ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка веб-сторінки для динамічного виявлення та блокування шкідливих файлів під час завантаження»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим БЄЛОУСОВ

Виконав: здобувач вищої освіти гр.ШІД-41
Максим БЄЛОУСОВ

Керівник: Сергій ЄЛЬЧЕНКО
к.т.н., доцент

Рецензент:
_____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту

_____ Ольга ЗІНЧЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бєлоусова Максима Юрійовича

1. Тема кваліфікаційної роботи: Розробка веб-сторінки для динамічного виявлення та блокування шкідливих файлів під час завантаження

керівник кваліфікаційної роботи Сергій ЄЛЬЧЕНКО к.т.н., доцент,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. № 36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, принципи розробки веб-сторінок, методи динамічного виявлення шкідливих файлів, вимоги до систем безпеки веб-додатків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз сучасних методів виявлення шкідливих файлів під час завантаження
Принципи розробки веб-сторінок для забезпечення безпеки
Розробка алгоритму динамічного виявлення шкідливих файлів

Інтеграція алгоритму в систему завантаження файлів на веб-сторінці
Тестування та оптимізація веб-сторінки для ефективного блокування шкідливих файлів

5. Перелік графічного матеріалу: *презентація*
    1. Розробка та інтеграція веб-сторінки
    2. Характеристики алгоритмів виявлення шкідливих файлів
    3. Архітектура веб-додатків для забезпечення безпеки
    4. Приклади успішного блокування шкідливих файлів

6. Дата видачі завдання «27» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз методів динамічного виявлення шкідливих файлів	27.02-05.11.23	Виконано
2	Початок розробки алгоритму динамічного виявлення шкідливих файлів	05.11-12.11.23	Виконано
3	Початок тестування веб-сторінки	13.11-19.11.23	Виконано
4	Оптимізація алгоритму виявлення шкідливих файлів	20.11-25.11.23	Виконано
5	Завершення тестування веб-сторінки	27.11-03.12.23	Виконано
6	Виведення проекту на hosting	04.12-10.12.23	Виконано
7	Останні правки та доопрацювання розрахунково-пояснювальної записки	11.12-20.12.23	Виконано
8	Розробка демонстраційних матеріалів	21.12-31.05.24	Виконано

Здобувач вищої освіти _____ Максим БЄЛОУСОВ

Керівник
кваліфікаційної роботи _____ Сергій ЄЛЬЧЕНКО

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 70 стор., 20 рис., 36 джерел.

Мета роботи – розробка веб-сторінки для динамічного виявлення та блокування шкідливих файлів під час завантаження.

Об'єкт дослідження – процес захисту інформаційних систем від шкідливих файлів.

Предмет дослідження – методи і засоби виявлення та блокування шкідливих файлів під час завантаження.

Короткий зміст роботи: У роботі проведено огляд існуючих технологій захисту веб-сторінок від шкідливих файлів. Проведено порівняльний аналіз різних підходів до динамічного виявлення шкідливого програмного забезпечення, таких як сигнатурний аналіз, евристичний аналіз та поведінковий аналіз. Розглянуто можливості використання JavaScript, HTML, CSS для розробки інтерактивного інтерфейсу веб-сторінки, що дозволяє користувачам завантажувати файли з перевіркою на шкідливість. Використано Node.js для створення серверної частини та інтеграції з антивірусним програмним забезпеченням через RESTful API. Для зберігання інформації про завантаження файлів застосовано базу даних MySQL. Окрему увагу приділено питанням безпеки та захисту особистих даних користувачів під час роботи з веб-сторінкою. Для забезпечення надійності системи застосовано HTTPS протокол та механізми аутентифікації і авторизації користувачів.

КЛЮЧОВІ СЛОВА: ВЕБ-СТОРІНКА, ШКІДЛИВІ ФАЙЛИ, ВИЯВЛЕННЯ, БЛОКУВАННЯ, АНТИВІРУС, ДИНАМІЧНИЙ АНАЛІЗ, БЕЗПЕКА.

ЗМІСТ

ВСТУП

У сучасному цифровому світі, де обсяги даних зростають експоненційно, а інтернет стає все більш інтегрованим у повсякденне життя людей, питання кібербезпеки набувають особливої актуальності. В останні роки ми стали свідками значного зростання кібератак, що вражають не тільки окремих індивідуумів, але й великі корпорації та навіть урядові структури. В такому контексті розробка ефективних засобів виявлення та блокування шкідливих файлів є не просто важливою, а життєво необхідною задачею, що вимагає негайного вирішення.

Мета даної роботи полягає в описі веб-сторінки для динамічного виявлення та блокування шкідливих файлів, яка б використовувала сучасні методики та технології для забезпечення надійного захисту користувачів від кіберзагроз.

