusion (RFI). Зловмисник може використовувати недостатньо захищену функцію для завантаження чи читання файлів на сервері або віддаленому ресурсі. Вразливості в роботі з файлами залишаються актуальними, і їх використання може призвести до витоку чутливої інформації або виконання зловмисного коду.

Вразливості в мережевих протоколах:

Вразливості у мережевих протоколах можуть бути використані для атак на мережевий рівень, включаючи атаки на перехоплення пакетів, аналіз трафіку та інші. Зловмисник може використовувати вразливості у протоколах, щоб перехопити чи модифікувати комунікацію між системами. За даними Common Vulnerabilities and Exposures (CVE), існує значна кількість вразливостей, пов'язаних з мережевими протоколами.

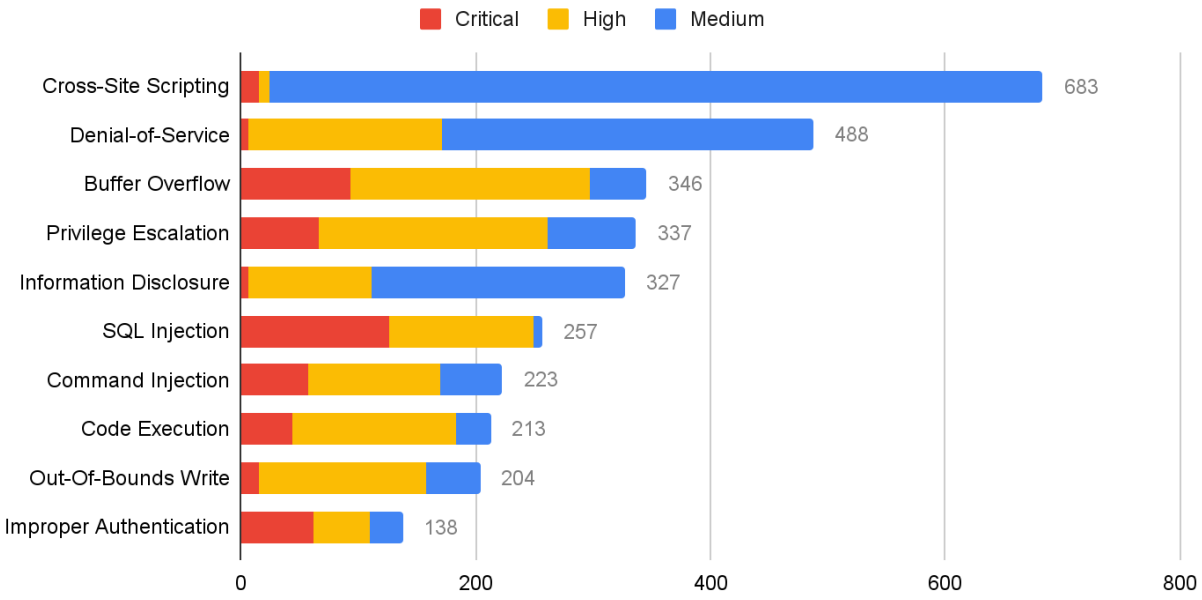

Рис. 1.6. Кількість атак на веб-додатки за типом

1.4. Законодавча база та регулювання кібербезпеки

Країни активно вдосконалюють свою законодавчу базу та регулювання кібербезпеки відповідно до стрімкого розвитку технологій та зростаючих загроз кібернетичної безпеки. Ці заходи спрямовані не лише на захист особистої інформації та даних, але й на зміцнення відповідальності у сфері кіберпростору для забезпечення стійкості економіки та національної безпеки.

Загальні закони про кібербезпеку:

Багато країн впроваджують загальні закони про кібербезпеку, які встановлюють основні вимоги та стандарти для захисту кіберпростору. Наприклад, в США, Центральне управління з кібербезпеки і інфраструктури (CISA) регулює кібербезпеку та відповідає за захист критичної інфраструктури.

Закони про обов'язкове повідомлення про порушення безпеки даних:

Багато країн впроваджують закони, які вимагають від організацій повідомляти про порушення безпеки даних і витоки конфіденційної інформації. Наприклад, в Європейському Союзі це регулюється Загальним регламентом про захист персональних даних (GDPR).

Закони про кіберзлочини:

В багатьох країнах існують закони, які визначають кіберзлочини та передбачають відповідальність за них. Це включає в себе атаки, такі як хакерський вторгнення, крадіжка даних та розповсюдження вірусів. Наприклад, в США, Закон про комп'ютерну злочинність (Computer Fraud and Abuse Act) регулює кіберзлочини.

Закони про кібербезпеку у критичних секторах:

Багато країн мають спеціальні закони та стандарти щодо кібербезпеки у критичних секторах, таких як енергетика, транспорт, фінанси та медицина. Ці закони накладають обов'язки з захисту критичної інфраструктури від кіберзагроз.

Міжнародна співпраця:

Кібербезпека також стала предметом міжнародної співпраці. Наприклад, Інтернет-простір регулюється міжнародними організаціями, такими як Інтернет-корпорація з призначення і надання доменних імен (ICANN), інтернет-інженерною робочою групою з надійності (IETF) та іншими. Організації, такі як Національний інститут стандартів і технологій (NIST) у США та Міжнародна організація зі стандартизації (ISO), розробляють стандарти та рамки для кібербезпеки, які використовуються у всьому світі. Ці законодавчі та регуляторні заходи призначені для захисту цифрового простору від кіберзагроз та забезпечення безпеки користувачів та організацій. Вони продовжують розвиватися і адаптуватися до нових викликів у сфері кібербезпеки.

Заходи щодо кібербезпеки у галузі критичної інфраструктури:

Багато країн приділяють особливу увагу заходам щодо кібербезпеки в галузі критичної інфраструктури, такі як енергетика, водопостачання, транспорт та інші. Це включає в себе регулярну оцінку ризиків, обов'язкові стандарти та вправний нагляд.

Цифрова аутентифікація та криптографія:

Цифрова аутентифікація та криптографія відіграють ключову роль у забезпеченні безпеки інформації в сучасному цифровому середовищі. Багато країн встановлюють стандарти для використання цих технологій з метою захисту даних та забезпечення конфіденційності, цілісності та доступності інформації.

Заходи щодо кібербезпеки в урядових органах:

Урядові органи впроваджують спеціальні заходи щодо кібербезпеки, включаючи захист важливих інформаційних ресурсів та навчання співробітників щодо безпеки.

Інтернаціональна співпраця:

Кібербезпека часто стає об'єктом міжнародної співпраці. Країни спільно розробляють стратегії та домовленості щодо кібербезпеки, в тому числі щодо обміну інформацією про кіберзагрози та спільних вправ для реагування на інциденти. Ця співпраця визначається як важливий елемент у зміцненні глобальної стійкості кіберпростору, оскільки кіберзагрози часто перетинають національні кордони. Міжнародна обмін інформацією та співпраця в галузі кібербезпеки сприяють покращенню загального рівня захисту інформаційної інфраструктури та попередженню масштабних кібератак. Додатково, узгоджені міжнародні підходи допомагають вирішувати виклики, пов'язані із кіберзлочинністю та кібершпигунством, і сприяють створенню відкритого та безпечного кіберсередовища для глобального співтовариства.

Висновки до розділу

У цьому розділі було проведено аналіз різноманітних методів захисту від кіберзагроз з метою розкриття їхньої ефективності та недоліків. Виявлені підходи до кібербезпеки надають краще розуміння стратегії захисту, що стає все більш актуальним у світі, де цифрові технології стають необхідною складовою всіх сфер життя. Одним із ключових висновків з аналізу є те, що загрози в цифровому просторі стають все більш розмаїтими та складними. Зловмисники постійно вдосконалюють свої техніки, використовуючи нові методи інструкцій та експлойтів, тим самим роблячи виклики для існуючих систем захисту. Важливо зауважити, що жодна одноразова стратегія не може гарантувати повністю ефективний захист і відповідь на кіберзагрози повинна бути комплексною та адаптивною. Однією з переваг, виявлених у дослідженні, є важливість багаторівневої аутентифікації та багатофакторної аутентифікації в системах безпеки. Вони забезпечують додатковий шар захисту, ускладнюючи завдання несанкціонованого доступу. Також виокремлена роль криптографії в забезпеченні конфіденційності та цілісності інформації, а також у встановленні захищених з'єднань в мережі Інтернет.

Однак, варто відзначити, що жодна система не є абсолютною. Безпека - це постійний процес, і компанії та організації повинні постійно адаптувати свої стратегії відповідно до змін у загрозах кібербезпеки.

РОЗДІЛ 2
МЕТОДИКА ЗАПОБІГАННЯ КІБЕРАТАКАМ

2.1. Комплекс методів для протидії кібератакам на браузер користувача

Кібератаки та загрози у веб-просторі є серйозною проблемою для сучасного світу. Вони можуть завдати значної шкоди як окремим особам, так і організаціям, на сьогоднішній день всесвітня мережа є основним джерелом загроз для кожного користувача. Для захисту від цих загроз необхідно застосовувати комплекс методів який повністю контролює трафік, створює сувору політику HTTP запитів для кожного сайту та вміє розуміти контент кожної сторінки. Багато виробників антивірусного програмного забезпечення разом зі своїм нативним додатком встановлюють на комп'ютер користувача браузерний додаток, тому що браузер користувача можна атакувати незалежно від операційної системи. У контексті браузерного розширення, комплекс методів для протидії кібератакам та загрозам у веб-просторі може включати в себе наступне:

- Технічні заходи
    1. Використання сучасних технологій для аналізу трафіку. Такі компанії як Amazon, Google, Microsoft надають сервіси для моніторингу ресурсів та подій системи своїх клієнтів, що напряму впливає на швидкість виявлення можливих загроз
    2. Браузерне розширення може використовувати сучасні технології, такі як машинне навчання та штучний інтелект, для аналізу трафіку та виявлення потенційних загроз.
    3. Використання сигнатурних баз для виявлення відомих загроз чи фішингових сайтів
    4. Використання поведінкових моделей для виявлення потенційних загроз. Браузерне розширення може використовувати поведінкові моделі для виявлення потенційних загроз, таких як зловживання ресурсами системи або аномалії в поведінці користувача.
- Організаційні заходи

1. Освіта користувачів про кібербезпеку. Кожна велика організація проводить тренінги з працівниками щодо того як запобігти фішингові атаки, виявляти підозрілі email листи та невідомі файли
2. Розробка та впровадження політик та процедур, що регулюють операції з персональними даними. Організації повинні розробити та впровадити політики та процедури, що унеможливлюють витік конфіденційної інформації
3. Встановлення відповідного програмного забезпечення на всі корпоративні пристрої що мають доступ до чутливої інформації

На основі вищезазначеного, можна виділити наступні конкретні заходи, які можна застосувати для протидії кібератакам та загрозам у веб-просторі через перевірку трафіку:

- Аналіз заголовків HTTP. Браузерне розширення може аналізувати заголовки HTTP, щоб виявити потенційні загрози, такі як шкідливі URL-адреси або несанкціоновані спроби доступу до систем. Одними з найважливіх заголовків безпеки є:

Таблиця 2.1

Заголовки що впливають на безпеку сторінок

Назва заголовку	Призначення	Рекомендоване значення
Content-Type	Використовується для позначення оригінального медіа-типу ресурсу (перед тим, як для надсилання буде застосовано будь-яке кодування вмісту).	Відповідно до розширення файлу
Set-Cookie	Використовується для надсилання файлу cookie із сервера до агента користувача, щоб агент користувача міг надіслати його назад на сервер пізніше. Це не заголовок безпеки сам по собі, але його атрибути безпеки є вирішальними	none

Закінчення таблиці 2.1.

Strict-Transport-Security	заголовок вказує браузеру використовувати тільки захищене з'єднання HTTPS з сервером і виключити можливість встановлення незахищених з'єднань.	Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Referrer-Policy	Заголовок вказує, які дані про посилання передаються при переході на інші сторінки.	no-referrer
X-Frame-Options	Встановлює, чи дозволяється браузеру вкладати сторінку в <frame>, <iframe>, <object> і т.д. Це захищає від атак типу Clickjacking	X-Frame-Options: DENY
X-XSS-Protection	Включає вбудований фільтр браузера проти атак типу Cross-Site Scripting (XSS).	X-XSS-Protection: 1; mode=block

- Аналіз поведінки користувача. Браузерне розширення може аналізувати поведінку користувача, щоб виявити потенційні загрози, такі як зловживання ресурсами системи або аномалії в поведінці користувача. Це забезпечує можливість вчасного виявлення та реагування на надзвичайні ситуації або атаки, що базуються на аномальних змінах у звичайному способі використання системи також може виявляти потенційно шкідливі дії, такі як спроби фішингу чи використання шкідливих сценаріїв, що забезпечує додатковий шар захисту в онлайн-середовищі.

Referer	Placement	Leads	Depth	Payout	Ip	Isp	Device	Keyword
https://l.facebook.com/	placement	0	0.00	$0.00	41.116.115.xxx	MTN Business Solutions	Generic Windows PC	keyword
http://m.facebook.com/	placement	0	0.00	$0.00	209.203.23.xxx	Vox Telecom	Android Generic	keyword
http://m.facebook.com/	placement	0	0.00	$0.00	102.249.6.xxx	Telkom Internet	Android Generic	keyword
http://m.facebook.com/	placement	0	0.00	$0.00	102.249.6.xxx	Telkom Internet	Android Generic	keyword
https://l.facebook.com/	placement	0	0.00	$0.00	197.95.33.xxx	OPTINET	Android Generic	keyword
https://l.facebook.com/	placement	0	0.00	$0.00	105.226.90.xxx	Telkom Internet	Huawei ANE-LX2	keyword
https://l.facebook.com/	placement	0	0.00	$0.00	196.250.151.xxx	IBITS Internet	Android Generic	keyword
http://m.facebook.com/	placement	0	0.00	$0.00	41.117.136.xxx	MTN Business Solutions	Android Generic	keyword
http://instagram.com/	placement	0	0.00	$0.00	102.32.213.xxx	Metrofibre Networx	Samsung Generic	keyword
no referer	placement	0	0.00	$0.00	41.0.129.xxx	Vodacom Business	Android Generic	keyword
http://instagram.com/	placement	0	0.00	$0.00	41.145.193.xxx	Telkom Internet	Samsung SM-G965U	keyword
http://instagram.com/	placement	0	0.00	$0.00	41.145.193.xxx	Telkom Internet	Samsung SM-G965U	keyword
https://l.facebook.com/	placement	0	0.00	$0.00	197.99.140.xxx	Internet Solutions	Hisense Infinity E30 Lite	k

Рис. 2.1. Трекінг сесій користувача для вебсайтів Instagram.com та Facebook.com, червоні рядки – заблоковано, жовті – потенційна небезпека, білі – авторизовано

Виявлення аномалій – це процес визначення точок у наборі даних або системі, які виходять за межі норми. Під час аналізу даних або за допомогою машинного навчання виявлення аномалій позначатиме екземпляри, які не відповідають звичайним шаблонам або статистичним моделям у більшості ваших даних. Аномалії можуть виглядати як викиди, несподівані зміни або помилки — це залежить від типу даних, які аналізуються, і будь-яких попередньо визначених параметрів, які ви встановили. Виявлення аномалій корисно, оскільки швидко й ефективно можна виявляти потенційні проблеми чи загрози та підтримувати цілісність і надійність вашої системи. Окремо можна виділити

- Аналіз даних, що передаються через HTTP. Браузерне розширення може аналізувати дані, що передаються через HTTP, щоб виявити потенційні загрози, такі як шкідливе програмне забезпечення або дані, що містять конфіденційну інформацію.

- Перевірка сертифікатів. Браузерне розширення може аналізувати сертифікати безпеки, щоб впевнитися, що вони дійсні і належать правильному серверу. Особлива примета всіх фішингових сайтів – вони використовують самопідписні безкоштовні сертифікати, в той час як всі технічні гіганти використовують сертифікати відомих служб, таких як Symantec, DigiCert, Let's Encrypt тощо. Також перевірка сертифікату допомагає уникнути атак типу "Man-in-the-Middle"(MITM). Атака MITM виникає, коли зловмисник знаходиться між комунікуючими сторонами, перехоплює або модифікує передавані дані. Основна ідея полягає в тому, що браузерне розширення може аналізувати сертифікати SSL/TLS, які використовуються для забезпечення шифрування трафіку між користувачем і сервером. Якщо сертифікат визнається недійсним, або відомості в ньому не відповідають відомостям сервера, це може свідчити про можливий MITM.

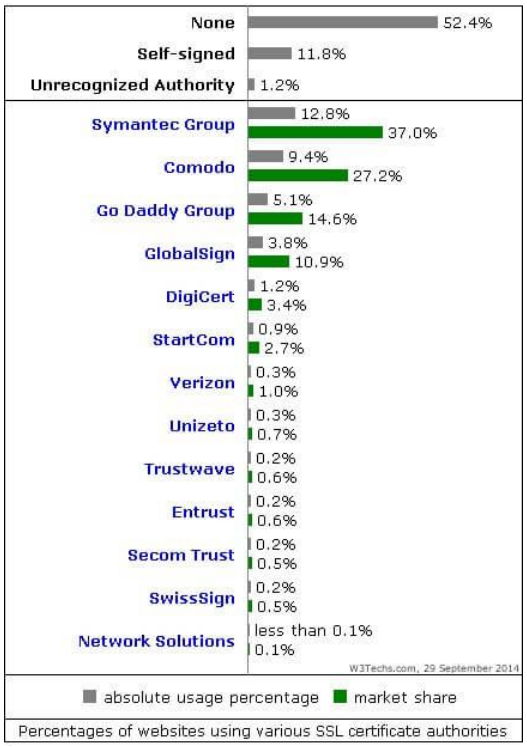

Рис. 2.2. Розподілення використання сертифікатів за центрами авторизації

- Політика безпеки вмісту (CSP) — це стандарт безпеки, який забезпечує додатковий рівень захисту від міжсайтових сценаріїв (XSS), клікджекінгу та інших атак із впровадженням коду. Це захід захисту від будь-яких атак, які покладаються на виконання шкідливого вмісту в надійному веб-контексті, або інші спроби обійти політику того самого походження. За допомогою CSP ви можете обмежити джерела даних, дозволені веб-додатком, визначивши відповідну директиву CSP у заголовку відповіді HTTP запиту до серверу.