Задачі дослідження включають:

1. Аналіз сучасного стану кіберзагроз та методів виявлення шкідливих файлів.
2. Визначення ключових вимог до системи виявлення та блокування шкідливих файлів.
3. Розробка архітектури програмного рішення на основі вибраних технологій.
4. Імплементація функціоналу для сканування, виявлення та блокування шкідливих файлів.
5. Тестування розробленої системи та аналіз її ефективності.

Об'єктом дослідження є процеси виявлення та блокування шкідливих файлів у мережі Інтернет.

Предметом дослідження є методи та технології розробки веб-орієнтованих систем для кіберзахисту.

Методологія дослідження базується на комплексному підході, що включає теоретичний аналіз наявних наукових робіт та статей, використання методів проектування програмного забезпечення, програмування та тестування.

Наукова новизна роботи полягає в інтеграції сучасних технологій штучного інтелекту та машинного навчання для підвищення точності та ефективності виявлення шкідливих файлів, а також у розробці гнучкої архітектури системи, яка дозволяє легко адаптуватися до нових видів загроз.

Теоретичне значення дослідження виражається у глибокому аналізі сучасних кіберзагроз і методів боротьби з ними, що може слугувати основою для подальших наукових робіт у галузі кібербезпеки.

Практичне значення дослідження полягає в створенні функціональної веб-сторінки, яка здатна ефективно захищати користувачів від шкідливих файлів, забезпечуючи високий рівень безпеки даних в Інтернеті.

Інформаційна база дослідження охоплює наукові статті, звіти дослідницьких інститутів у галузі кібербезпеки, офіційні документи стандартів безпеки, а також документацію та вихідний код відкритих проектів, що займаються розробкою інструментів для виявлення та нейтралізації шкідливих програм.

Структура роботи включає вступ, три основні розділи, що покривають теоретичні основи виявлення шкідливих файлів, програмну реалізацію веб-сторінки для їх блокування, розробку користувацького інтерфейсу та механізми зворотного зв'язку, а також висновки, у яких підбиваються основні результати дослідження та визначаються напрямки подальшої роботи.

1 ТЕОРЕТИЧНІ ОСНОВИ ВИЯВЛЕННЯ ТА БЛОКУВАННЯ ШКІДЛИВИХ ФАЙЛІВ

1.1 Огляд шкідливого ПЗ та його типи

Шкідливе програмне забезпечення, або малвар (від англ. "malicious software"), — це загальний термін, що описує будь-яку програму або код, спеціально розроблені для виконання небажаних або зловмисних дій на комп'ютері, мережі або іншому цифровому пристрої. Ці дії можуть включати завдання шкоди системі, крадіжку, зміну або видалення даних, несанкціонований доступ до системних ресурсів і навіть контроль над цілими мережами заражених пристроїв. Шкідливе ПЗ може приймати різні форми, включаючи віруси, троянські програми, черв'яки, шпигунське ПЗ, рекламне ПЗ та вимагачі, кожен з яких має свої особливості та методи поширення [1].

Ці програми часто маскуються під легітимне програмне забезпечення або розповсюджуються шляхом експлуатації вразливостей у програмному забезпеченні. З моменту свого створення та поширення шкідливе ПЗ еволюціонувало, стаючи дедалі складнішим і винахідливішим, що вимагає від захисних механізмів неперервної адаптації та розвитку.

Шкідливе ПЗ становить значну загрозу не тільки для індивідуальних користувачів, але й для організацій та навіть урядів, оскільки може призвести до втрати важливої інформації, фінансових збитків, порушення роботи критично важливих систем та інших негативних наслідків. У світлі цього, розуміння природи шкідливого ПЗ, його типів та методів поширення є фундаментальним аспектом кібербезпеки, який дозволяє розробляти ефективні стратегії захисту та запобігання кібератакам [3].

Шкідливе програмне забезпечення, або малвар, являє собою широкий спектр програм і файлів, створених із метою виконання небажаних або шкідливих дій у цифрових системах, включаючи комп'ютери, мобільні телефони та сервери. Завдання, які виконує шкідливе ПЗ, варіюються від крадіжки конфіденційної інформації та персональних даних до завдання шкоди операційним системам, злому безпеки мережі,

і навіть до контролю над цілими мережами інфікованих пристроїв. Таке ПЗ може маскуватися під законне програмне забезпечення або розповсюджуватися за допомогою експлуатації вразливостей у вже існуючому програмному забезпеченні. Основні типи малвару включають віруси, які розповсюджуються, інфікуючи інші файли; черв'яки, що можуть самостійно копіюватися на інші комп'ютери через мережеві з'єднання; троянські програми, які приховують свою шкідливу поведінку під виглядом корисних програм; шпигунське ПЗ, призначене для збору інформації без відома користувача; рекламне ПЗ, що нав'язливо відображає рекламні оголошення; та рансомвар, який блокує доступ до системи або файлів, вимагаючи викуп за їх розблокування.

Зростання кількості та складності кібератак підкреслює критичну необхідність постійного дослідження у сфері кібербезпеки для розробки нових методів виявлення та захисту від шкідливих програм. Використання передових технологій, таких як штучний інтелект та машинне навчання, дозволяє створювати більш ефективні системи для протистояння малвару, адаптуючись до його еволюції та вдосконалюючи механізми виявлення та блокування в реальному часі [4].

Для організацій наслідки інфекції шкідливим ПЗ можуть бути ще більш катастрофічними. Вони включають порушення операційної діяльності, втрату важливих бізнес-даних, інтелектуальної власності, а також клієнтської інформації, що може призвести до значних фінансових втрат та падіння довіри з боку клієнтів та партнерів. Витік конфіденційної інформації може мати довготривалі наслідки, включаючи юридичні позови, штрафи за порушення нормативних вимог та постійне пошкодження репутації. Крім того, організації можуть зіткнутися з вимушеним простоєм систем, що веде до додаткових втрат доходів та збільшення витрат на відновлення системи та відновлення даних.

У відповідь на ці загрози, як індивідуальні користувачі, так і організації повинні вживати проактивних заходів для захисту своїх систем та даних. Це включає регулярне оновлення програмного забезпечення та операційних систем, використання надійних антивірусних рішень, освіту співробітників щодо загроз кібербезпеки та безпечних

практик поводження з даними, а також розробку та впровадження комплексної стратегії кібербезпеки, яка включає регулярне резервне копіювання даних та план відновлення після інцидентів. За допомогою цих заходів, як індивідуальні користувачі, так і організації можуть значно знизити ризик інфекції шкідливим ПЗ та мінімізувати потенційні наслідки в разі атаки.

Трояни та черв'яки є двома з найпоширеніших типів шкідливого програмного забезпечення (ПЗ), кожен з яких має унікальні характеристики та способи розповсюдження, що ставлять під загрозу цифрову безпеку індивідуальних користувачів та організацій.

Трояни названі на честь легендарного Троянського коня, оскільки вони використовують аналогічну тактику обману для введення в оману користувачів та отримання доступу до їх комп'ютерів. Троян маскується під легітимне програмне забезпечення або вбудовується в легітимні програми, що змушує користувача встановити його, вважаючи, що він безпечний. Однак, на відміну від легітимного програмного забезпечення, після активації троян може виконувати шкідливі дії, такі як викрадення даних, встановлення іншого шкідливого ПЗ або надання віддаленого доступу до інфікованої системи кіберзлочинцям. Трояни не можуть самостійно розповсюджуватися від однієї системи до іншої, їхня "інфекція" відбувається виключно через дії користувачів, які завантажують та встановлюють шкідливе ПЗ.

Черв'яки є самостійними програмами, які можуть розповсюджуватися без втручання користувача, використовуючи мережеві з'єднання та вразливості в програмному забезпеченні. Вони можуть автоматично копіювати себе на інші комп'ютери через мережі, електронні листи або інші файлові системи. Черв'яки можуть завдавати шкоди шляхом споживання системних ресурсів, таких як пропускна здатність мережі або процесорний час, що призводить до уповільнення або зупинки систем. Крім того, черв'яки часто містять додаткові шкідливі вантажі, які можуть виконувати різноманітні зловмисні дії, від викрадення даних до встановлення додаткового шкідливого ПЗ [6].

Шпигунське програмне забезпечення, відоме також як шпигунський софт або spyware, є одним з типів малвару, який призначений для таємного моніторингу діяльності користувача на його комп'ютері або мобільному пристрої без його відома чи згоди. Цей тип шкідливого ПЗ може збирати різноманітну інформацію, від історії перегляду веб-сторінок та клавіатурних натискань до особистих і фінансових даних, таких як паролі, номери банківських рахунків, дані кредитних карт та іншу конфіденційну інформацію.

Шпигунське ПЗ часто розповсюджується за допомогою фішингових електронних листів, шкідливих веб-сайтів або маскується під законні програми, які користувач може завантажити та встановити, не підозрюючи про небезпеку. В деяких випадках шпигунське ПЗ може бути встановлено на пристрій в результаті експлуатації системних вразливостей, без будь-яких дій з боку користувача.

Однією з особливих характеристик шпигунського ПЗ є його здатність працювати в прихованому режимі, уникаючи виявлення користувачем або антивірусним програмним забезпеченням. Це створює серйозні виклики для виявлення та видалення такого типу малвару.

Вплив шпигунського ПЗ на індивідуальних користувачів та організації може бути значним, включаючи втрату приватності, фінансові втрати в результаті крадіжки ідентичності або шахрайства, а також потенційне розголошення конфіденційної або комерційної інформації. Особливо це стосується організацій, де шпигунське ПЗ може бути використане конкурентами або зловмисниками для збору стратегічної інформації, що може призвести до втрати конкурентоспроможності, юридичних проблем або шкоди репутації [5].