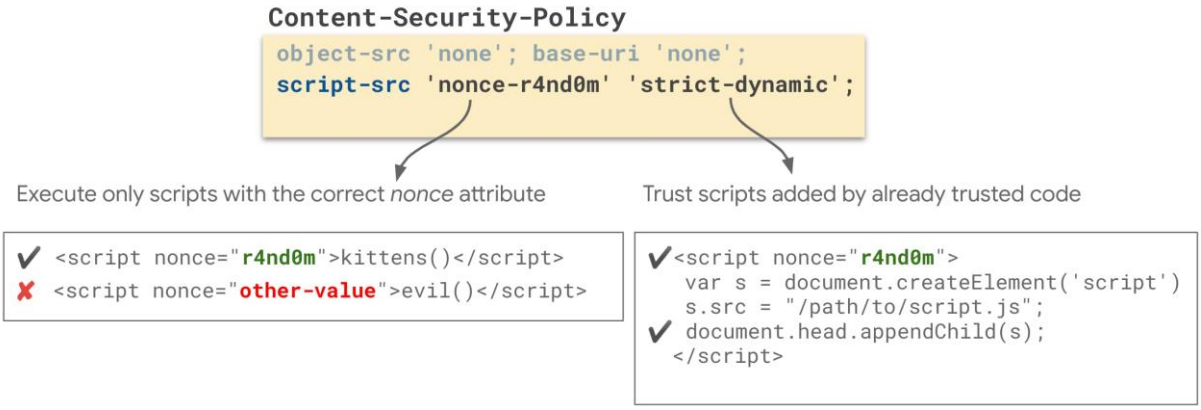

Рис. 2.3 Content-Security-Policy принцип дії

Як приклад реалізації комплексного підходу до протидії кібератакам та загрозам у веб-просторі через перевірку трафіку в контексті браузерного розширення, можна розглянути наступний сценарій:

1. Браузерне розширення використовує сучасні технології для аналізу трафіку, включаючи машинне навчання та штучний інтелект
2. Розширення також використовує сигнатурну базу для виявлення відомих шкідливих програм та інших загроз
3. Використання поведінкових моделі для виявлення потенційних загроз, таких як зловживання ресурсами системи або аномалії в поведінці користувача.
4. Примусова модифікація правил сторінки для запобігання різних типів загроз

2.2. Розпізнання потенційно шахрайських веб-сайтів шляхом аналізу URL адреси та мережево-доменних характеристик

Багато користувачів щодня і щогодини мимоволі переходять на фішингові домени. Зловмисники націлені як на користувачів, так і на компанії. Згідно з 3-м звітом Microsoft Computing Safer Index Report, опублікованим у лютому 2015 року, щорічні збитки від фішингу у світі можуть сягати 6 мільярдів доларів США. Основна причина - недостатня обізнаність користувачів. Великі компанії впроваджують нові рішення безпеки, щоб не допустити зіткнення користувачів з цими шкідливими сайтами. Запобігання цим величезним витратам може початися з підвищення обізнаності людей на додаток до побудови надійних механізмів безпеки, які здатні виявляти і запобігати потраплянню фішингових доменів до користувача.

Характерні ознаки фішингових доменів

Уніфікований покажчик ресурсів (URL) створюється для адресації веб-сторінок. На малюнку нижче показані відповідні частини в структурі типової URL-адреси.

Рис. 2.4. Структура URL ресурсу Microsoft

Зловмисник може зареєструвати будь-яке доменне ім'я, яке не було зареєстровано раніше. Ця частина URL може бути задана тільки один раз. Фішер може будь-коли змінити FreeURL, щоб створити нову URL-адресу. Причина, по якій захисникам безпеки важко виявляти фішингові домени, полягає в унікальній частині домену веб-сайту (FreeURL). Коли домен виявлено як шахрайський, його можна легко заблокувати до того, як користувач отримає до нього доступ. Деякі компанії, що займаються виявленням загроз, виявляють і публікують шахрайські

веб-сторінки або IP-адреси в чорних списках, таким чином, запобігаючи використанню цих шкідливих ресурсів іншими особами, наприклад проект firehol.

Розглянемо URL: https://visa.com-appuser.active-content.com/webapps/12101

Таблиця 2.2

Структура URL зловмисного ресурсу

Protocol	https://
Domain name	active-content.com
Path	/webapps/12101
Subdomain #1	com-appuser
Subdomain #2	visa

Справжнє доменне ім'я - active-content.com, в цьому випадку зловмисник намагався зробити домен схожим на paypal.com, додавши FreeURL. Коли користувачі бачать paypal.com на початку URL-адреси, вони можуть довіряти сайту і підключитися до нього, після чого можуть поділитися своєю конфіденційною інформацією з цим шахрайським сайтом. Це часто використовуваний зловмисниками метод.

Іншими методами, які часто використовуються зловмисниками, є доменний сквотинг (Cybersquatting) і тайпсквотинг (Typosquatting). Доменний сквотинг - це реєстрація, торгівля або використання доменного імені з недобросовісним наміром отримати вигоду з гудвілу торгової марки, що належить іншій особі. Зловмисник може запропонувати продати домен особі або компанії, яка володіє торговою маркою, що міститься в імені, за завищеною ціною або використовувати його в шахрайських цілях, наприклад, для фішингу. Наприклад, назва вашої компанії "abcompany", а ви реєструєтесь як abcompany.com. Тоді фішери можуть зареєструвати abcompany.net, abcompany.org, abcompany.biz і використовувати їх з шахрайською метою. Тайпсквотинг, який також називають викраденням URL-адрес, - це форма кіберсквотингу, яка ґрунтується на помилках, таких як друкарські помилки, допущені користувачами Інтернету при введенні адреси веб-сайту у веб-браузері, або на друкарських помилках, які важко помітити при швидкому читанні. URL-адреси, створені за допомогою Typosquatting, виглядають як довірений домен.

Користувач може випадково ввести неправильну адресу веб-сайту або натиснути на посилання, яке виглядає як довірений домен, і таким чином потрапити на альтернативний веб-сайт, що належить фішеру. Відомим прикладом Typosquatting є goggle.com, надзвичайно небезпечний веб-сайт. Іншим подібним сайтом є yutube.com, який схожий на goggle.com, за винятком того, що він націлений на користувачів Youtube. Аналогічно, www.booking.com було типосквотовано як www.bo0king.com, що перенаправляє користувачів на сайт, який продає подорожі зі знижками. Деякі інші приклади: paywpal.com, microroft.com, appIe.com, appie.com.

Ознаки що використовуються для виявлення фішингових доменів

У науковій літературі та комерційних продуктах існує безліч алгоритмів і широкий спектр даних для виявлення фішингу. Фішингова URL-адреса та відповідна сторінка мають кілька особливостей, за якими їх можна відрізнити від шкідливої URL-адреси. Наприклад, зловмисник може зареєструвати довгий і заплутаний домен, щоб приховати справжнє доменне ім'я (кіберсквотинг, тайпсквотинг). У деяких випадках зловмисники можуть використовувати прямі IP-адреси замість доменного імені або використовувати короткі доменні імена, які не мають відношення до легальних торгових марок і не мають доповнення FreeUrl. Цей тип подій вимагає іншого рішення – використання NLP моделей для аналізу вмісту сторінки, її дій в браузері та подальшої категоризації. На основі ознак URL-адреси, у процесі виявлення в академічних дослідженнях використовуються різні види ознак, які застосовуються в алгоритмах машинного навчання. Ознаки, зібрані з академічних досліджень для виявлення фішингових доменів за допомогою методів машинного навчання, згруповані, як показано нижче.

1. Ознаки на основі URL-адрес

URL-адреса - це перше, що потрібно проаналізувати, щоб вирішити, чи є сайт фішинговим, чи ні. Як ми вже згадували раніше, URL-адреси фішингових доменів мають деякі характерні особливості. Характеристики, пов'язані з цими особливостями, отримуються при обробці URL-адреси. Нижче наведені деякі з цих ознак, що базуються на URL-адресі.

- Кількість цифр в URL-адресі
- Загальна довжина URL-адреси
- Перевірка того, чи URL-адреса не містить опечаток. (google.com → goggle.com)
- Перевірка, чи містить вона легітимну назву бренду (apple-icloud-login.com)
- Кількість субдоменів в URL-адресі

2. Ознаки на основі домену

- Чи є доменне ім'я або його IP-адреса в чорних списках відомих репутаційних сервісів?
- Скільки днів пройшло з моменту реєстрації домену?
- Чи приховано ім'я реєстранта?

3. Ознаки на основі сторінок

Деякі функції на основі сторінок дають нам інформацію про активність користувачів на цільовому сайті. Отримати ці типи даних непросто. Існують деякі платні сервіси для отримання цих типів даних.

- Орієнтовна кількість відвідувань для домену на щоденній, щотижневій або щомісячній основі
- Середня кількість переглядів сторінок за один візит
- Середня тривалість відвідування
- Частка веб-трафіку за країнами
- Кількість посилань з соціальних мереж на даний домен
- Категорія домену
- Схожі сайти і т.д.

4. Ознаки на основі контенту

Для отримання цих функцій потрібне активне сканування цільового домену. Вміст сторінок обробляється для того, щоб можна було визначити, чи використовується цільовий домен для фішингу чи ні. Нижче наведено деяку оброблену інформацію про сторінки.

- Заголовки сторінок
- Мета-теги
- Прихований текст
- Текст у компонентах сторінки
- Динамічний контент в скриптах
- Зображення тощо.

Аналізуючи ці дані, ми можемо зібрати таку інформацію, як:

- Чи заборонити входити на сайт
- Категорія веб-сайту
- Категорія та мета контенту
- Інформація про профіль аудиторії тощо.

Усі описані вище ознаки корисні для виявлення фішингових доменів. У деяких випадках використання деяких з них може бути недоцільним, тому існують певні обмеження для використання цих функцій. Наприклад, може бути нелогічно використовувати деякі функції, такі як Content-Based Features, для розроблюваного механізму швидкого виявлення, який здатний аналізувати кількість доменів від 100 000 до 200 000. Іншим прикладом може бути, якщо ми хочемо проаналізувати нові зареєстровані домени, функції на основі сторінок не дуже корисні. Таким чином, ознаки, які будуть використовуватися механізмом виявлення, залежать від мети механізму виявлення. Які ознаки використовувати в механізмі виявлення, слід вибирати дуже ретельно.

Таблиця 2.3

Різні ресурси та механізми виявлення небезпеки

URL (скорочений)	Тип	Механізм виявлення
br-icloud.com.br	Фішинг	Domain Based Features
9779.info/%E5	Шкідливе ПЗ	Content Based Features
boeuf-soufflenheim.com	Введення в оману	Content Based Features
international.uiowa.edu	—	—
en.wikipedia.org	—	—

halkbankparaf-para.com	Фішинг	Content Based Features
appieid-enable.com	Фішинг	Domain Based Features

2.3. Використання методів селективного аналізу потоку даних

У проблемах класифікації URL-адрес більшість досліджень було зосереджено на вдосконаленні класифікаторів ML (Machine Learning) та DL (Deep Learning). Лише деякі дослідження зосереджувалися на відборі та вилученні комбінації ознак [4, 5, 6]. Більшість з цих досліджень застосовували або тільки лексичні ознаки, або тільки на основі контенту, або тільки на основі мережі(доменні). Вони не застосовували комбінацію всіх цих трьох типів для виконання прогнозування за допомогою моделей ML/DL. Беручи до уваги, що доменні ознаки в більшості випадків можна приховати, комбінування їх відіграє важливу роль у роботі класифікаторів, наша стаття має на меті виокремити різні ознаки URL-адрес, а саме: лексичні, мережеві та контентні ознаки, щоб допомогти класифікатору точно ідентифікувати шкідливі URL-адреси. Крім того, було проведено серію експериментів для вибору методу відбору ознак, який дає найкращі результати з високим рівнем точності. На основі проведених експериментів було обрано комбінацію з трьох методів відбору ознак, а саме: Random Tree, хі-квадрат та кореляція, для побудови остаточного набору даних, які будуть подані на вхід інтелектуальних моделей. Деякі дослідники вивчали інші особливості для аналізу та покращення продуктивності моделей. M.Mamun та ін. [7] використовували дані про трафік кліків, зібрані з Bitly для більш ніж 600 000 коротких URL-адрес, і проаналізували трафік шкідливих і нешкідливих URL-адрес, найкращий результат був досягнутий за допомогою алгоритму Random Tree з точністю 90,81% і значенням F-міри 91,3%. Загалом було зібрано 100 000 повідомлень з платформи Sina Weibo, які були використані для побудови 3 моделей ML: Байєсова мережа, J48 та RF. Середня досягнута точність склала 83,21%, середня F-позитивна частота - 10,03%, а середня F-міра - 86%. Аналогічно, Догра та ін. [8] також розглянули структуру розміщення URL-адрес у соціальних мережах Twitter, проаналізувавши поведінку користувачів, які розміщують URL-

адреси, та користувачів, які натискають на URL-адреси. Використовуючи API твіттера в поєднанні з API Bitly, вони зібрали близько 7 мільйонів твітів, що містять скорочені URL-адреси, створені Bitly, і спробували різні набори функцій, включаючи середню кількість кліків, кількість публікацій, середню кількість підписників, середню кількість друзів, функцію оцінювання та категорію оцінок. Найкраща досягнута точність досягла 86%. З попередніх досліджень можна зробити висновок, що для побудови інтелектуальних моделей можна застосовувати загальну методологія, як показано на рисунку 2.5 (нижче)

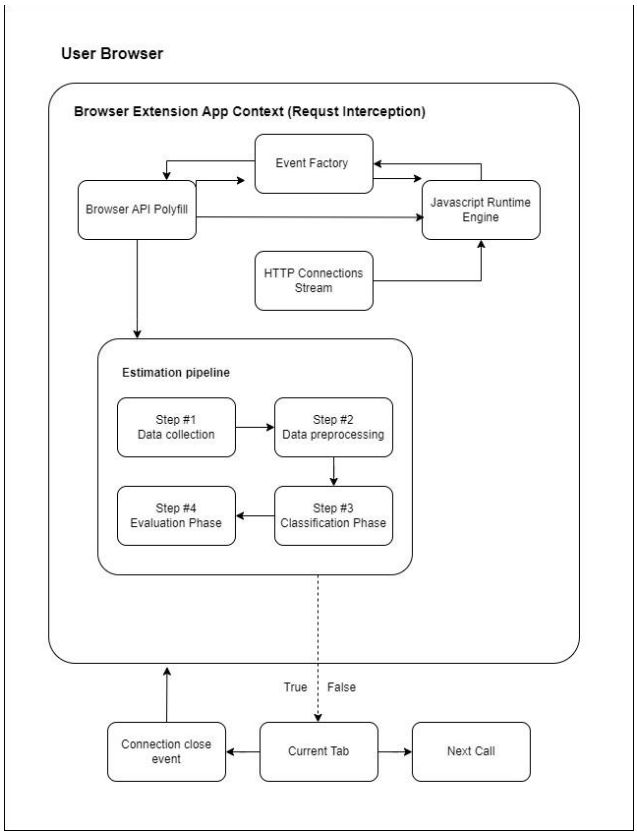

Рис. 2.5. Загальна схема валідації даних запиту в браузері

Набір даних

Для тренування моделі використовувся датасет "Malicious and Benign Webpages Dataset", який є загальнодоступним і був опублікований Сінгхом і Кумаром у 2020 році [9]. Збір інформації проводився за допомогою інструменту Mal Crawler [10], який сканував Інтернет. Мітки для веб-сторінок (чи є вони безпечними чи шкідливими) були перевірені за допомогою Google Safe Browsing

API. Набір даних містив різні атрибути веб-сайтів, такі як URL, IP-адреса, код JavaScript, зашифрований код, географічне розташування, домен верхнього рівня та протокол HTTPS, які були корисні для класифікації веб-сторінок. До складу набору даних також входив необроблений контент сторінок, включаючи код JavaScript, який може бути використаний для видобування інших атрибутів. Набір даних складався з двох частин: одна частина містила тренувальні дані з 2,2 мільйона записів та 11 атрибутів, а інша - тестові дані з 0,564 мільйона записів та 11 атрибутів. У тренувальному наборі було 300 253 зазначено як шкідливі та 1 972 747 - як безпечні. У тестовому наборі 353 872 URL-адреси були визначені як безпечні, а 8 062 - як шкідливі. В Таблиці 1 подано короткий огляд усіх атрибутів цього набору даних.