Хоча троянські програми, черв'яки і ransomware є різними типами шкідливого програмного забезпечення, вони відрізняються за способами атаки, механізмами поширення та наслідками для системи:

Способи атаки:

1. Троянські програми використовують соціальну інженерію для введення в оману користувачів та отримання доступу до їх систем.

2. Черв'яки вміють самостійно поширюватися через мережі, використовуючи вразливості у програмному забезпеченні або соціально інженерні методи.
3. Ransomware блокує доступ до файлів або систем шляхом їх шифрування та вимагає викуп за відновлення доступу.

Механізми поширення:

1. Троянські програми зазвичай поширюються через шахрайські посилання, вкладені файли електронних листів або вбудовуються в легітимне програмне забезпечення.
2. Черв'яки можуть поширюватися через мережі без втручання користувача, використовуючи вразливості у мережевому програмному забезпеченні.
3. Ransomware може поширюватися через спам-пошту, фішингові атаки або експлуатацію вразливостей у програмному забезпеченні.

Наслідки для системи:

1. Троянські програми можуть виконувати різноманітні шкідливі дії, такі як крадіжка особистої інформації, встановлення додаткового шкідливого ПЗ або надання віддаленого доступу кіберзлочинцям.
2. Черв'яки можуть споживати ресурси системи, поширювати інші шкідливі програми або навіть створювати мережеві ботнети.
3. Ransomware може блокувати доступ до важливих файлів або цілих систем, призводячи до фінансових втрат та порушень нормальної роботи [4].

Фішингові електронні листи - це вид атаки, коли кіберзлочинці намагаються використовувати соціальну інженерію, щоб введенням в оману користувачів отримати їхні конфіденційні дані, такі як паролі, номери кредитних карток або особисту інформацію. Фішингові листи часто виглядають як легітимні повідомлення від відомих організацій або осіб, таких як банки, платіжні системи, соціальні мережі або інші компанії.

Основна мета фішингових електронних листів - переконати отримувача у необхідності виконання певної дії, наприклад, натискання на посилання, завантаження вкладеного файлу або введення конфіденційної інформації на підроблені веб-сайти. Ці

листи можуть містити загрозливі посилання або вкладені файли, які містять шкідливе програмне забезпечення, таке як трояни або ransomware.

Одним із способів захисту від фішингових електронних листів є освіта користувачів та навчання їх розпізнавати підозрілі сигнали, такі як незвичайні запити на введення конфіденційної інформації, надмірна терміновість чи загрозливий тон повідомлення. Крім того, ефективні фільтри спаму та антивірусне програмне забезпечення можуть блокувати фішингові листи до того, як вони потраплять до скриньки отримувача.

Заражені веб-сайти - це веб-ресурси, які були скомпрометовані зловмисниками і містять шкідливий код або загрози безпеки. Вони можуть бути використані для розповсюдження шкідливого програмного забезпечення, включаючи троянські програми, черв'яки, ransomware та інші види малвару, а також для здійснення фішингових атак або інших видів кіберзлочинності.

Зазвичай заражені веб-сайти стають такими через експлуатацію вразливостей у веб-додатках або слабкість в конфігурації сервера. Наприклад, зловмисники можуть використовувати SQL-ін'єкції або вразливості в управлінні сеансами для внедрення шкідливого коду безпосередньо на веб-сайт. Також вони можуть використовувати атаки на веб-сервер або внедрювати шкідливий код через компрометовані сторонні ресурси, такі як віджети, плагіни або рекламні сценарії.

Після того, як веб-сайт стає зараженим, він може почати поширювати шкідливий код на комп'ютери відвідувачів, які взаємодіють із сайтом. Наприклад, заражений веб-сайт може використовувати вразливості в браузері або використовувати вишкребені дані автентифікації для встановлення троянських програм або крадіжки особистої інформації відвідувачів.

Захист від заражених веб-сайтів включає в себе регулярне оновлення веб-додатків і серверного програмного забезпечення, використання відповідних механізмів аутентифікації та авторизації, застосування фільтрів трафіку та використання надійних рішень забезпечення безпеки веб-додатків. Також важливо вести моніторинг безпеки веб-сайту для виявлення можливих атак та швидкого реагування на них [2].

Мережеві атаки - це атаки, спрямовані на комп'ютерні мережи з метою незаконного доступу до інформації, перешкоджання нормальному функціонуванню систем або викрадення конфіденційних даних. Ці атаки можуть бути спрямовані на різні рівні мережевої інфраструктури, включаючи локальні мережі, Інтернет та хмарні сервіси.