Таблиця 2.4

Атрибути набору даних з датасету "Malicious and Benign Webpages Dataset"

#	Атрибут	Опис
1	url	URL-адреса веб-сайту
2	add	IP-адреса веб-сайту
3	cert_details	Деталі https з’єднання
4	url_length	Довжина URL-адреси веб-сайту
5	js_length	Довжина JavaScript коду, присутнього на веб-сайті
6	js_obf_length	Довжина прихованого JavaScript коду, присутнього на веб-сайті
7	tld	Домен верхнього рівня веб-сайту
8	who_is	Інформація про домен WHOIS повна чи ні
9	https	Вказує, чи використовує веб-сайт протокол HTTPS
10	content	Сирий вміст веб-сторінки з кодом JavaScript
11	label	Мітка, що вказує на те, чи є веб-сайт шкідливим або безпечним

Однією з ключових фаз у моделях машинного та глибокого навчання є етап попередньої обробки, який підготовлює дані для подальшого застосування методів класифікації. На Рисунку 3 представлені етапи цього процесу, які включають в себе обробку неповної вибірки, вилучення ознак, кодування міток та вибір необхідних характеристик. Детальні пояснення цих кроків далі

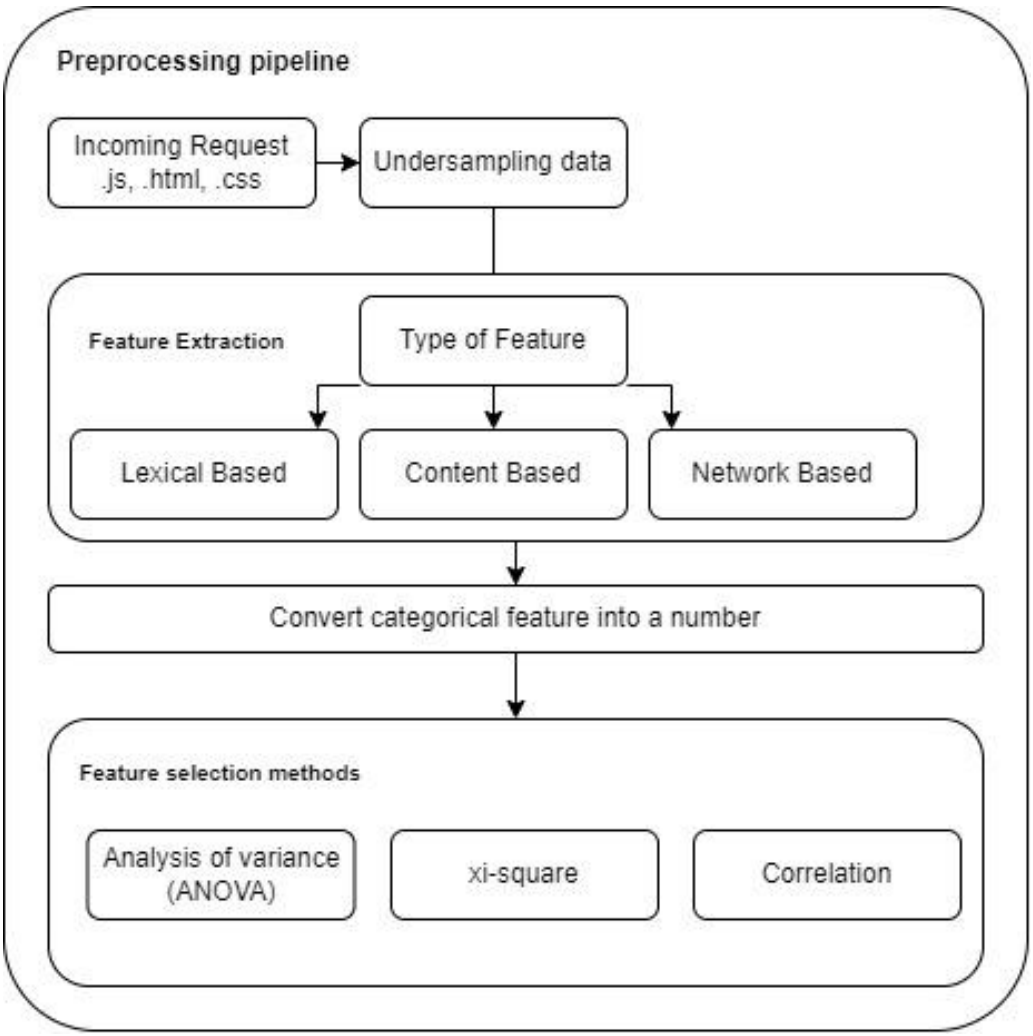

Рис. 2.6. Схема попередньої обробки даних для селективної виборки

Метод рандомізованої неповної вибірки

Метод рандомізованої неповної вибірки використовується для вирішення проблеми дисбалансу класів у наборі даних. Це включає у себе зменшення кількості зразків у класі більшості з метою досягнення балансу з класом меншості в наборі даних. Незбалансований розподіл класів матиме один або кілька класів з малою кількістю прикладів (класи меншості) та один або кілька класів з великою

кількістю прикладів (класи більшості). Найкраще це зрозуміти в контексті бінарної (двокласової) задачі класифікації, де клас 0 - це клас більшості, а клас 1 - клас меншості. Методи недостатньої вибірки видаляють з навчального набору даних приклади, які належать до класу більшості, щоб краще збалансувати розподіл класів, наприклад, зменшити асиметрію з 1:100 до 1:10, 1:2 або навіть 1:1. Це відрізняється від надмірної вибірки, яка передбачає додавання прикладів до класу меншості, щоб зменшити асиметрію в розподілі класів. Методи рандомізованої неповної вибірки можна використовувати безпосередньо на навчальному наборі даних, який потім, у свою чергу, можна використовувати для підгонки моделі машинного навчання. Зазвичай методи заниженої вибірки використовуються разом з методами надмірної вибірки для класу меншості, і така комбінація часто дає кращі результати, ніж використання надмірної або заниженої вибірки окремо на навчальному наборі даних.

Найпростіший метод неповної вибірки полягає у випадковому виборі прикладів з класу більшості і видаленні їх з навчального набору даних. Це називається випадковою неповною вибіркою. Незважаючи на простоту та ефективність, обмеження цього методу полягає в тому, що приклади видаляються без урахування того, наскільки вони можуть бути корисними або важливими для визначення межі рішення між класами. Це означає, що існує ймовірність того, що корисна інформація буде видалена. Розширенням цього підходу є більш розбірливий підхід до прикладів з класу більшості, які вилучаються. Зазвичай для цього використовують евристики або моделі навчання, які намагаються визначити надлишкові приклади для видалення або корисні приклади для невидалення. Існує багато методів неповної вибірки, які використовують ці типи евристик. Синтетичний набір даних бінарної класифікації за допомогою функції make_classification() з бібліотеки scikit-learn. Наприклад, ми можемо створити 10 000 прикладів з двома вхідними змінними і розподілом 1:100. Після цього можна створити діаграму розсіювання набору даних за допомогою функції scatter() Matplotlib, щоб зрозуміти просторовий зв'язок прикладів у кожному класі та їхній дисбаланс. Далі створюється діаграма розсіювання, яка показує всі приклади в

наборі даних. Ми бачимо велику кількість прикладів для класу 0 (синій колір) і невелику кількість прикладів для класу 1 (помаранчевий). Ми також бачимо, що класи перетинаються з деякими прикладами з класу 1 в тій частині простору ознак, яка належить до іншого класу, рисунок 2.7

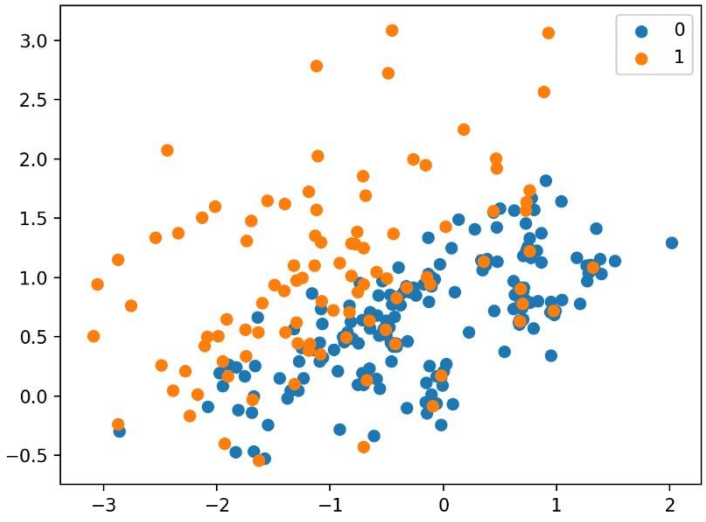

Рис. 2.7. Діаграма розсіювання набору даних

Комбінації методів збереження та видалення

У цьому дослідженні навчальний набір даних мав незбалансовані класи, оскільки більшість записів вибірки були доброякісними (1 142 747 з 1,2 мільйона), тоді як лише 57 253 були визначені як зловмисні. Застосування рандомізованої неповної вибірки призвело до того, що у навчальному наборі даних з'явилися рівні кількості шкідливих і безпечних записів (27 273 кожен), зробивши загальну кількість записів у навчальному наборі даних 55 506. У випадку тестового файлу даних із 300 тисяч записів вихідного файлу випадковим чином було вибрано 15 000 записів. Отже, загальна кількість записів у навчальному наборі даних становила 55 506, а в тестовому - 15 000.

Обробка ознак

Після уважного аналізу набору даних, було виділено найбільш значущі ознаки для подальшого використання у моделях. Покладаючись не тільки на наявні

мережеві ознаки, але й розширюючи їх, було враховано лексичні характеристики на основі URL-адрес та ознаки, які визначаються вмістом веб-сторінок. Для ефективного вилучення лексичних ознак, які відображають текстові особливості URL, такі як спеціальні символи, розширення шляху, довжина шляху, хост тощо. Використано кілька бібліотек на мові програмування Python та платформі Scikit Learn. Ці інструменти дозволили вилучити та врахувати ключові ознаки для подальшого використання у аналізі та моделюванні наступних ознак:

1. Аналіз довжини шляху в URL-адресі важливий для визначення точного місцезнаходження ресурсу. Врахування розміру шляху, кількості крапок та косих рисок дозволяє виявити потенційно підозрілі або складні URL-адреси.
2. Перевірка SSL сертифікату, наявність TA (Trusted Authorities) є показником прозорості сайту, так як такий сертифікат не можна згенерувати самому, для цього потрібна перевірка 3-ю стороною та перелік документів, що унеможливлює анонімність власника домену.
3. Перевірка наявності спеціальних символів в URL-адресі, таких як крапки, &, /, -, =, , $, #, *, !, @, і ?, є ключовим етапом для виявлення потенційно шкідливих дій. Зловмисники можуть використовувати ці символи для обхіду валідації та виклику атак на закодовані URL.
4. Оцінка довжини хоста та наявності цифр в імені хоста стає новим напрямком для виявлення аномалій. Ця інформація може свідчити про особливості хоста, які вказують на можливі ризики або неправомірні дії.Довжина URL-адреси: представляє загальну довжину всієї URL-адреси.
5. Довжина всього URL-адреси є загальною мірою її складності та може вказати на певний ступінь потенційної загрози або неправомірності.
6. Розгляд кількості цифр і букв в URL-адресі через властивості count_digits та count_letters допомагає виявити числові та текстові характеристики, що може бути важливим для подальшого аналізу та класифікації.

Для контент-орієнтованих ознак, які в основному стосуються функцій JavaScript та HTML-тегів, що можуть бути використані зловмисниками для експлуатації веб-сайтів, виокремлено наступні ознаки:

1. Наявність підозрілої JavaScript-функції:
2. Функція Eval: ця функція використовується зловмисниками для генерації шкідливого коду на шкідливих веб-сайтах під час виконання, щоб перешкодити виявленню [30]. У наборі даних ми шукали наявність цієї функції eval() у коді JavaScript у функціях вмісту та зберігали її кількість у новій функції під назвою count_eval як числовий атрибут.
3. Наявність iFrame на сайті: HTML-тег iFrame використовувався в ряді атак для завантаження шкідливих JavaScript-експлойтів
4. Функція escape: хакери зазвичай кодують шкідливий код за допомогою цієї функції, щоб потім декодувати його та ініціювати атаки. У наборі даних перевіряється наявність цієї функції escape() у JavaScript-коді в елементах вмісту та зберігали її кількість у новому елементі під назвою count_escape як числовий атрибут.
5. Функція unescape: після кодування шкідливого коду за допомогою escape() зловмисники зазвичай розшифровують його за допомогою функції unescape() і зрештою ініціюють атаку. Тому кількість функцій escape та unescape є хорошим індикатором шкідливої активності [31]. У наборі даних ми шукали наявність цієї функції unescape() у JavaScript-коді у функціях контенту та зберігали її кількість у новій функції під назвою count_unescape як числовий атрибут.
6. Функція find: щоб розшифрувати шкідливий код під час виконання, зловмисники інтегрують find() разом з функціями eval() та unescape() [31]. Тому ми шукали наявність цієї функції find() в JavaScript-коді у функціях контенту та зберігали її кількість у новій функції count_find як числовий атрибут.
7. Функції Exec, Search та Link: ми шукали наявність цих інших підозрілих функцій, що використовуються зловмисниками в JavaScript-коді в

елементах контенту, і зберігали їх кількість у нових елементах як числові атрибути.

8. Загальна кількість підозрілих функцій: оскільки всі вищезгадані функції використовуються зловмисниками для міжсайтового скриптингу та розповсюдження шкідливого програмного забезпечення [12], було підсумувано кількість всіх цих функцій, які присутні в JavaScript-коді в елементах контенту, і зберегли їх підсумкову кількість в новому елементі під назвою count_all_functions як числовий атрибут.
9. Наявність функції Windows.open: функції Windows.open зазвичай використовуються для рекламних спливаючих вікон і можуть бути використані для ін'єкційних атак на шкідливі сайти. Наявність цього спливаючого вікна в контенті зберігається як булеве значення.

Відбір ознак

Відбір ознак – це стратегія, яка полягає у виборі ключових стовпців або характеристик з набору даних. Використання цього методу дозволяє підвищити ефективність та результативність алгоритмів машинного та глибокого навчання, адже вони оптимізуються для роботи з меншою кількістю ознак, що сприяє зменшенню вимог до пам'яті та скороченню часу обробки. Важливо відзначити, що цей підхід також спрощує високу розмірність даних, адаптуючи їх до більш зручного та оптимального формату. Зокрема, виключення непотрібних ознак може уникнути неправильних або марних впливів на алгоритми, покращуючи загальну їх продуктивність. У цій роботі розглянуто 3 основних методи які використовуються для відбіру ознак:

1. ANOVA - це статистичний метод, який використовується для оцінки того, чи існують статистично значущі відмінності між середніми значеннями трьох або більше незалежних груп. Нульова гіпотеза припускає, що середні значення в групах рівні, тоді як альтернативна гіпотеза припускає, що принаймні в одній групі середні значення відрізняються.

$$H_0: \mu_1 = \mu_1 = \cdots = \mu_k$$

H_0: як мінімум 1 μ_k відрізняється

Загальне середнє значення(Y) обчислюється як:

$$A = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i} Y_{ij}}{N},$$

де $n_i -$ Номер спостереження у групі,

$i -$ Індекс, що представляє кожну групу. Він варіюється від 1 до k

$k -$ загальна кількість груп,

$j -$ індекс кожного спостереження для деякої групи

$Y_{ij} -$ спостереження в деякій i^{th} групі та j^{th} позиції

Сума квадратів групи(SST) обчислюється за формулою:

$$SST = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(Y_{ij} - A)^2$$

Міжгрупова сума квадратів(SSB):

$$SSB = \sum_{i=1}^{k} n_i(A_{ij} - A)^2$$

Внутрішньо-групова сума квадратів(SSB):

$$SSW = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(Y_{ij} - A)^2$$

Тоді діапазон вибору значень:

$$df_{Total} = N - 1$$
$$df_{Between} = N - 1$$
$$df_{Within} = N - k$$

Середні квадрати між та всередині групи, відповідно:

$$MSB = \frac{SSB}{df_{Between}}$$

$$MSW = \frac{SSW}{df_{Within}}$$

Застосувавши цей метод, було знайдено ознаки які мають незначну ефективність ($MSW_i < \frac{df_{Between}}{df_{Within}}$) та не вплинуть на результат розподілення (Рис 2.8).

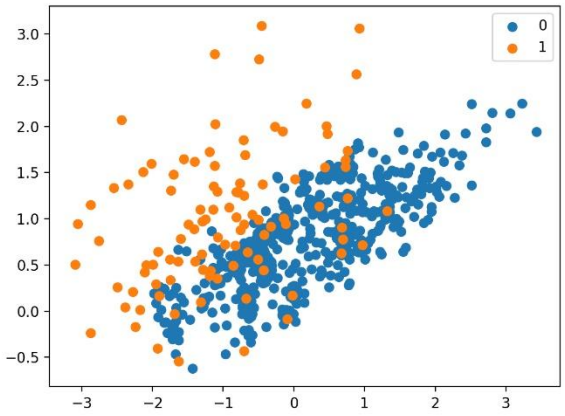

Рис. 2.8. Діаграма розсіювання набору даних з використанням ANOVA функції

2. хі-квадрат - Це статистична функція використовується дослідниками для визначення різниці між тим, що фактично спостерігається, та тим, що очікується. Іншими словами, вона служить для перевірки незалежності двох змінних [33]. У цьому контексті "O" позначає фактично спостережуване значення, "E" відображає очікуване значення, а хі-квадрат вимірює, наскільки ці дві змінні відрізняються одна від одної.

$$x^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Ця формула дозволяє визначити значення хі-квадрат, використовується для оцінки того, наскільки великою є залежність між двома змінними та чи можна вважати їх пов'язаними. Чим вище отримане значення хі-квадрат, тим сильніше вказується на залежність між ознаками. У контексті аналізу, це означає, що дану ознаку можна

вважати значущою та корисною для включення в модель. Такий результат свідчить про те, що взаємозв'язок між ознаками варто враховувати при подальшому моделюванні чи аналізі даних.

3. Кореляція - це статистичний показник, що використовується для вимірювання ступеня лінійного взаємозв'язку між двома змінними. Вона застосовується для виявлення зв'язку між різними ознаками та ознакою цільового класу. Коли ознаки проявляють лінійну незалежність (вони некорельовані), значення коефіцієнта кореляції дорівнює 0. У випадку лінійної залежності (коли ознаки корельовані), значення коефіцієнта кореляції може бути або +1 (позитивна кореляція) або -1 (негативна кореляція). Припустимо, що маємо дві вибірки, X та Y, кожна з яких складається з m спостережень (x1, x2, ..., xm) та (y1, y2, ..., ym) відповідно [34].

Коефіцієнт кореляції Пірсона обчислюється за наступною формулою:

$$r = \frac{(N * \sum x_i y_i - \sum x_i y_i)}{\left(N * x_i^2 - \left(\sum x_i\right)^2\right) * \left(N * y_i^2 - \left(\sum y_i\right)^2\right)}$$

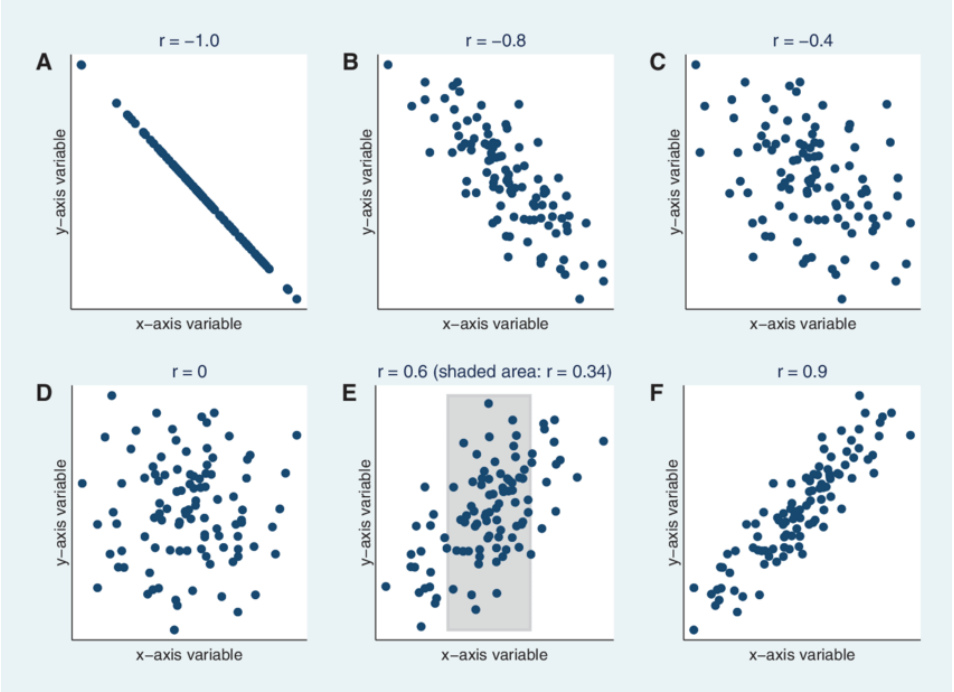

Рис. 2.9. Кореляція ознак з різними значеннями коефіцієнту Пірсона

2.4. Розробка вимог та застосунку для протидії кібератакам на браузер користувача

Браузер є одним із найважливіших компонентів сучасної інформаційної інфраструктури. Він використовується для доступу до веб-сайтів, веб-додатків та інших онлайн-ресурсів. Тому браузер є привабливою ціллю для кіберзлочинців. Кібератаки на браузери можуть призвести до різних наслідків, включаючи:

- Вкрасту особистих даних, таких як імена користувачів, паролі та номери кредитних карток.
- Встановлення шкідливого програмного забезпечення, такого як троянські коні, віруси та шпигунське програмне забезпечення.
- Втручання в роботу браузера, наприклад, шляхом перенаправлення користувачів на підроблені веб-сайти або блокування доступу до законних веб-сайтів.

Застосунок для протидії кібератакам на браузери повинен відповідати таким вимогам:

- Ефективність: застосунок повинен бути здатним виявляти та блокувати широкий спектр кіберзагроз.
- Неінвазивність: застосунок не повинен надмірно впливати на продуктивність браузера або на досвід користувача.
- Простота використання: застосунок повинен бути простим у встановленні та використанні.

Застосунок для протидії кібератакам на браузери складається з таких основних компонентів:

Фільтр шкідливого програмного забезпечення використовується для виявлення та блокування шкідливого програмного забезпечення, яке може бути завантажено в браузер. Шкідливе програмне забезпечення може бути завантажено з веб-сайтів, електронних листів або інших джерел. Фільтр шкідливого програмного забезпечення використовує різні методи для виявлення шкідливого програмного забезпечення, включаючи сигнатурне виявлення, аналіз поведінки та аналіз файлів.

Фільтр фішингу використовується для виявлення та блокування веб-сайтів, які використовують фішингові атаки для крадіжки особистих даних. Фішингові атаки використовують підроблені веб-сайти, які виглядають як справжні веб-сайти, щоб обманом заманити користувачів ввести свої особисті дані, такі як імена користувачів, паролі та номери кредитних карток. Фільтр фішингу використовує різні методи для виявлення фішингових атак, включаючи порівняння веб-сайтів із базою даних відомих фішингових веб-сайтів, аналіз поведінки користувачів та аналіз тексту веб-сайтів.

Сервіс криптографії шифрує весь вихідний та вхідний трафік, включаючи токени доступу, щоб захистити важливу інформацію від потенційних зловмисників або перехоплювачів. Криптографічні методи, використовувані плагіном, допомагають у запобіганні атакам, таким як атаки "Man-in-the-Middle," забезпечуючи захист від неправомірного перехоплення трафіку чи зміни передаваних даних.

Фільтр XSS використовується для виявлення та блокування сценаріїв міжсайтової підміни (XSS), які можуть використовуватися для злому браузера. XSS-атаки використовують вразливості в браузері, щоб вставити шкідливий код на веб-сайт, який потім може бути виконаний, коли користувач відвідує веб-сайт. Фільтр XSS використовує різні методи для виявлення XSS-атак, включаючи аналіз тексту веб-сайтів та аналіз поведінки користувачів.

Таблиця 2.5

Функціональні вимогу для застосунку

Функціональні вимоги	Опис
Функція відкриття веб-сторінок	Застосунок повинен дозволяти користувачам відкривати веб-сторінки за допомогою адреси URL або пошуку.
Функція перегляду веб-сторінок	Застосунок повинен дозволяти користувачам переглядати веб-сторінки, включаючи прокручування, збільшення та зменшення масштабу.
Функція керування вкладками	Застосунок повинен дозволяти користувачам керувати вкладками, включаючи створення,

	закриття та перемикання між вкладками.
Функція пошуку	Застосунок повинен дозволяти користувачам виконувати пошук в Інтернеті.
Функція завантаження файлів	Застосунок повинен дозволяти користувачам завантажувати файли з Інтернету.
Функція зберігання паролів	Застосунок повинен дозволяти користувачам зберігати паролі для веб-сайтів.

Закінчення таблиці 2.5.

Функція розширень	Застосунок повинен дозволяти користувачам встановлювати розширення, які додають нові функції або можливості.

Таблиця 2.6

Нефункціональні вимогу для застосунку

Нефункціональні вимоги	Опис
Вимога до безпеки	Застосунок повинен бути безпечним для використання, включаючи захист від шкідливого програмного забезпечення та фішингових атак.
Вимога до продуктивності	Застосунок повинен працювати швидко і ефективно, навіть при відкритті великої кількості вкладок.
Вимога до зручності використання	Застосунок повинен бути простим у використанні та зрозумілим для користувачів.
Вимога до доступності	Застосунок повинен бути доступним для користувачів з обмеженими можливостями.

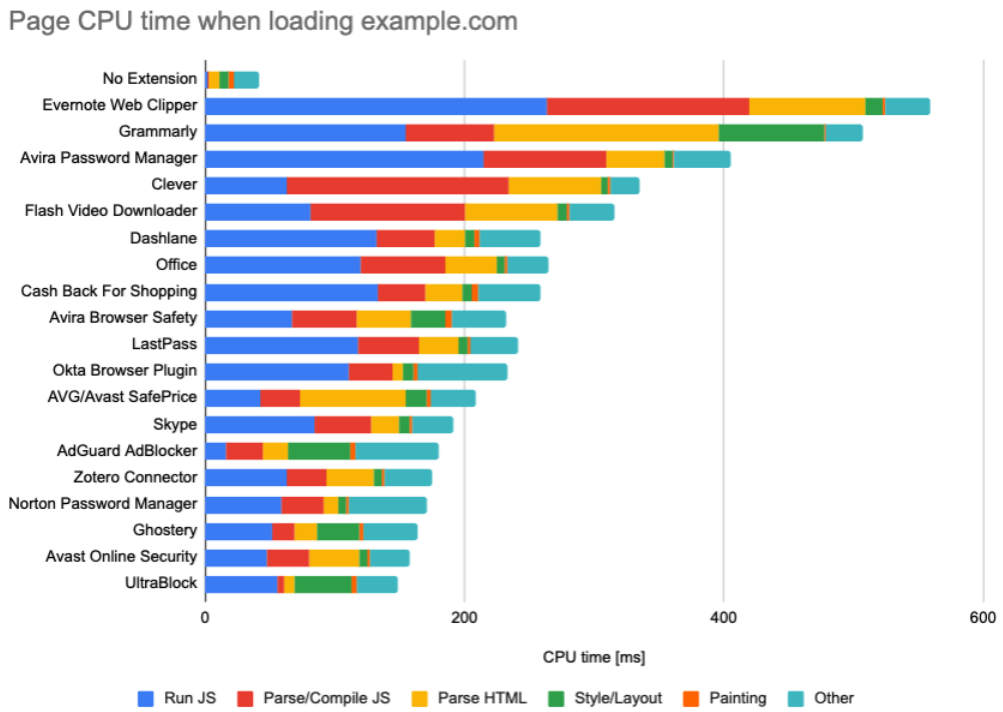

Рис. 2.10. Порівняння навантаження на процесор з існуючими рішеннями та без них

Базовий алгоритм дій

1. Коли користувач відвідує веб-сайт, плагін перехоплює запит до веб-сайту.
2. Плагін визначає домен веб-сайту.
3. Плагін перевіряє домен у базі даних відомих фішингових та шкідливих доменів.
4. Якщо домен не є відомим, плагін продовжує аналіз.
5. Плагін аналізує текст запиту.
6. Якщо в тексті запиту виявлені ознаки XSS-атаки, плагін знешкоджує атаку.
7. Якщо в тексті запиту не виявлені ознаки XSS-атаки, плагін продовжує аналіз.
8. Плагін аналізує код, що надходить від веб-сайту. Якщо в коді виявлені ознаки шкідливого програмного забезпечення, плагін блокує виконання коду.
9. Якщо в коді не виявлені ознаки шкідливого програмного забезпечення, плагін продовжує аналіз.
10. Плагін перенаправляє запит до веб-сайту.

Висновки до розділу

В даному розділі були розглянуті та досліджені ключові аспекти комплексного підходу до протидії кібератакам на браузер користувача. Було визначено, що ефективний захист включає в себе використання комплексу заходів, таких як поглиблений моніторинг мережевої активності, використання алгоритмів на основі машинного навчання та додаткове шифрування чутливої інформації Розпізнавання потенційно шахрайських веб-сайтів є важливої частиної цієї роботи. Аналіз URL адрес та мережево-доменних характеристик виявився ефективним методом визначення легітимності веб-ресурсів. Посилання на підозрілі ресурси може слугувати сигналом можливих загроз та допомагати вчасно реагувати на них. Використання методів селективного аналізу потоку даних дозволяє виявляти аномалії та підозрілі патерни в мережевому трафіку, сприяючи вчасному виявленню та блокуванню кіберзагроз. Розробки вимог та застосунків для протидії кібератакам на браузер користувача включає в себе розробку політик безпеки, які визначають стандарти безпеки та вимоги до програмного забезпечення, що використовується.

Узагальнюючи, комплексний підхід до захисту браузера користувача з урахуванням комбінованої методики моніторингу даних сприятиме створенню надійної та ефективної системи кібербезпеки для користувачів.

РОЗДІЛ 3
СТРУКТУРА СЕРВІСІВ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1. Середовище виконання та зона доступу застосунку

Середа виконання та зона доступу застосунку Chrome – це два важливі поняття, які необхідно розуміти при розробці розширень Chrome. Середа виконання – це середовище, в якому працює розширення Chrome. Воно включає в себе ядро Chrome, а також інші бібліотеки та компоненти, які необхідні для роботи розширення. Зона доступу – це набір ресурсів, до яких має доступ розширення. Ресурси можуть включати в себе:

- Дані користувача, такі як закладки, історія та файли cookie.
- Дані про веб-сторінки, такі як вміст сторінки та заголовки HTTP.
- Дані про сам Chrome, такі як версія браузера та список завантажених розширень.

Розробники розширень повинні уважно розглядати середовище виконання та зону доступу при розробці своїх розширень. Це допоможе забезпечити, щоб розширення працювало належним чином та не порушувало конфіденційність користувачів. Середа виконання Середа виконання Chrome включає в себе такі основні компоненти: Ядро Chrome – це основне ядро браузера, яке відповідає за такі функції, як відображення веб-сторінок, обробка JavaScript та управління процесами. WebExtensions API – це набір API, який надає розширенням доступ до функцій Chrome. Web Platform APIs – це набір API, який надає розширенням доступ до функцій веб-платформи, таких як DOM та CSS.

При роботі з конфіденційною інформацією потрібно дотримуватись 3 рекомендацій політики GDPR

- Розробники розширень повинні пам’ятати, що їхні розширення можуть бути використані для збирання даних про користувачів. Ці дані можуть використовуватися для цілей маркетингу, таргетингу або навіть для злому.

- Розробники розширень повинні бути прозорими щодо того, які дані вони збирають і як вони їх використовують. Користувачі повинні мати можливість надавати свою згоду на збирання даних.
- Розробники розширень повинні надавати користувачам можливість управляти збором даних. Користувачі повинні мати можливість видаляти дані, які вони надали, або заборонити розширенню збирати дані в майбутньому.
- Розробники повинні використовувати шифрування для забезпечення конфіденційності зібраних даних. Важливо гарантувати, що дані користувачів залишаються в безпеці від несанкціонованого доступу.
- Розробники повинні використовувати шифрування для забезпечення конфіденційності зібраних даних. Важливо гарантувати, що дані користувачів залишаються в безпеці від несанкціонованого доступу.
- Утримання від запиту зайвих дозволів, які не пов'язані з функціональністю розширення зменшить ризик зловживання та неправомірного збору даних.

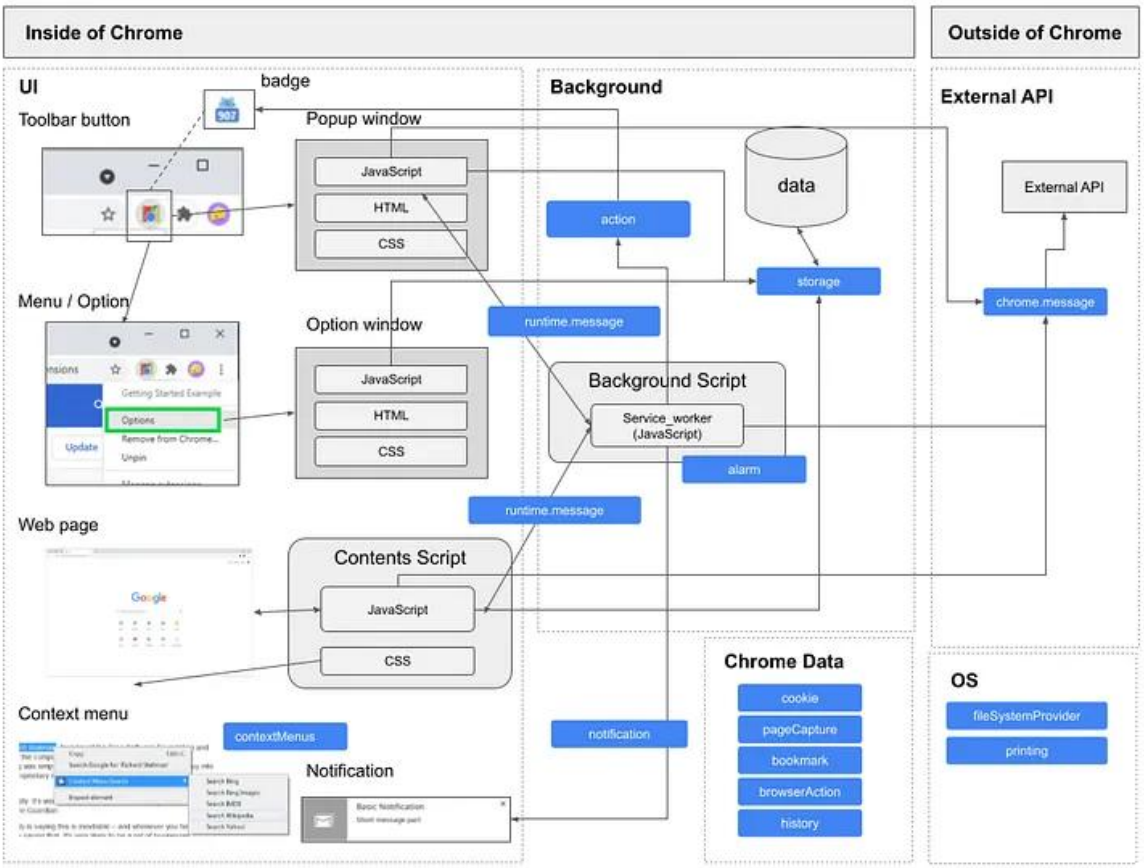

Рис. 3.1. Схема взаємодії компонентів в браузері Chrome

Таблиця 3.1

Зони дозволу застосунку в контексті браузера

Назва дозволу	Опис	Функціональність
browsingData	Дозволяє розширенню отримувати доступ до історії веб-перегляду, закладок та файлів cookie користувача.	Приватна інформація
bookmarks	Дозволяє розширенню отримувати доступ до закладок користувача та керувати ними.	Приватна інформація
downloads	Дозволяє розширенню отримувати доступ до інформації про завантаження користувача.	Приватна інформація
history	Дозволяє розширенню отримувати доступ до історії веб-перегляду користувача.	Приватна інформація
sessions	Дозволяє розширенню отримувати доступ до інформації про сеанси веб-перегляду користувача.	Приватна інформація
tabs	Дозволяє розширенню отримувати доступ до інформації про вкладки та керувати ними.	Базова функціональність
webNavigation	Дозволяє розширенню отримувати доступ до інформації про навігацію по веб-сторінках.	Базова функціональність
activeTab	Дозволяє розширенню звертатися до поточної активної вкладки.	Вміст веб-сторінок