Перехоплення даних (Sniffing) - Атаки на перехоплення даних включають в себе моніторинг трафіку мережі з метою викрадення конфіденційної інформації, такої як паролі, номери кредитних карток або дані аутентифікації.

ARP Spoofing як атака включає в себе відправку фальшивих ARP-пакетів для маніпулювання таблицями маршрутизації в локальній мережі, що дозволяє зловмисникам перенаправляти мережевий трафік через свій комп'ютер, щоб перехопити або змінити дані.

DDoS (Distributed Denial of Service) - це атака, при якій зловмисники намагаються перевантажити цільовий сервер або мережу, спровокуючи велику кількість запитів або трафіку, що призводить до відмови в обслуговуванні для легітимних користувачів.

Фішинг (Phishing) - атаки використовують соціальну інженерію для отримання конфіденційної інформації, такої як паролі або особисті дані, шляхом використання підроблених веб-сайтів або електронних повідомлень.

Мережеві вразливості де зловмисники можуть використовувати вразливості в мережевих протоколах або програмному забезпеченні для отримання несанкціонованого доступу до систем або виконання атак в мережі [3].

Щоб захистити себе від мережевих атак, важливо використовувати надійні механізми аутентифікації та авторизації, шифрування трафіку, фаєрволи, інтригів і системи виявлення вторгнень (IDS) та системи запобігання вторгненням (IPS). Крім того, регулярне оновлення програмного забезпечення та мережевого обладнання може допомогти запобігти використанню вразливостей для атак.

1.2 Методи виявлення шкідливих файлів

Методи виявлення шкідливих файлів включають в себе різні підходи та техніки, які дозволяють ідентифікувати потенційно шкідливі файли або програми на комп'ютері або в мережі, логічно буде почати з найпоширеніщого методу.

Статичний аналіз - це метод виявлення шкідливих файлів, який використовує аналіз структури, змісту та властивостей файлу без його виконання або запуску. Основна ідея полягає в тому, щоб проаналізувати вміст файлу, включаючи його код, метадані та іншу інформацію, для виявлення ознак, які вказують на те, що файл може бути шкідливим [5].

Одним з основних методів статичного аналізу є використання сигнатур відомого шкідливого програмного забезпечення. Сигнатури - це унікальні характеристики або підписи, які ідентифікують конкретні шкідливі програми. Вони можуть включати рядок коду, хеш-суму, характеристики файлу або інші ознаки, які можуть бути однозначно пов'язані з певним видом шкідливого програмного забезпечення.

Під час статичного аналізу антивірусні програми чи інші засоби безпеки перевіряють вміст файлу на відповідність цим сигнатурам відомих загроз. Якщо файл містить хоча б одну з таких сигнатур, він вважається потенційно шкідливим і може бути помічений як загроза безпеки.

Важливою перевагою статичного аналізу є те, що він може бути швидким і ефективним способом виявлення шкідливого програмного забезпечення, оскільки він не вимагає виконання коду. Однак цей метод може бути обмежений у виявленні нових або маловідомих загроз, оскільки вони можуть не мати відповідних сигнатур.

Динамічний аналіз - це метод виявлення шкідливих програм, який полягає в запуску шкідливого ПЗ у контрольованому середовищі, такому як віртуальна машина або пісочниця, для спостереження за його поведінкою. Основна мета цього методу - виявити потенційно небезпечні дії програми під час виконання, такі як спроби зміни системних файлів, створення або видалення файлів, спроби встановлення зв'язку зі зловмисними серверами та інші аномальні дії [4].

Існують різні методики динамічного аналізу як от:

1. Віртуальні машини і пісочниці – цей підхід полягає в запуску програми у віртуальному середовищі, що імітує операційну систему. Віртуальні машини і пісочниці дозволяють ізолювати шкідливе ПЗ від основної системи, забезпечуючи безпеку та контрольоване спостереження за його поведінкою.
2. Інструменти моніторингу системи - де системи відстежують дії програми під час її виконання, включаючи зміни файлів, реєстраційні записи, мережевий трафік та інші системні події. Це допомагає виявити ненормальну або підозрілу поведінку.
3. Аналіз динамічного виконання коду – як метод включає в себе відстеження виконання коду шкідливого ПЗ та аналіз його дій в реальному часі. Це може включати моніторинг системних викликів, роботу з файлами, мережевий взаємодію та інші операції.
4. Евристичний аналіз - це метод виявлення потенційно шкідливої поведінки програми або файлу на основі алгоритмів та правил, які визначаються заздалегідь на основі набору характеристик або аномалій. Він використовується для пошуку загроз, які можуть не мати відомих сигнатур або зберігатися в криптованому вигляді [6].

Основні принципи евристичного аналізу включають створення правил і евристик, аналіз відмінностей, аналіз змін та використання методів машинного навчання та аналізу великих даних. При використанні цього методу розробники визначають правила або евристики, які вказують на певні типи потенційно шкідливої поведінки. Ці правила можуть базуватися на типових аномаліях, характеристиках або знаннях про відомі загрози.

Програми, які використовують евристичний аналіз, перевіряють дії програми на відповідність цим правилам. Якщо програма виявляє аномальну поведінку, яка не відповідає визначеним правилам, може бути виявлена підозра на потенційну загрозу. Крім того, програми, що використовують евристичний аналіз, відстежують зміни в системі або програмі, такі як створення нових файлів, модифікація системних

налаштувань, або спроби встановлення зв'язку зі зловмисними джерелами. Ці зміни можуть бути підставою для виявлення потенційно шкідливої діяльності [7].

Останні роки показали, що евристичний аналіз став ще ефективнішим завдяки використанню методів машинного навчання та аналізу великих даних. Вони дозволяють автоматизувати процес виявлення аномальної поведінки та вдосконалювати правила в залежності від нових загроз та відмінностей. Використання евристичного аналізу дозволяє ідентифікувати нові загрози, оскільки він не обмежується відомими сигнатурами та може виявляти невідомі або недетектовані раніше загрози на основі їх аномальної поведінки. Цей метод є важливим інструментом для забезпечення безпеки комп'ютерних систем та даних в умовах постійно змінюючогося кіберзлочинного ландшафту.

1.3 Сучасні програмні засоби захисту від шкідливих файлів

Антивірусні програми є важливою складовою комп'ютерної безпеки. Вони призначені для виявлення та видалення шкідливого програмного забезпечення (шкідливих програм), таких як віруси, черв'яки, троянські коні, різні варіації шпигунського ПЗ, ransomware та інші загрози. Основні характеристики антивірусних програм включають виявлення загроз, видалення загроз, оновлення вірусних баз, відсутність впливу на продуктивність системи та додаткові функції [9].

Антивірусні програми використовують різні методи для виявлення шкідливого ПЗ на комп'ютері. Це може бути базоване на сигнатурах виявлення (відомі підписи вірусів), евристичному аналізі (виявлення аномальної поведінки), поведінковому аналізу (спостереження за діями програм), аналізу репутації (оцінка безпеки за допомогою бази даних відомих джерел) та інші методи. Після виявлення шкідливого ПЗ антивірусна програма намагається видалити або усунути його з інфікованої системи. Цей процес може включати карантинування файлів, видалення шкідливих об'єктів, блокування доступу до заражених файлів та інші заходи безпеки.

Деякі антивірусні програми постійно оновлюють свої вірусні бази даних, щоб реагувати на нові та еволюціонуючі загрози. Це дозволяє антивірусним програмам

розпізнавати найновіші варіанти шкідливих програм та виявляти їх на комп'ютері. Деякі антивірусні програми розроблені з урахуванням низького впливу на продуктивність системи. Вони оптимізовані для ефективного виявлення та видалення шкідливого ПЗ, не сповільнюючи роботу комп'ютера або інших пристроїв.

Розвиток та адаптація антивірусних програм до сучасних загроз є критично важливими для забезпечення безпеки комп'ютерних систем. Швидкий та ефективний реагування на нові види загроз вимагає постійного вдосконалення технологій та методів, що використовуються антивірусними програмами.

Один із основних способів адаптації антивірусних програм полягає в постійному оновленні їх вірусних баз даних. Це дозволяє програмам розпізнавати нові варіанти вірусів та інших шкідливих програм, які постійно з'являються [10].

Багато сучасних антивірусних програм використовують методи штучного інтелекту та машинного навчання для виявлення та аналізу потенційно шкідливих об'єктів. Це дозволяє автоматизувати процес виявлення загроз та робить антивірусні програми більш ефективними в реагуванні на нові, раніше невідомі загрози.

Сучасні антивірусні програми все більше інтегруються з іншими засобами безпеки, такими як брандмауери, системи виявлення вторгнень та інші. Це дозволяє створювати комплексні заходи безпеки, які ефективно захищають систему від різних видів загроз.

З розвитком технологій з'являються нові формати загроз, такі як атаки на мобільні пристрої, Інтернет речей (IoT) та інші. Сучасні антивірусні програми активно вивчають ці нові формати загроз та розробляють методи їх виявлення та захисту.

Оновлення і адаптація антивірусних програм до сучасних загроз є важливими завданнями для забезпечення ефективного захисту комп'ютерних систем від шкідливого програмного забезпечення.

Системи виявлення та запобігання вторгненням (IDS/IPS) є ключовими складовими в системах безпеки мережі. IDS (Intrusion Detection System) та IPS (Intrusion Prevention System) аналізують мережевий трафік з метою виявлення та блокування потенційно шкідливої активності. IDS фокусується на виявленні можливих атак або

порушень безпеки, в той час як IPS може навіть автоматично реагувати на такі атаки, блокуючи або відхиляючи трафік.