Продовження таблиці 3.1.

webRequest	Дозволяє розширенню перехоплювати та модифікувати мережеві запити та відповіді.	Важливе, потенційно небезпечне
webRequestBlocking	Дозволяє розширенню блокувати мережеві запити до їхнього відправлення.	Важливе, потенційно руйнівне
webRequestHeaders	Дозволяє розширенню модифікувати заголовки HTTP запитів.	Вміст веб-сторінок
declarativeContent	Дозволяє розширенню визначати правила для вставки скриптів та стилів у веб-сторінки.	Вміст веб-сторінок
contextMenus	Дозволяє розширенню створювати та керувати контекстними менюми в веб-сторінках.	Взаємодія з користувачем
cookies	Дозволяє розширенню отримувати доступ до файлів cookie браузера.	Приватна інформація
storage	Дозволяє розширенню зберігати власні дані.	Базова функціональність
chrome.storage.local	Дозволяє розширенню отримувати доступ до локальних даних зберігання (постійних).	Приватна інформація
chrome.storage.sync	Дозволяє розширенню отримувати доступ до синхронізованих даних зберігання (доступних на всіх пристроях).	Приватна інформація
chrome.storage.managed	Дозволяє розширенню отримувати доступ до керованих даних зберігання (спрямованих адміністратором підприємства).	Корпоративні

Закінчення таблиці 3.1.

identity.email	Дозволяє розширенню отримувати доступ до основної адреси електронної пошти користувача.	Приватна інформація
identity.profile	Дозволяє розширенню отримувати доступ до інформації про профіль користувача.	Приватна інформація
permissions	Дозволяє розширенню запитувати додаткові дозволи у користувача.	Потужне, вимагає обережного обґрунтування
privacy	Дозволяє розширенню отримувати доступ до налаштувань конфіден	

3.2. Архітектура застосунку

Середа виконання застосунку дає змогу робити різні операції з усім АПІ браузера: перехоплювати, зупиняти та переадресовувати запити, що є критично важливим для правильної роботи застосунку. Застосунок складається з різних сервісів що спілкуються між собою:

Background Script – У контексті веб-розробки та браузерних розширень фоновий скрипт – js файл, який виконується у фоновому режимі веб-браузера. Розширення для браузерів часто складаються з декількох компонентів, а фоновий скрипт виконує завдання, які безпосередньо не пов'язані з користувацьким інтерфейсом, але є важливими для функціональності розширення. Використовується для керування подіями, збереження даних або виконувати завдання, які не потребують взаємодії з користувачем.

Content Script – це тип скриптів, який використовується в розширеннях браузерів для взаємодії з вмістом веб-сторінок, до яких застосовується розширення. Розширення для браузерів, наприклад, для Google Chrome або Mozilla Firefox, часто складаються з декількох компонентів, і скрипт вмісту є одним з таких компонентів.

Base Rule Protection – набір індивідуальних інструкцій для кожного з’єднання, створені для регулювання можливостей кожної сторонки та її скриптів

Shared Database – централізований механізм зберігання, який дозволяє різним компонентам розширення, таким як фонові скрипти, скрипти вмісту, спливаючі скрипти та сторінки налаштувань, обмінюватися даними та отримувати до них доступ. Розширенням Chrome часто потрібно постійно зберігати та отримувати дані, а спільна база даних або рішення для зберігання полегшує комунікацію та обмін даними між різними частинами розширення.

Domain Feature Detector – механізм у розширенні, який ідентифікує або виявляє особливості, характерні для певного домену (веб-сайту), де активне розширення.

Content Feature Detector – механізм у розширенні, який ідентифікує або виявляє особливості, характерні контенту веб-сайту. Це може бути фішинг або шкідлива реклама

Signature Database – База даних відбитків файлів, строкових паттернів,

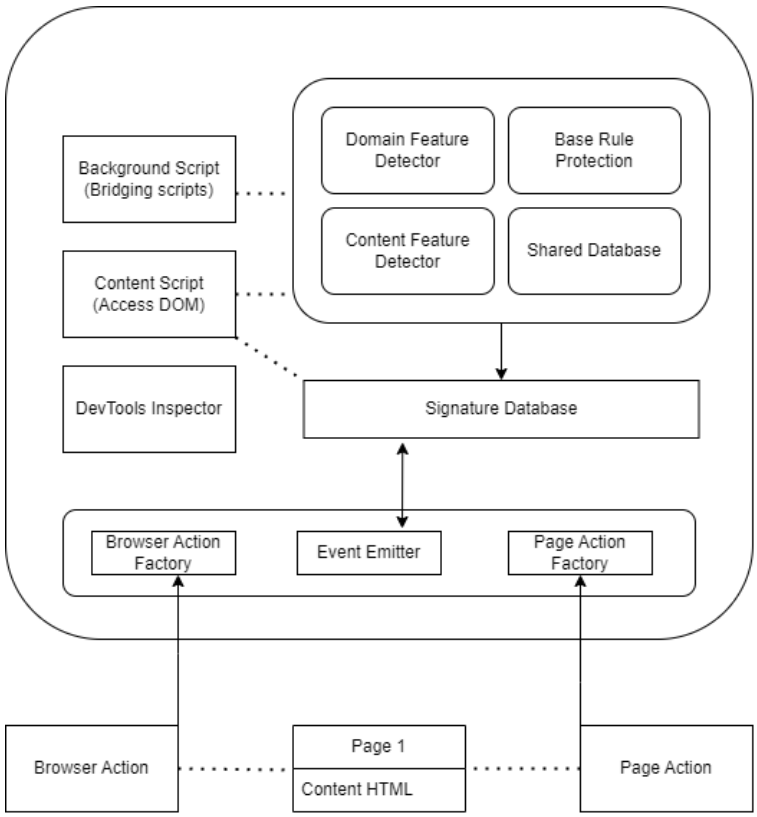

Рис. 3.2. Схема взаємодії веб-сторінки з сервісами застосунку

3.3. Код програми

Обираючи фреймворк для реалізації взаємодії між сервісами, було взято за основу фреймворк Nest.js. Він спрямований на обслуговування серверів та побудову масштабованих додатків. Nest.js базується на концепціях, які позичені з Angular і надає високорівневі абстракції для розробки серверної частини додатків.

Однією з ключових особливостей Nest.js є його модульна структура, яка дозволяє легко організовувати код та використовувати компоненти для певних функціональних частин додатка. Це особливо корисно при розробці мікросервісної архітектури, де кожен сервіс може бути реалізований як окремий модуль.

Додатково забезпечує підтримку різних протоколів для обміну даними, таких як HTTP, WebSockets і багатьох інших. Це надає гнучкість у виборі оптимального методу взаємодії між сервісами в залежності від конкретних потреб проекту.

Крім того, Nest.js дозволяє легко впроваджувати масштабованість, що робить його відмінним вибором для проектів, де важливо підтримувати високий рівень продуктивності та обробки великої кількості запитів.

Таким чином, використання фреймворку цього фреймворку для розробки сервісної архітектури вказує на вибір потужного інструменту, який сприяє швидкому розгортанню та масштабуванню серверних додатків у великих та складних проектах.

Лістинг 3.1

Модуль комунікації між сервісами

```
1:  // Файл rmq-connections.ts
2:  import { Logger } from '@nestjs/common/services/logger.service';
3:  import { loadPackage } from '@nestjs/common/utils/load-package.util';
4:  import { randomStringGenerator } from '@nestjs/common/utils/random-string-
generator.util';
5:  import { isFunction } from '@nestjs/common/utils/shared.utils';
6:  import { EventEmitter } from 'events';
7:  import {
8:    EmptyError,
9:    firstValueFrom,
10:   fromEvent,
11:   merge,
12:   Observable,
13:   ReplaySubject,
```

```
14: } from 'rxjs';
15: import { first, map, retryWhen, scan, skip, switchMap } from
'rxjs/operators';
16: import {
17:   CONNECT_EVENT,
18:   CONNECT_FAILED_EVENT,
19:   DISCONNECT_EVENT,
20:   DISCONNECTED_RMQ_MESSAGE,
21:   ERROR_EVENT,
22:   RQM_DEFAULT_IS_GLOBAL_PREFETCH_COUNT,
23:   RQM_DEFAULT_NO_ASSERT,
24:   RQM_DEFAULT_NOACK,
25:   RQM_DEFAULT_PERSISTENT,
26:   RQM_DEFAULT_PREFETCH_COUNT,
27:   RQM_DEFAULT_QUEUE,
28:   RQM_DEFAULT_QUEUE_OPTIONS,
29:   RQM_DEFAULT_URL,
30: } from '../constants';
31: import { RmqUrl } from '../external/rmq-url.interface';
32: import { ReadPacket, RmqOptions, WritePacket } from '../interfaces';
33: import { RmqRecord } from '../record-builders';
34: import { RmqRecordSerializer } from '../serializers/rmq-record.serializer';
35: import { ClientProxy } from './client-proxy';
36:
37: // import type {
38: //   AmqpConnectionManager,
39: //   ChannelWrapper,
40: // } from 'amqp-connection-manager';
41: // import type { Channel, ConsumeMessage } from 'amqplib';
42:
43: type Channel = any;
44: type ChannelWrapper = any;
45: type ConsumeMessage = any;
46: type AmqpConnectionManager = any;
47:
48: let rqmPackage: any = {};
49:
50: const REPLY_QUEUE = 'amq.rabbitmq.reply-to';
51:
52: /**
53:  * @publicApi
54:  */
55: export class ClientRMQ extends ClientProxy {
56:   protected readonly logger = new Logger(ClientProxy.name);
57:   protected connection$: ReplaySubject<any>;
58:   protected connection: Promise<any>;
59:   protected client: AmqpConnectionManager = null;
60:   protected channel: ChannelWrapper = null;
61:   protected urls: string[] | RmqUrl[];
62:   protected queue: string;
```

```
63:   protected queueOptions: Record<string, any>;
64:   protected responseEmitter: EventEmitter;
65:   protected replyQueue: string;
66:   protected persistent: boolean;
67:   protected noAssert: boolean;
68:
69:   constructor(protected readonly options: RmqOptions['options']) {
70:     super();
71:     this.urls = this.getOptionsProp(this.options, 'urls') ||
[RQM_DEFAULT_URL];
72:     this.queue =
73:       this.getOptionsProp(this.options, 'queue') || RQM_DEFAULT_QUEUE;
74:     this.queueOptions =
75:       this.getOptionsProp(this.options, 'queueOptions') ||
76:       RQM_DEFAULT_QUEUE_OPTIONS;
77:     this.replyQueue =
78:       this.getOptionsProp(this.options, 'replyQueue') || REPLY_QUEUE;
79:     this.persistent =
80:       this.getOptionsProp(this.options, 'persistent') ||
RQM_DEFAULT_PERSISTENT;
81:     this.noAssert =
82:       this.getOptionsProp(this.options, 'noAssert') ||
RQM_DEFAULT_NO_ASSERT;
83:
84:     loadPackage('amqplib', ClientRMQ.name, () => require('amqplib'));
85:     rqmPackage = loadPackage('amqp-connection-manager', ClientRMQ.name, ()
=>
86:       require('amqp-connection-manager'),
87:     );
88:
89:     this.initializeSerializer(options);
90:     this.initializeDeserializer(options);
91:   }
92:
93:   public close(): void {
94:     this.channel && this.channel.close();
95:     this.client && this.client.close();
96:     this.channel = null;
97:     this.client = null;
98:   }
99:
100:      public connect(): Promise<any> {
101:        if (this.client) {
102:          return this.convertConnectionToPromise();
103:        }
104:        this.client = this.createClient();
105:        this.handleError(this.client);
106:        this.handleDisconnectError(this.client);
107:
108:        this.responseEmitter = new EventEmitter();
```

```
109:         this.responseEmitter.setMaxListeners(0);
110:
111:         const connect$ = this.connect$(this.client);
112:         const withDisconnect$ = this.mergeDisconnectEvent(
113:           this.client,
114:           connect$,
115:         ).pipe(switchMap(() => this.createChannel()));
116:
117:         const withReconnect$ = fromEvent(this.client,
CONNECT_EVENT).pipe(skip(1));
118:         const source$ = merge(withDisconnect$, withReconnect$);
119:
120:         this.connection$ = new ReplaySubject(1);
121:         source$.subscribe(this.connection$);
122:
123:         return this.convertConnectionToPromise();
124:       }
125:
126:       public createChannel(): Promise<void> {
127:         return new Promise(resolve => {
128:           this.channel = this.client.createChannel({
129:             json: false,
130:             setup: (channel: Channel) => this.setupChannel(channel,
resolve),
131:           });
132:         });
133:       }
134:
135:       public createClient(): AmqpConnectionManager {
136:         const socketOptions = this.getOptionsProp(this.options,
'socketOptions');
137:         return rqmPackage.connect(this.urls, {
138:           connectionOptions: socketOptions,
139:         });
140:       }
141:
142:       public mergeDisconnectEvent<T = any>(
143:         instance: any,
144:         source$: Observable<T>,
145:       ): Observable<T> {
146:         const eventToError = (eventType: string) =>
147:           fromEvent(instance, eventType).pipe(
148:             map((err: unknown) => {
149:               throw err;
150:             }),
151:           );
152:         const disconnect$ = eventToError(DISCONNECT_EVENT);
153:
154:         const urls = this.getOptionsProp(this.options, 'urls', []);
155:         const connectFailed$ = eventToError(CONNECT_FAILED_EVENT).pipe(
```

```
156:           retryWhen(e =>
157:             e.pipe(
158:               scan((errorCount, error: any) => {
159:                 if (urls.indexOf(error.url) >= urls.length - 1) {
160:                   throw error;
161:                 }
162:                 return errorCount + 1;
163:               }, 0),
164:             ),
165:           ),
166:         );
167:         // If we ever decide to propagate all disconnect errors & re-emit
them through
168:         // the "connection" stream then comment out "first()" operator.
169:         return merge(source$, disconnect$, connectFailed$).pipe(first());
170:       }
171:
172:       public async convertConnectionToPromise() {
173:         try {
174:           return await firstValueFrom(this.connection$);
175:         } catch (err) {
176:           if (err instanceof EmptyError) {
177:             return;
178:           }
179:           throw err;
180:         }
181:       }
182:
183:       public async setupChannel(channel: Channel, resolve: Function) {
184:         const prefetchCount =
185:           this.getOptionsProp(this.options, 'prefetchCount') ||
186:           RQM_DEFAULT_PREFETCH_COUNT;
187:         const isGlobalPrefetchCount =
188:           this.getOptionsProp(this.options, 'isGlobalPrefetchCount') ||
189:           RQM_DEFAULT_IS_GLOBAL_PREFETCH_COUNT;
190:
191:         if (!this.queueOptions.noAssert) {
192:           await channel.assertQueue(this.queue, this.queueOptions);
193:         }
194:         await channel.prefetch(prefetchCount, isGlobalPrefetchCount);
195:         await this.consumeChannel(channel);
196:         resolve();
197:       }
198:
199:       public async consumeChannel(channel: Channel) {
200:         const noAck = this.getOptionsProp(this.options, 'noAck',
RQM_DEFAULT_NOACK);
201:         await channel.consume(
202:           this.replyQueue,
203:           (msg: ConsumeMessage) =>
```

```
204:             this.responseEmitter.emit(msg.properties.correlationId, msg),
205:           {
206:             noAck,
207:           },
208:         );
209:       }
210:
211:       public handleError(client: AmqpConnectionManager): void {
212:         client.addListener(ERROR_EVENT, (err: any) =>
this.logger.error(err));
213:       }
214:
215:       public handleDisconnectError(client: AmqpConnectionManager): void {
216:         client.addListener(DISCONNECT_EVENT, (err: any) => {
217:           this.logger.error(DISCONNECTED_RMQ_MESSAGE);
218:           this.logger.error(err);
219:         });
220:       }
221:
222:       public async handleMessage(
223:         packet: unknown,
224:         callback: (packet: WritePacket) => any,
225:       );
226:       public async handleMessage(
227:         packet: unknown,
228:         options: Record<string, unknown>,
229:         callback: (packet: WritePacket) => any,
230:       );
231:       public async handleMessage(
232:         packet: unknown,
233:         options: Record<string, unknown> | ((packet: WritePacket) => any),
234:         callback?: (packet: WritePacket) => any,
235:       ) {
236:         if (isFunction(options)) {
237:           callback = options as (packet: WritePacket) => any;
238:           options = undefined;
239:         }
240:
241:         const { err, response, isDisposed } = await
this.deserializer.deserialize(
242:           packet,
243:           options,
244:         );
245:         if (isDisposed || err) {
246:           callback({
247:             err,
248:             response,
249:             isDisposed: true,
250:           });
251:         }
```

```
252:         callback({
253:           err,
254:           response,
255:         });
256:       }
257:
258:       protected publish(
259:         message: ReadPacket,
260:         callback: (packet: WritePacket) => any,
261:       ): () => void {
262:         try {
263:           const correlationId = randomStringGenerator();
264:           const listener = ({
265:             content,
266:             options,
267:           }: {
268:             content: Buffer;
269:             options: Record<string, unknown>;
270:           }) =>
271:             this.handleMessage(
272:               this.parseMessageContent(content),
273:               options,
274:               callback,
275:             );
276:
277:           Object.assign(message, { id: correlationId });
278:           const serializedPacket: ReadPacket & Partial<RmqRecord> =
279:             this.serializer.serialize(message);
280:
281:           const options = serializedPacket.options;
282:           delete serializedPacket.options;
283:
284:           this.responseEmitter.on(correlationId, listener);
285:           this.channel
286:             .sendToQueue(
287:               this.queue,
288:               Buffer.from(JSON.stringify(serializedPacket)),
289:               {
290:                 replyTo: this.replyQueue,
291:                 persistent: this.persistent,
292:                 ...options,
293:                 headers: this.mergeHeaders(options?.headers),
294:                 correlationId,
295:               },
296:             )
297:             .catch(err => callback({ err }));
298:           return () => this.responseEmitter.removeListener(correlationId,
listener);
299:         } catch (err) {
300:           callback({ err });
```

```
301:         }
302:       }
303:
304:       protected dispatchEvent(packet: ReadPacket): Promise<any> {
305:         const serializedPacket: ReadPacket & Partial<RmqRecord> =
306:           this.serializer.serialize(packet);
307:
308:         const options = serializedPacket.options;
309:         delete serializedPacket.options;
310:
311:         return new Promise<void>((resolve, reject) =>
312:           this.channel.sendToQueue(
313:             this.queue,
314:             Buffer.from(JSON.stringify(serializedPacket)),
315:             {
316:               persistent: this.persistent,
317:               ...options,
318:               headers: this.mergeHeaders(options?.headers),
319:             },
320:             (err: unknown) => (err ? reject(err) : resolve()),
321:           ),
322:         );
323:       }
324:
325:       protected initializeSerializer(options: RmqOptions['options']) {
326:         this.serializer = options?.serializer ?? new RmqRecordSerializer();
327:       }
328:
329:       protected mergeHeaders(
330:         requestHeaders?: Record<string, string>,
331:       ): Record<string, string> | undefined {
332:         if (!requestHeaders && !this.options?.headers) {
333:           return undefined;
334:         }
335:
336:         return {
337:           ...this.options?.headers,
338:           ...requestHeaders,
339:         };
340:       }
341:
342:       protected parseMessageContent(content: Buffer) {
343:         const rawContent = content.toString();
344:         try {
345:           return JSON.parse(rawContent);
346:         } catch {
347:           return rawContent;
348:         }
349:       }
350:     }
```