Ці системи аналізують мережевий трафік шляхом моніторингу пакетів даних, що проходять через мережеві пристрої, такі як маршрутизатори або файерволи. Вони перевіряють цей трафік на наявність аномалій або відхилень від типового мережевого зв'язку, що може свідчити про атаку або іншу небезпечну активність.

IDS/IPS використовують різноманітні методи аналізу, такі як сигнатурний аналіз, що порівнює підписи відомих атак з мережевим трафіком для виявлення відповідних відхилень, та аналіз аномалій, який виявляє несподівані патерни чи поведінку, що може бути ознакою атаки.

Після виявлення підозрілого трафіку IDS може сповістити адміністратора мережі про потенційну загрозу, або навіть спробувати заблокувати чи відхилити цей трафік самостійно, якщо функціональність IPS активована. Такий підхід дозволяє швидко реагувати на загрози та мінімізувати ризик інцидентів безпеки в мережі [8].

Серед прикладів систем виявлення та запобігання вторгненням (IDS/IPS) можна виділити декілька популярних рішень.

Перш за все, Snort варто зазначити як одну з найпопулярніших відкритих систем IDS/IPS. Snort використовує сигнатурний аналіз для виявлення відомих атак та аналіз аномалій для виявлення нових загроз. Він може бути налаштований для реагування на виявлені загрози шляхом блокування трафіку або відправки сповіщень адміністраторам.

Другим прикладом є Suricata, також відкрита система IDS/IPS. Вона пропонує сигнатурний аналіз, аналіз аномалій та підтримку протоколів на основі правил для виявлення та запобігання атак.

Комерційним рішенням є Cisco Firepower, яке використовується для захисту мережі Cisco. Ця система IDS/IPS надає розширені можливості виявлення загроз та реагування на них, включаючи інтеграцію з іншими продуктами безпеки Cisco.

Ще одним прикладом є Palo Alto Networks Next-Generation Firewall (NGFW), який включає в себе не тільки функціональність файерволу, але також має вбудовану систему IDS/IPS для виявлення та блокування атак на мережу.

Ці системи надають організаціям можливість ефективно виявляти та запобігати широкому спектру кіберзагроз на мережевому рівні, що є ключовим для забезпечення безпеки мереж та даних користувачів.

Хмарні сервіси безпеки стають все більш популярними, оскільки вони надають користувачам ряд переваг у захисті від шкідливих файлів та кіберзагроз [11].

Одна з ключових переваг хмарних сервісів безпеки полягає в їх еластичності масштабування. Замість того, щоб обмежуватися обладнанням або ресурсами одного конкретного пристрою чи сервера, хмарні сервіси можуть розгортати захист у великих масштабах, щоб забезпечити безпеку всієї мережі чи інфраструктури користувача. Це особливо корисно для підприємств та організацій з великим обсягом даних та мережевими потоками, оскільки вони можуть ефективно масштабувати свої заходи захисту відповідно до потреб.

Оновлення в реальному часі є ще однією важливою функцією хмарних сервісів безпеки. Швидкість реакції на нові кіберзагрози та оновлення важлива для забезпечення надійного захисту. Хмарні сервіси можуть автоматично отримувати оновлення в реальному часі, щоб негайно реагувати на нові загрози та забезпечити високий рівень безпеки для користувачів [12].

Крім того, хмарні сервіси безпеки можуть використовувати колективний інтелект для виявлення та аналізу загроз. За допомогою штучного інтелекту та машинного навчання вони можуть аналізувати великі обсяги даних, щоб виявляти та передбачати нові кіберзагрози, використовуючи зібрані дані від користувачів по всьому світу. Це дозволяє надавати швидкі та точні оновлення та реакцію на найновіші загрози, що забезпечує більш ефективний захист від шкідливих файлів та кібератак.

Одним із прикладів хмарної платформи безпеки є Microsoft Defender for Cloud. Цей сервіс надає широкий спектр інструментів для виявлення, аналізу та захисту від кіберзагроз у хмарному середовищі. Microsoft Defender for Cloud використовує

штучний інтелект та аналіз великих обсягів даних для виявлення аномальної активності, вразливостей та потенційних загроз у хмарному середовищі користувача. Він надає засоби моніторингу, аудиту та автоматичного реагування на загрози, щоб забезпечити безпеку інфраструктури та даних користувача в реальному часі [13].

Іншим прикладом є Amazon Web Services (AWS) Security Hub, який є центром управління безпекою для хмарних сервісів AWS. AWS Security Hub автоматично аналізує дані безпеки з різних джерел, включаючи сервіси AWS, інші хмарні сервіси та сторонні інтеграції, для виявлення потенційних загроз та вразливостей. Він надає централізоване керування безпекою, аналіз інформації про загрози та інтеграцію з іншими інструментами безпеки для забезпечення повного захисту хмарної інфраструктури користувача.