Лістинг 3.2

Логіка обслуговування подій

```
1:  import { Injectable } from '@nestjs/common/interfaces';
2:  import { Controller } from
'@nestjs/common/interfaces/controllers/controller.interface';
3:  import { isUndefined } from '@nestjs/common/utils/shared.utils';
4:  import { ContextIdFactory } from '@nestjs/core/helpers/context-id-factory';
5:  import { ExecutionContextHost } from '@nestjs/core/helpers/execution-
context-host';
6:  import { STATIC_CONTEXT } from '@nestjs/core/injector/constants';
7:  import { NestContainer } from '@nestjs/core/injector/container';
8:  import { Injector } from '@nestjs/core/injector/injector';
9:  import {
10:   ContextId,
11:   InstanceWrapper,
12: } from '@nestjs/core/injector/instance-wrapper';
13: import { Module } from '@nestjs/core/injector/module';
14: import { GraphInspector } from '@nestjs/core/inspector/graph-inspector';
15: import { MetadataScanner } from '@nestjs/core/metadata-scanner';
16: import { REQUEST_CONTEXT_ID } from '@nestjs/core/router/request/request-
constants';
17: import {
18:   forkJoin,
19:   from as fromPromise,
20:   isObservable,
21:   mergeMap,
22:   Observable,
23:   ObservedValueOf,
24:   of,
25: } from 'rxjs';
26: import { IClientProxyFactory } from './client/client-proxy-factory';
27: import { ClientsContainer } from './container';
28: import { ExceptionFiltersContext } from './context/exception-filters-
context';
29: import { RequestContextHost } from './context/request-context-host';
30: import { RpcContextCreator } from './context/rpc-context-creator';
31: import {
32:   DEFAULT_CALLBACK_METADATA,
33:   DEFAULT_GRPC_CALLBACK_METADATA,
34: } from './context/rpc-metadata-constants';
35: import { BaseRpcContext } from './ctx-host/base-rpc.context';
36: import { Transport } from './enums';
37: import {
38:   CustomTransportStrategy,
39:   MessageHandler,
40:   PatternMetadata,
41:   RequestContext,
42: } from './interfaces';
43: import { MicroserviceEntrypointMetadata } from './interfaces/microservice-
entrypoint-metadata.interface';
```

```
44: import {
45:   EventOrMessageListenerDefinition,
46:   ListenerMetadataExplorer,
47: } from './listener-metadata-explorer';
48: import { ServerGrpc } from './server';
49: import { Server } from './server/server';
50:
51: export class ListenersController {
52:   private readonly metadataExplorer = new ListenerMetadataExplorer(
53:     new MetadataScanner(),
54:   );
55:   private readonly exceptionFiltersCache = new WeakMap();
56:
57:   constructor(
58:     private readonly clientsContainer: ClientsContainer,
59:     private readonly contextCreator: RpcContextCreator,
60:     private readonly container: NestContainer,
61:     private readonly injector: Injector,
62:     private readonly clientFactory: IClientProxyFactory,
63:     private readonly exceptionFiltersContext: ExceptionFiltersContext,
64:     private readonly graphInspector: GraphInspector,
65:   ) {}
66:
67:   public registerPatternHandlers(
68:     instanceWrapper: InstanceWrapper<Controller | Injectable>,
69:     server: Server & CustomTransportStrategy,
70:     moduleKey: string,
71:   ) {
72:     const { instance } = instanceWrapper;
73:
74:     const isStatic = instanceWrapper.isDependencyTreeStatic();
75:     const patternHandlers = this.metadataExplorer.explore(instance as
object);
76:     const moduleRef = this.container.getModuleByKey(moduleKey);
77:     const defaultCallMetadata =
78:       server instanceof ServerGrpc
79:         ? DEFAULT_GRPC_CALLBACK_METADATA
80:         : DEFAULT_CALLBACK_METADATA;
81:
82:     patternHandlers
83:       .filter(
84:         ({ transport }) =>
85:           isUndefined(transport) ||
86:           isUndefined(server.transportId) ||
87:           transport === server.transportId,
88:       )
89:       .reduce((acc, handler) => {
90:         handler.patterns.forEach(pattern =>
91:           acc.push({ ...handler, patterns: [pattern] }),
92:         );
```

```
93:         return acc;
94:       }, [])
95:       .forEach((definition: EventOrMessageListenerDefinition) => {
96:         const {
97:           patterns: [pattern],
98:           targetCallback,
99:           methodKey,
100:              extras,
101:              isEventHandler,
102:            } = definition;
103:
104:            this.insertEntrypointDefinition(
105:              instanceWrapper,
106:              definition,
107:              server.transportId,
108:            );
109:
110:            if (isStatic) {
111:              const proxy = this.contextCreator.create(
112:                instance as object,
113:                targetCallback,
114:                moduleKey,
115:                methodKey,
116:                STATIC_CONTEXT,
117:                undefined,
118:                defaultCallMetadata,
119:              );
120:              if (isEventHandler) {
121:                const eventHandler: MessageHandler = async (...args:
unknown[]) => {
122:                  const originalArgs = args;
123:                  const [dataOrContextHost] = originalArgs;
124:                  if (dataOrContextHost instanceof RequestContextHost) {
125:                    args = args.slice(1, args.length);
126:                  }
127:                  const returnValue = proxy(...args);
128:                  return this.forkJoinHandlersIfAttached(
129:                    returnValue,
130:                    originalArgs,
131:                    eventHandler,
132:                  );
133:                };
134:                return server.addHandler(
135:                  pattern,
136:                  eventHandler,
137:                  isEventHandler,
138:                  extras,
139:                );
140:              } else {
```

```
141:                 return server.addHandler(pattern, proxy, isEventHandler,
extras);
142:               }
143:             }
144:             const asyncHandler = this.createRequestScopedHandler(
145:               instanceWrapper,
146:               pattern,
147:               moduleRef,
148:               moduleKey,
149:               methodKey,
150:               defaultCallMetadata,
151:               isEventHandler,
152:             );
153:             server.addHandler(pattern, asyncHandler, isEventHandler,
extras);
154:           });
155:       }
156:
157:       public insertEntrypointDefinition(
158:         instanceWrapper: InstanceWrapper,
159:         definition: EventOrMessageListenerDefinition,
160:         transportId: Transport | symbol,
161:       ) {
162:
 this.graphInspector.insertEntrypointDefinition<MicroserviceEntrypointMetadata>
(
163:           {
164:             type: 'microservice',
165:             methodName: definition.methodKey,
166:             className: instanceWrapper.metatype?.name,
167:             classNodeId: instanceWrapper.id,
168:             metadata: {
169:               key: definition.patterns.toString(),
170:               transportId:
171:                 typeof transportId === 'number'
172:                   ? (Transport[transportId] as keyof typeof Transport)
173:                   : transportId,
174:               patterns: definition.patterns,
175:               isEventHandler: definition.isEventHandler,
176:               extras: definition.extras,
177:             },
178:           },
179:           instanceWrapper.id,
180:         );
181:       }
182:
183:       public forkJoinHandlersIfAttached(
184:         currentReturnValue: Promise<unknown> | Observable<unknown>,
185:         originalArgs: unknown[],
186:         handlerRef: MessageHandler,
```

```
187:       ) {
188:         if (handlerRef.next) {
189:           const returnedValueWrapper = handlerRef.next(
190:             ...(originalArgs as Parameters<MessageHandler>),
191:           );
192:           return forkJoin({
193:             current: this.transformToObservable(currentReturnValue),
194:             next: this.transformToObservable(returnedValueWrapper),
195:           });
196:         }
197:         return currentReturnValue;
198:       }
199:
200:       public assignClientsToProperties(instance: Controller | Injectable) {
201:         for (const {
202:           property,
203:           metadata,
204:         } of this.metadataExplorer.scanForClientHooks(instance as object))
{
205:           const client = this.clientFactory.create(metadata);
206:           this.clientsContainer.addClient(client);
207:
208:           this.assignClientToInstance(instance as object, property,
client);
209:         }
210:       }
211:
212:       public assignClientToInstance<T = any>(
213:         instance: Controller | Injectable,
214:         property: string,
215:         client: T,
216:       ) {
217:         Reflect.set(instance as object, property, client);
218:       }
219:
220:       public createRequestScopedHandler(
221:         wrapper: InstanceWrapper,
222:         pattern: PatternMetadata,
223:         moduleRef: Module,
224:         moduleKey: string,
225:         methodKey: string,
226:         defaultCallMetadata: Record<string, any> =
DEFAULT_CALLBACK_METADATA,
227:         isEventHandler = false,
228:       ) {
229:         const collection = moduleRef.controllers;
230:         const { instance } = wrapper;
231:
232:         const isTreeDurable = wrapper.isDependencyTreeDurable();
233:
```

```
234:         const requestScopedHandler: MessageHandler = async (...args:
unknown[]) => {
235:           try {
236:             let contextId: ContextId;
237:
238:             let [dataOrContextHost] = args;
239:             if (dataOrContextHost instanceof RequestContextHost) {
240:               contextId = this.getContextId(dataOrContextHost,
isTreeDurable);
241:               args.shift();
242:             } else {
243:               const [data, reqCtx] = args;
244:               const request = RequestContextHost.create(
245:                 pattern,
246:                 data,
247:                 reqCtx as BaseRpcContext,
248:               );
249:               contextId = this.getContextId(request, isTreeDurable);
250:               dataOrContextHost = request;
251:             }
252:
253:             const contextInstance = await this.injector.loadPerContext(
254:               instance,
255:               moduleRef,
256:               collection,
257:               contextId,
258:             );
259:             const proxy = this.contextCreator.create(
260:               contextInstance,
261:               contextInstance[methodKey],
262:               moduleKey,
263:               methodKey,
264:               contextId,
265:               wrapper.id,
266:               defaultCallMetadata,
267:             );
268:
269:             const returnValue = proxy(...args);
270:             if (isEventHandler) {
271:               return this.forkJoinHandlersIfAttached(
272:                 returnValue,
273:                 [dataOrContextHost, ...args],
274:                 requestScopedHandler,
275:               );
276:             }
277:             return returnValue;
278:           } catch (err) {
279:             let exceptionFilter = this.exceptionFiltersCache.get(
280:               instance[methodKey],
281:             );
```

```
282:             if (!exceptionFilter) {
283:               exceptionFilter = this.exceptionFiltersContext.create(
284:                 instance,
285:                 instance[methodKey],
286:                 moduleKey,
287:               );
288:               this.exceptionFiltersCache.set(instance[methodKey],
exceptionFilter);
289:             }
290:             const host = new ExecutionContextHost(args);
291:             host.setType('rpc');
292:             return exceptionFilter.handle(err, host);
293:           }
294:         };
295:         return requestScopedHandler;
296:       }
297:
298:       private getContextId<T extends RequestContext = any>(
299:         request: T,
300:         isTreeDurable: boolean,
301:       ): ContextId {
302:         const contextId = ContextIdFactory.getByRequest(request);
303:         if (!request[REQUEST_CONTEXT_ID as any]) {
304:           Object.defineProperty(request, REQUEST_CONTEXT_ID, {
305:             value: contextId,
306:             enumerable: false,
307:             writable: false,
308:             configurable: false,
309:           });
310:
311:           const requestProviderValue = isTreeDurable ? contextId.payload :
request;
312:           this.container.registerRequestProvider(requestProviderValue,
contextId);
313:         }
314:         return contextId;
315:       }
316:
317:       public transformToObservable<T>(
318:         resultOrDeferred: Observable<T> | Promise<T>,
319:       ): Observable<T>;
320:       public transformToObservable<T>(
321:         resultOrDeferred: T,
322:       ): never extends Observable<ObservedValueOf<T>>
323:         ? Observable<T>
324:         : Observable<ObservedValueOf<T>>;
325:       public transformToObservable(resultOrDeferred: any) {
326:         if (resultOrDeferred instanceof Promise) {
327:           return fromPromise(resultOrDeferred).pipe(
328:             mergeMap(val => (isObservable(val) ? val : of(val))),
```

```
329:           );
330:         }
331:
332:         if (isObservable(resultOrDeferred)) {
333:           return resultOrDeferred;
334:         }
335:
336:         return of(resultOrDeferred);
337:       }
338:     }
```

Лістинг 3.3

Реалізація шаблону сервісу

```
1:  import {
    2:    INestApplicationContext,
    3:    Logger,
    4:    LoggerService,
    5:    LogLevel,
    6:    ShutdownSignal,
    7:  } from '@nestjs/common';
    8:  import {
    9:    Abstract,
    10:   DynamicModule,
    11:   GetOrResolveOptions,
    12:   Type,
    13: } from '@nestjs/common/interfaces';
    14: import { NestApplicationContextOptions } from
'@nestjs/common/interfaces/nest-application-context-options.interface';
    15: import { isEmpty } from '@nestjs/common/utils/shared.utils';
    16: import { iterate } from 'iterare';
    17: import { MESSAGES } from './constants';
    18: import { UnknownModuleException } from './errors/exceptions';
    19: import { createContextId } from './helpers/context-id-factory';
    20: import {
    21:   callAppShutdownHook,
    22:   callBeforeAppShutdownHook,
    23:   callModuleBootstrapHook,
    24:   callModuleDestroyHook,
    25:   callModuleInitHook,
    26: } from './hooks';
    27: import { AbstractInstanceResolver } from './injector/abstract-instance-
resolver';
    28: import { ModuleCompiler } from './injector/compiler';
    29: import { NestContainer } from './injector/container';
    30: import { Injector } from './injector/injector';
    31: import { InstanceLinksHost } from './injector/instance-links-host';
    32: import { ContextId } from './injector/instance-wrapper';
    33: import { Module } from './injector/module';
    34:
```

```
35: /**
36:  * @publicApi
37:  */
38: export class NestApplicationContext<
39:     TOptions extends
40:       NestApplicationContextOptions = NestApplicationContextOptions,
41:   >
42:   extends AbstractInstanceResolver
43:   implements INestApplicationContext
44: {
45:   protected isInitialized = false;
46:   protected injector: Injector;
47:   protected readonly logger = new Logger(NestApplicationContext.name, {
48:     timestamp: true,
49:   });
50:
51:   private shouldFlushLogsOnOverride = false;
52:   private readonly activeShutdownSignals = new Array<string>();
53:   private readonly moduleCompiler = new ModuleCompiler();
54:   private shutdownCleanupRef?: (...args: unknown[]) => unknown;
55:   private _instanceLinksHost: InstanceLinksHost;
56:   private _moduleRefsForHooksByDistance?: Array<Module>;
57:
58:   protected get instanceLinksHost() {
59:     if (!this._instanceLinksHost) {
60:       this._instanceLinksHost = new InstanceLinksHost(this.container);
61:     }
62:     return this._instanceLinksHost;
63:   }
64:
65:   constructor(
66:     protected readonly container: NestContainer,
67:     protected readonly appOptions: TOptions = {} as TOptions,
68:     private contextModule: Module = null,
69:     private readonly scope = new Array<Type<any>>(),
70:   ) {
71:     super();
72:     this.injector = new Injector();
73:
74:     if (this.appOptions.preview) {
75:       this.printInPreviewModeWarning();
76:     }
77:   }
78:
79:   public selectContextModule() {
80:     const modules = this.container.getModules().values();
81:     this.contextModule = modules.next().value;
82:   }
83:
84:   /**
```

```
    85:    * Allows navigating through the modules tree, for example, to pull
out a specific instance from the selected module.
    86:    * @returns {INestApplicationContext}
    87:    */
    88:   public select<T>(
    89:     moduleType: Type<T> | DynamicModule,
    90:   ): INestApplicationContext {
    91:     const modulesContainer = this.container.getModules();
    92:     const contextModuleCtor = this.contextModule.metatype;
    93:     const scope = this.scope.concat(contextModuleCtor);
    94:
    95:     const moduleTokenFactory = this.container.getModuleTokenFactory();
    96:     const { type, dynamicMetadata } =
    97:       this.moduleCompiler.extractMetadata(moduleType);
    98:     const token = moduleTokenFactory.create(type, dynamicMetadata);
    99:
    100:        const selectedModule = modulesContainer.get(token);
    101:        if (!selectedModule) {
    102:          throw new UnknownModuleException();
    103:        }
    104:        return new NestApplicationContext(
    105:          this.container,
    106:          this.appOptions,
    107:          selectedModule,
    108:          scope,
    109:        );
    110:      }
    111:
    112:      /**
    113:       * Retrieves an instance of either injectable or controller,
otherwise, throws exception.
    114:       * @returns {TResult}
    115:       */
    116:      public get<TInput = any, TResult = TInput>(
    117:        typeOrToken: Type<TInput> | Function | string | symbol,
    118:      ): TResult;
    119:      /**
    120:       * Retrieves an instance of either injectable or controller,
otherwise, throws exception.
    121:       * @returns {TResult}
    122:       */
    123:      public get<TInput = any, TResult = TInput>(
    124:        typeOrToken: Type<TInput> | Function | string | symbol,
    125:        options: {
    126:          strict?: boolean;
    127:          each?: undefined | false;
    128:        },
    129:      ): TResult;
    130:      /**
```

```
    131:       * Retrieves a list of instances of either injectables or
controllers, otherwise, throws exception.
    132:       * @returns {Array<TResult>}
    133:       */
    134:      public get<TInput = any, TResult = TInput>(
    135:        typeOrToken: Type<TInput> | Function | string | symbol,
    136:        options: {
    137:          strict?: boolean;
    138:          each: true;
    139:        },
    140:      ): Array<TResult>;
    141:      /**
    142:       * Retrieves an instance (or a list of instances) of either
injectable or controller, otherwise, throws exception.
    143:       * @returns {TResult | Array<TResult>}
    144:       */
    145:      public get<TInput = any, TResult = TInput>(
    146:        typeOrToken: Type<TInput> | Abstract<TInput> | string | symbol,
    147:        options: GetOrResolveOptions = { strict: false },
    148:      ): TResult | Array<TResult> {
    149:        return !(options && options.strict)
    150:          ? this.find<TInput, TResult>(typeOrToken, options)
    151:          : this.find<TInput, TResult>(typeOrToken, {
    152:              moduleId: this.contextModule?.id,
    153:              each: options.each,
    154:            });
    155:      }
    156:
    157:      /**
    158:       * Resolves transient or request-scoped instance of either
injectable or controller, otherwise, throws exception.
    159:       * @returns {Array<TResult>}
    160:       */
    161:      public resolve<TInput = any, TResult = TInput>(
    162:        typeOrToken: Type<TInput> | Function | string | symbol,
    163:      ): Promise<TResult>;
    164:      /**
    165:       * Resolves transient or request-scoped instance of either
injectable or controller, otherwise, throws exception.
    166:       * @returns {Array<TResult>}
    167:       */
    168:      public resolve<TInput = any, TResult = TInput>(
    169:        typeOrToken: Type<TInput> | Function | string | symbol,
    170:        contextId?: {
    171:          id: number;
    172:        },
    173:      ): Promise<TResult>;
    174:      /**
    175:       * Resolves transient or request-scoped instance of either
injectable or controller, otherwise, throws exception.
```

```
176:        * @returns {Array<TResult>}
177:        */
178:       public resolve<TInput = any, TResult = TInput>(
179:         typeOrToken: Type<TInput> | Function | string | symbol,
180:         contextId?: {
181:           id: number;
182:         },
183:         options?: {
184:           strict?: boolean;
185:           each?: undefined | false;
186:         },
187:       ): Promise<TResult>;
188:       /**
189:        * Resolves transient or request-scoped instances of either
injectables or controllers, otherwise, throws exception.
190:        * @returns {Array<TResult>}
191:        */
192:       public resolve<TInput = any, TResult = TInput>(
193:         typeOrToken: Type<TInput> | Function | string | symbol,
194:         contextId?: {
195:           id: number;
196:         },
197:         options?: {
198:           strict?: boolean;
199:           each: true;
200:         },
201:       ): Promise<Array<TResult>>;
202:       /**
203:        * Resolves transient or request-scoped instance (or a list of
instances) of either injectable or controller, otherwise, throws exception.
204:        * @returns {Promise<TResult | Array<TResult>>}
205:        */
206:       public resolve<TInput = any, TResult = TInput>(
207:         typeOrToken: Type<TInput> | Abstract<TInput> | string | symbol,
208:         contextId = createContextId(),
209:         options: GetOrResolveOptions = { strict: false },
210:       ): Promise<TResult | Array<TResult>> {
211:         return this.resolvePerContext<TInput, TResult>(
212:           typeOrToken,
213:           this.contextModule,
214:           contextId,
215:           options,
216:         );
217:       }
218:
219:       /**
220:        * Registers the request/context object for a given context ID
(DI container sub-tree).
221:        * @returns {void}
222:        */
```

```
    223:       public registerRequestByContextId<T = any>(request: T, contextId:
ContextId) {
    224:         this.container.registerRequestProvider(request, contextId);
    225:       }
    226:
    227:       /**
    228:        * Initializes the Nest application.
    229:        * Calls the Nest lifecycle events.
    230:        *
    231:        * @returns {Promise<this>} The NestApplicationContext instance
as Promise
    232:        */
    233:       public async init(): Promise<this> {
    234:         if (this.isInitialized) {
    235:           return this;
    236:         }
    237:         await this.callInitHook();
    238:         await this.callBootstrapHook();
    239:
    240:         this.isInitialized = true;
    241:         return this;
    242:       }
    243:
    244:       /**
    245:        * Terminates the application
    246:        * @returns {Promise<void>}
    247:        */
    248:       public async close(signal?: string): Promise<void> {
    249:         await this.callDestroyHook();
    250:         await this.callBeforeShutdownHook(signal);
    251:         await this.dispose();
    252:         await this.callShutdownHook(signal);
    253:         this.unsubscribeFromProcessSignals();
    254:       }
    255:
    256:       /**
    257:        * Sets custom logger service.
    258:        * Flushes buffered logs if auto flush is on.
    259:        * @returns {void}
    260:        */
    261:       public useLogger(logger: LoggerService | LogLevel[] | false) {
    262:         Logger.overrideLogger(logger);
    263:
    264:         if (this.shouldFlushLogsOnOverride) {
    265:           this.flushLogs();
    266:         }
    267:       }
    268:
    269:       /**
    270:        * Prints buffered logs and detaches buffer.
```

```
271:        * @returns {void}
272:        */
273:       public flushLogs() {
274:         Logger.flush();
275:       }
276:
277:       /**
278:        * Define that it must flush logs right after defining a custom
logger.
279:        */
280:       public flushLogsOnOverride() {
281:         this.shouldFlushLogsOnOverride = true;
282:       }
283:
284:       /**
285:        * Enables the usage of shutdown hooks. Will call the
286:        * `onApplicationShutdown` function of a provider if the
287:        * process receives a shutdown signal.
288:        *
289:        * @param {ShutdownSignal[]} [signals=[]] The system signals it
should listen to
290:        *
291:        * @returns {this} The Nest application context instance
292:        */
293:       public enableShutdownHooks(signals: (ShutdownSignal | string)[] =
[]): this {
294:         if (isEmpty(signals)) {
295:           signals = Object.keys(ShutdownSignal).map(
296:             (key: string) => ShutdownSignal[key],
297:           );
298:         } else {
299:           // given signals array should be unique because
300:           // process shouldn't listen to the same signal more than
once.
301:           signals = Array.from(new Set(signals));
302:         }
303:
304:         signals = iterate(signals)
305:           .map((signal: string) =>
signal.toString().toUpperCase().trim())
306:           // filter out the signals which is already listening to
307:           .filter(signal =>
!this.activeShutdownSignals.includes(signal))
308:           .toArray();
309:
310:         this.listenToShutdownSignals(signals);
311:         return this;
312:       }
313:
314:       protected async dispose(): Promise<void> {
```

```
315:         // Nest application context has no server
316:         // to dispose, therefore just call a noop
317:         return Promise.resolve();
318:       }
319:
320:       /**
321:        * Listens to shutdown signals by listening to
322:        * process events
323:        *
324:        * @param {string[]} signals The system signals it should listen
to
325:        */
326:       protected listenToShutdownSignals(signals: string[]) {
327:         let receivedSignal = false;
328:         const cleanup = async (signal: string) => {
329:           try {
330:             if (receivedSignal) {
331:               // If we receive another signal while we're waiting
332:               // for the server to stop, just ignore it.
333:               return;
334:             }
335:             receivedSignal = true;
336:             await this.callDestroyHook();
337:             await this.callBeforeShutdownHook(signal);
338:             await this.dispose();
339:             await this.callShutdownHook(signal);
340:             signals.forEach(sig => process.removeListener(sig,
cleanup));
341:             process.kill(process.pid, signal);
342:           } catch (err) {
343:             Logger.error(
344:               MESSAGES.ERROR_DURING_SHUTDOWN,
345:               (err as Error)?.stack,
346:               NestApplicationContext.name,
347:             );
348:             process.exit(1);
349:           }
350:         };
351:         this.shutdownCleanupRef = cleanup as (...args: unknown[]) =>
unknown;
352:
353:         signals.forEach((signal: string) => {
354:           this.activeShutdownSignals.push(signal);
355:           process.on(signal as any, cleanup);
356:         });
357:       }
358:
359:       /**
360:        * Unsubscribes from shutdown signals (process events)
361:        */
```

```
362:       protected unsubscribeFromProcessSignals() {
363:         if (!this.shutdownCleanupRef) {
364:           return;
365:         }
366:         this.activeShutdownSignals.forEach(signal => {
367:           process.removeListener(signal, this.shutdownCleanupRef);
368:         });
369:       }
370:
371:       /**
372:        * Calls the `onModuleInit` function on the registered
373:        * modules and its children.
374:        */
375:       protected async callInitHook(): Promise<void> {
376:         const modulesSortedByDistance =
this.getModulesToTriggerHooksOn();
377:         for (const module of modulesSortedByDistance) {
378:           await callModuleInitHook(module);
379:         }
380:       }
381:
382:       /**
383:        * Calls the `onModuleDestroy` function on the registered
384:        * modules and its children.
385:        */
386:       protected async callDestroyHook(): Promise<void> {
387:         const modulesSortedByDistance =
this.getModulesToTriggerHooksOn();
388:         for (const module of modulesSortedByDistance) {
389:           await callModuleDestroyHook(module);
390:         }
391:       }
392:
393:       /**
394:        * Calls the `onApplicationBootstrap` function on the registered
395:        * modules and its children.
396:        */
397:       protected async callBootstrapHook(): Promise<void> {
398:         const modulesSortedByDistance =
this.getModulesToTriggerHooksOn();
399:         for (const module of modulesSortedByDistance) {
400:           await callModuleBootstrapHook(module);
401:         }
402:       }
403:
404:       /**
405:        * Calls the `onApplicationShutdown` function on the registered
406:        * modules and children.
407:        */
```

```
    408:       protected async callShutdownHook(signal?: string): Promise<void>
{
    409:         const modulesSortedByDistance =
this.getModulesToTriggerHooksOn();
    410:         for (const module of modulesSortedByDistance) {
    411:           await callAppShutdownHook(module, signal);
    412:         }
    413:       }
    414:
    415:       /**
    416:        * Calls the `beforeApplicationShutdown` function on the
registered
    417:        * modules and children.
    418:        */
    419:       protected async callBeforeShutdownHook(signal?: string):
Promise<void> {
    420:         const modulesSortedByDistance =
this.getModulesToTriggerHooksOn();
    421:         for (const module of modulesSortedByDistance) {
    422:           await callBeforeAppShutdownHook(module, signal);
    423:         }
    424:       }
    425:
    426:       protected assertNotInPreviewMode(methodName: string) {
    427:         if (this.appOptions.preview) {
    428:           const error = `Calling the "${methodName}" in the preview
mode is not supported.`;
    429:           this.logger.error(error);
    430:           throw new Error(error);
    431:         }
    432:       }
    433:
    434:       private getModulesToTriggerHooksOn(): Module[] {
    435:         if (this._moduleRefsForHooksByDistance) {
    436:           return this._moduleRefsForHooksByDistance;
    437:         }
    438:         const modulesContainer = this.container.getModules();
    439:         const compareFn = (a: Module, b: Module) => b.distance -
a.distance;
    440:         const modulesSortedByDistance =
Array.from(modulesContainer.values()).sort(
    441:           compareFn,
    442:         );
    443:
    444:         this._moduleRefsForHooksByDistance = this.appOptions?.preview
    445:           ? modulesSortedByDistance.filter(moduleRef =>
moduleRef.initOnPreview)
    446:           : modulesSortedByDistance;
    447:         return this._moduleRefsForHooksByDistance;
    448:       }
```

```
449:
450:       private printInPreviewModeWarning() {
451:         this.logger.warn('-------------------------------------------
---');
452:         this.logger.warn('Application is running in the PREVIEW
mode!');
453:         this.logger.warn('Providers/controllers will not be
instantiated.');
454:         this.logger.warn('-------------------------------------------
---');
455:       }
456:     }
```

Висновки до розділу

В цьому розділі детально розглянуті ключові аспекти, необхідні для розуміння при розробці розширень для браузерів. Середа виконання визначена як середовище, в якому працює розширення Chrome, включаючи ядро браузера та основні компоненти середи, різноманітні API (WebExtensions API та Web Platform APIs). Розуміння цих компонентів є ключовим для ефективної розробки розширень, оскільки вони надають доступ до функціональності браузера та веб-платформи. Зона доступу, у свою чергу, охоплює різноманітні ресурси, від даних користувача до інформації про веб-сторінки та сам браузер. Розробники розширень надаються відповідальністю дотримуватися рекомендацій політики GDPR, особливо при роботі з конфіденційною інформацією. Три основні принципи включають у себе обов'язок бути прозорим щодо збору та використання даних, надання користувачам контролю над своєю приватністю, а також використання шифрування для забезпечення безпеки даних. Важливою є ідея утримання від надання зайвих дозволів, які не пов'язані з функціональністю розширення, що допомагає зменшити ризик зловживання та неправомірного збору даних. Загальний підхід до розробки повинен бути орієнтованим на забезпечення конфіденційності та безпеки для кінцевих користувачів.

РОЗДІЛ 4
ПРОТОТИП ПРОГРАМНОГО ЗАСТОСУНКУ ТА ПЕРЕВІРКА ЙОГО ЕФЕКТИВНОСТІ

4.1. Прототип програмного застосунку

Після успішного завантаження застосунку в браузер користувач потрапляє на головне вікно застосунку (Рис. 4.1). Після цього застосунок відразу починає спостерігати за всім вкладками браузеру та при необхідності фільтрувати трафік, ці дії невидімі для користувача проте в разі необхідності користувач може переглянути їх. На головному вікні застосунку представлені зверху до низу – селектор зміни ір адреси, інформацію про поточну сесію, меню налаштувань захисту та статистика виявлених загроз. Застосунок не вимагає від користувача якихось додаткових налаштувань. Модальне вікно з’являється тільки тоді коли з’єднання є небезпечним. Користувач може керувати налаштуваннями застосунку, використовуючи правила-виключення та селектори які розміщені на головному вікні.

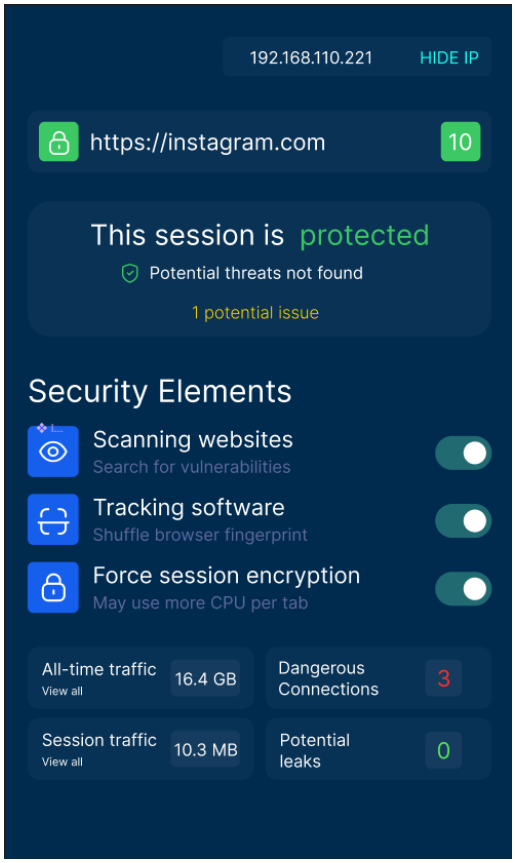

Рис. 4.1. Головне вікно застосунку у випадку якщо веб-сайт є безпечним

Під час взаємодії з модальним вікном користувач може отримати рекомендації щодо подальших дій. Наприклад, можливі варіанти можуть включати попередження про можливі ризики та рекомендації щодо того, як уникнути потенційних небезпек. Також, користувач може мати можливість надіслати звіт про виявлену загрозу для подальших заходів забезпечення безпеки. У випадку якщо користувач потрапляє на шкідливий сайт головне модальне вікно застосунку автоматично показується з поточними проблемами та загальним балом, за допомогою якого він може швидко зрозуміти рівень небезпеки та надійності відвідуваного веб-сайту (Рис 4.2).

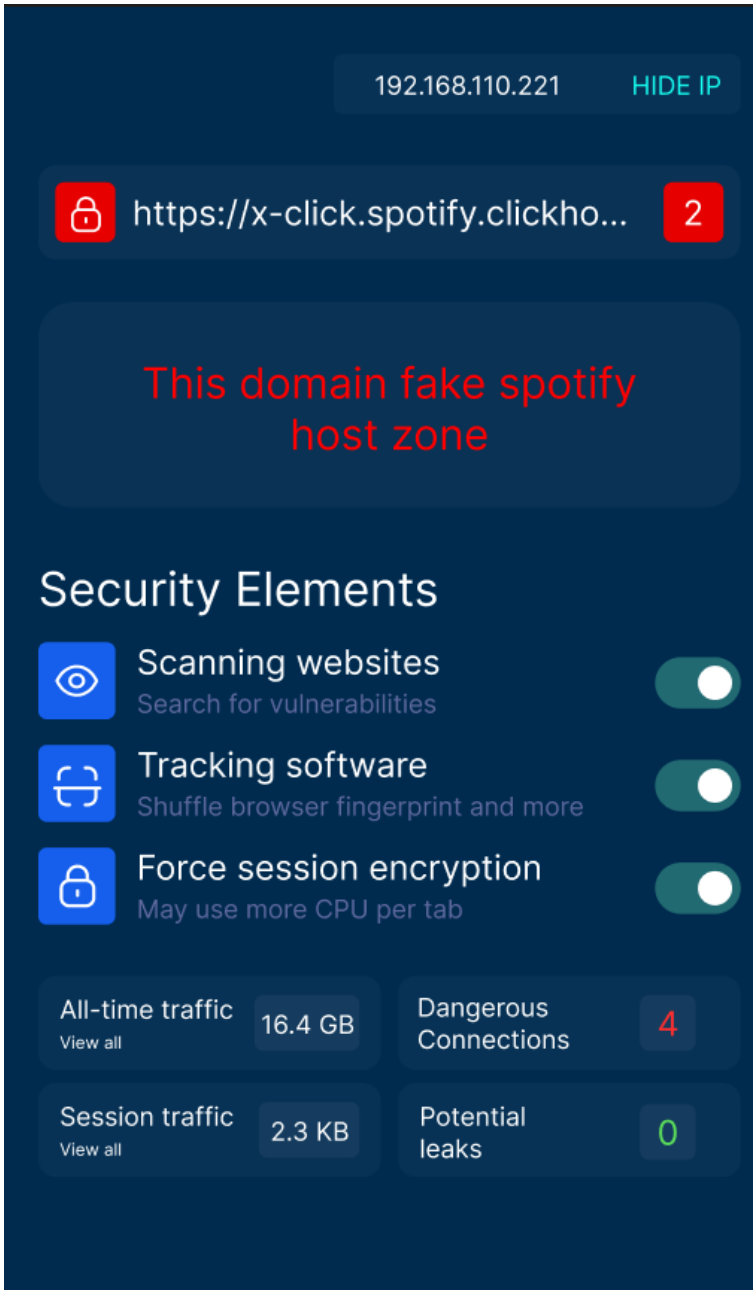

Рис. 4.2. Головне вікно застосунку у випадку якщо веб-сайт є небезпечним

При необхідності додаток може переадресувати користувача на внутрішню безпечну сторінку, у разі якщо продовжувати виконання скриптів небезпечно.

В вікні додатку(Рис 4.3) користувач також може переглянути детальну інформацію про те чому цей веб сайт є поганим і всі файли з місцями, які провокують спробу захисту. Це може бути корисним для тих хто намагається захистити чи перевірити свій веб-сайт на небезпеку.

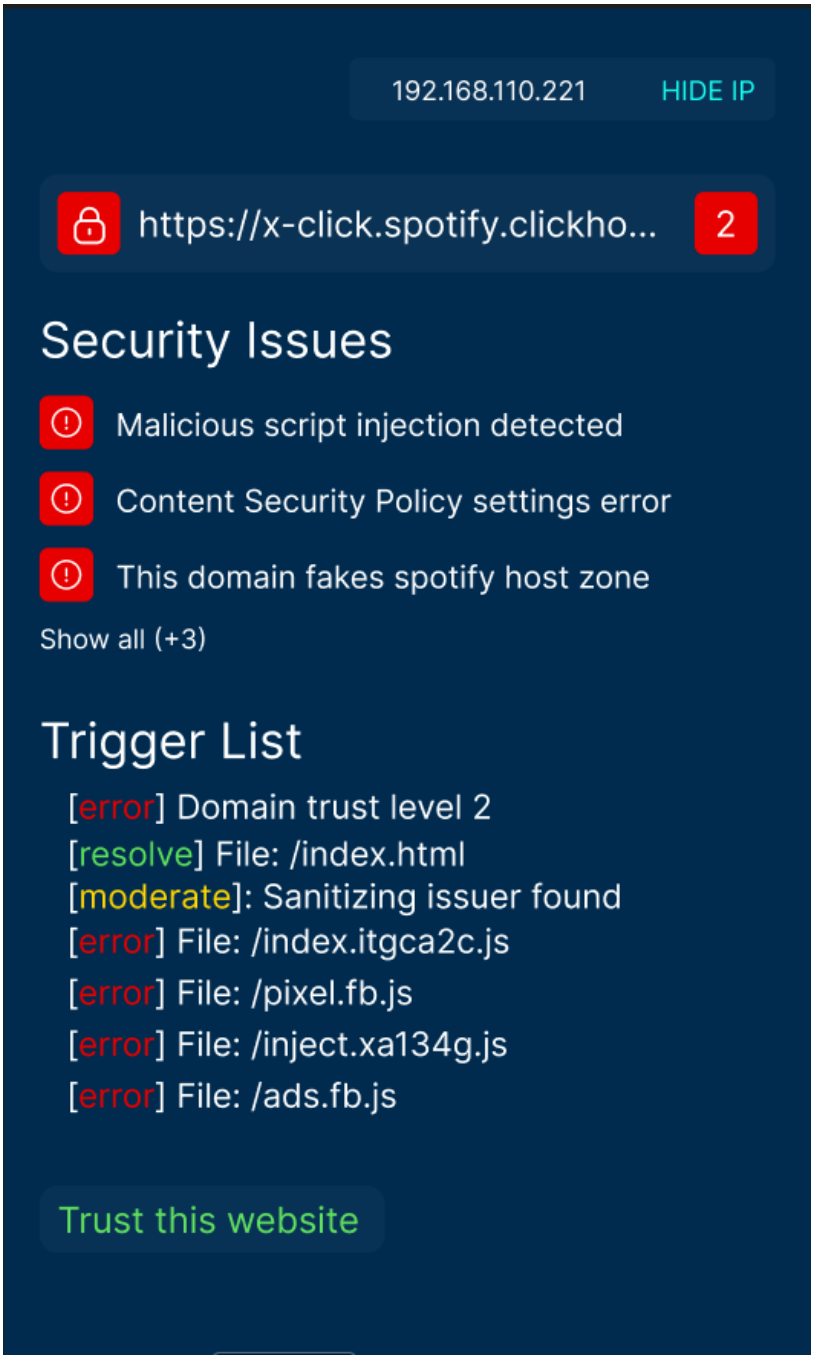

Рис. 4.3. Сторінка детальної інформації про веб-сайт

4.2. Перевірка ефективності застосунку

У даному розділі буде проведено тестування ефективності програми (розширення Chrome), що розроблене для запобігання кібератак в онлайн-середовищі. Додаток включає в себе серію служб та функцій, спрямованих на підвищення рівня безпеки користувача під час використання браузера. Тестування ефективності додатку буде проведено з використанням реальних тестових сценаріїв, що охоплюють різні аспекти безпеки в онлайн-середовищі. Результати

тестування були аналізовані для оцінки ефективності додатку та визначення його здатності ефективно захищати користувача від потенційних кіберзагроз. Нижче наведена таблиця із зразками тестів для перевірки на XSS (Cross-Site Scripting) у веб-додатку. Ці тести важливі для визначення того, чи додаток належним чином фільтрує та обробляє введені дані для запобігання вразливості XSS.

Таблиця 4.1

Тест-кейси атаки на веб-сайт

Тестовий сценарій	Вміст	Виявлено
Текстовий вміст	`<script>alert("XSS");</script>`	+
HTML-теги в атрибутах	`<img src="x" onerror="alert('XSS');">`	+
Введення через URL-параметр	`http://example.com/?name=<script>alert('XSS');</script>`	+
Обхід фільтрації	`<s%00cript>alert('XSS');</script>`	+
JavaScript URI	`javascript:alert('XSS')`	+
Обходить одинарні кавички	`'><script>alert('XSS');</script>`	+
Обходить подвійні кавички	`"><script>alert('XSS');</script>`	+
Обходить регулярні вирази	onerror="alert(String.fromCharCode(88,83,83));">`	+

Закінчення таблиці 4.1.

Обходить обробку подій	`<img src="x" onerror="javascript:alert('XSS');">`	+
Обхід простого фільтру	`<img/src=x onerror=alert('XSS')>`	+
Обхід фільтру через SVG	`<svg/onload=alert('XSS')>`	+

Обхід фільтру через фрагменти	`<iframe/src="javascript:alert('XSS')">`	+
Введення в атрибуті стилю	`<div style="background-image:url(javascript:alert('XSS'))">`	+
Ін'єкція в атрибут onClick	`<button onclick="alert('XSS')">Click me</button>`	+
Обхід фільтру через BASE64	`<img src="('XSS')">`	-
Ін'єкція в атрибут data	`<div data-name="user-input" data-content="<img src=x onerror=alert('XSS')>"></div>`	+
Ін'єкція в JavaScript URL	`<a href="javascript:alert('XSS')">Click me</a>`	+

Зростання кількості інтернет-шахраїв та фішерських атак вимагає від нас особливої уваги до виявлення небезпечних веб-сайтів, які можуть загрожувати нашим особистим даним та безпеці. У даному контексті визначення ознак контенту, які вказують на потенційно небезпечні веб-ресурси, є важливою складовою кібербезпеки. Це включає в себе розпізнавання фішингових сторінок на основі їхнього контенту. Таблиця, представлена нижче, надає конкретні ознаки, за якими можна визначити, чи має справу користувач з потенційно небезпечним веб-ресурсом.

Таблиця 4.2.

Ознаки потенційно небезпечних веб-ресурсів

Комбіновані ознаки	Веб-сторінка	Використання	Виявлено
Шахрайський URL та	www.apple-update-	Фішинговий сайт може мати неправильно написаний	+

неправильна граматика	support.com	бренд у URL та граматичні помилки на сторінці.	
Вимоги до негайного дії та велика вигода	claim-your-prize-now.biz	Сайт може пропонувати "неймовірні" подарунки та надто короткий термін для їх викупу.	+
Несподівані листи від "офіційних" джерел	Лист від "Microsoft" із проханням терміново змінити пароль та введенням конфіденційних даних.	Несподівані повідомлення можуть містити лінки на фішингові сторінки.	+
Відсутність контактної інформації та HTTPS	www.security-update.org	Фішинговий сайт може уникати надання засобів для зв'язку та відсутність захисту HTTPS.	+
Несподівані запитання та несанкціоновані вкладені файли	Лист від "PayPal" із вимогою відправити скан паспорта.	Фішери можуть використовувати неочікувані запитання та вкладені файли для збору конфіденційної інформації.	+

Закінчення таблиці 4.2.

Запит облікових даних на сторінці з акцією	www.login-for-discount-deals.com	Фішерський сайт може використовувати логін-форму під прикриттям акції чи знижки.	+

Несподівана реклама та велика кількість попереджень	urgent-action-required.info	Фішинговий сайт може викликати паніку через рекламу та тривожні повідомлення.	+
Підроблені соціальні мережі та запитання про доступ	facebook-verify-identity.net	Фішерські сайти можуть імітувати соцмережі, запитуючи про доступ до облікових даних.	+
Фальшива сторінка для зміни паролю з логотипом банку	www.bank-password-reset.com	Сайт може імітувати банк і вимагати зміни паролю, але насправді використовує дані для шахрайства.	+
Відсутність аутентичних відгуків чи рецензій	www.trusted-reviews-site.org	Фішинговий сайт може не мати аутентичних відгуків та рецензій.	+

Висновки до розділу

У ході розробки прототипу програмного застосунку та проведення його ефективності було здійснено цілий комплекс робіт, спрямованих на створення функціонального і надійного інструменту для вирішення визначених завдань. Прототип був створений відповідно до вимог та залучав увесь необхідний функціонал для ефективної реалізації поставлених завдань. В ході перевірки ефективності програмного застосунку були проведені тестування, що дозволило оцінити його працездатність та взаємодію з користувачем. Результати тестів свідчать про високу стабільність та продуктивність розробленого прототипу.

Застосунок виявився ефективним у вирішенні поставлених завдань, а його інтерфейс взаємодії з користувачем сприяє легкості використання та зручності у роботі

Отже, можна зробити висновок, що розроблений прототип програмного застосунку відповідає вимогам технічного завдання та виявляється ефективним інструментом для вирішення поставлених завдань. Подальший розвиток та оптимізація можуть ще покращити його функціональність та надійність.

ВИСНОВКИ

В процесі виконання даної кваліфікаційної роботи був проведений аналіз існуючих методів захисту від кіберзагроз , що надав можливість з’ясувати актуальні та ефективні стратегії в цій сфері. На основі цього було створено

комбіновану методику запобігання кібератакам. Структура сервісів програмного застосунку має декілька рівнів взаємодії, було виділено ключові елементи та їх взаємодію в системі. Нарешті, був розроблений прототип програмного застосунку, а також проведено перевірку його ефективності.

Отримані результати вказують на позитивний результат у забезпеченні безпеки браузера користувача в сучасних умовах. Враховуючи поширеність кіберзагроз у сучасному інформаційному середовищі, розроблений прототип та методика запобігання можуть стати корисними у підвищення рівня безпеки та стійкості інформаційних систем. Однак, важливо зауважити, що швидкий розвиток технологій і кіберзагрози вимагають постійного удосконалення заходів захисту.

Майбутні дослідження та вдосконалення прототипу можуть сприяти подальшому розвитку сфери кібербезпеки та підвищенню рівня стійкості інформаційних систем до нових загроз.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. М. В. Карташов. “Імовірність, процеси, статистика.” – Київ : ВПЦ Київський університет, 2007

2. В.А. Шовкута, С.В. Фролов. “Аналіз механізмів захисту та вразливостей бездротових Wi-Fi мереж”, 2016
3. В.І. Маліновський. “Мінімізація факторів кіберзагроз і спеціалізовані підходи до інформаційного захисту мікропроцесорних систем індустріального Інтернету речей”, 2022
4. J. McGahagan, D. Bhansali, C. Pinto-Coelho, and M. Cukier. “Discovering features for detecting malicious websites: an empirical study,” 2021.
5. R. Mohammad, F. Thabtah, and T. McCluskey. “An assessment of features related to phishing websites using an automated technique,” – London, 2012
6. E. C. Blessie and E. Karthikeyan. “Sigmis: a feature selection algorithm using correlation based method” Journal of Algorithms & Computations, 2012.
7. M. Mamun, M. Rathore, A. Lashkari, N. Stakhanova, and A. Ghorbani, “Detecting malicious URLs using lexical analysis”, Taiwan, 2016.
8. V. Dogra, S. Verma, Kavita et al., “A complete process of text classification system using state-of-the-art NLP models”, 2022.
9. F. Alkhudair, M. Alassaf, R. Khan, and S. Alfarraj, “Detecting malicious URL” Tabuk, 2020.
10. D. Wang, S. B. Navathe, L. Liu, D. Irani, A. Tamersoy, and C. Pu, “Click traffic analysis of short URL spam on Twitter” Austin, 2013.
11. A. Singh and N. Goyal, “MalCrawler: {A} crawler for seeking and crawling malicious websites”, Bhubaneswar, 2017
12. M. Atrees, A. Ahmad, and F. Alghanim, “Enhancing detection of malicious URLs using boosting and lexical features,” *Intelligent Automation & Soft Computing*, 2022.