Крім того, Cisco Umbrella є прикладом хмарного сервісу безпеки, який забезпечує захист від кіберзагроз для користувачів незалежно від їхнього місця роботи. Cisco Umbrella використовує глобальну мережу центрів обробки даних та інтелектуальні технології для виявлення та блокування шкідливих доменів, URL-адрес та IP-адрес, забезпечуючи захист на рівні мережі навіть поза корпоративною мережею.

Використання хмарних антивірусів та рішень ендпойнт захисту має низку переваг, які сприяють підвищенню рівня безпеки та ефективності управління безпекою в сучасних організаціях.

Однією з ключових переваг є централізоване управління та моніторинг. Це означає, що адміністратори мають можливість контролювати та відслідковувати безпекові події на всіх пристроях та в мережі з одного централізованого інтерфейсу. Це спрощує управління та дозволяє швидше реагувати на потенційні загрози.

Ще однією перевагою є швидкість реагування на загрози. Хмарні антивіруси та ендпойнт захист можуть отримувати оновлення та нові сигнатури миттєво, що дозволяє їм швидко реагувати на нові загрози та виявляти їх ще до того, як вони завдають шкоди системі.

Також слід зазначити, що хмарні рішення захисту забезпечують велику гнучкість та масштабованість. Вони можуть легко адаптуватися до змінних потреб організацій та масштабуватися відповідно до їхніх розмірів та потреб у безпеці [14].

Крім того, використання інтелектуального аналізу та машинного навчання дозволяє розпізнавати нові, раніше невідомі загрози та вчасно реагувати на них. Це робить такі рішення ефективними та надійними засобами захисту від широкого спектру кіберзагроз.

1.4 Висновки до розділу 1

У розділі 1 ми детально розглянули теоретичні основи виявлення та блокування шкідливих файлів. Розпочавши з загального визначення шкідливого програмного забезпечення, ми проаналізували його призначення та основні цілі. Зрозуміло, що шкідливе ПЗ створюється з метою завдати шкоди індивідуальним користувачам, організаціям або комп'ютерним системам шляхом різноманітних атак та методів.

Ми розглянули різноманітні типи шкідливого ПЗ, такі як віруси, троянці, черв'яки, шпигунське ПЗ та інші, і з'ясували, як кожен з них використовує різні методи атаки та має різний вплив на систему.

Підкреслили важливість розуміння різних типів шкідливого ПЗ та методів його поширення для розробки ефективних систем захисту. Також обговорили сучасні програмні засоби захисту, такі як антивірусні програми та системи виявлення та запобігання вторгненням, які грають важливу роль у боротьбі з кіберзагрозами [15].

Наголосили на необхідності постійного дослідження та розвитку в області кібербезпеки для виявлення та запобігання новим загрозам. Це є важливою передумовою для створення безпечного цифрового середовища для користувачів та організацій у сучасному інформаційному світі.

2 РОЗРОБКА ВЕБ-СТОРІНКИ ДЛЯ БЛОКУВАННЯ ШКІДЛИВИХ ФАЙЛІВ

2.1 Архітектура веб-сторінки

Архітектура веб-сторінки для блокування шкідливих файлів включає в себе вибір мов програмування, фреймворків та бібліотек, що надають необхідний функціонал та забезпечують ефективну реалізацію поставлених завдань.

Один із варіантів для розробки веб-сторінки - використання Angular та Node.js. Angular - це фронтенд фреймворк, який надає потужні інструменти для розробки користувацького інтерфейсу веб-додатків. Він дозволяє легко створювати високопродуктивні та масштабовані застосунки завдяки компонентній архітектурі та двосторонньому зв'язку даних.

Node.js - це середовище виконання JavaScript, яке дозволяє розробляти серверну частину веб-додатків. Використання Node.js дозволяє створювати швидкі та ефективні сервери за допомогою JavaScript, що спрощує розробку та підтримку веб-додатків.

Такий стек технологій дозволить забезпечити веб-сторінку для блокування шкідливих файлів із зручним та інтуїтивно зрозумілим інтерфейсом, швидкою відповіддю на запити користувачів та надійною захистом від потенційних кіберзагроз.

JavaScript є основною мовою програмування для веб-розробки та має безліч фреймворків і бібліотек, таких як React, Angular і Vue.js, які допомагають розробникам швидше та ефективніше створювати складні веб-додатки. Перевагами JavaScript є його широке використання та підтримка від усіх сучасних браузерів. Він забезпечує динамічну взаємодію на веб-сторінках та дозволяє створювати інтерактивні елементи, що полегшує взаємодію користувачів з веб-додатками [17].

React, Angular і Vue.js є популярними фреймворками для розробки користувацького інтерфейсу. React відомий своєю простотою та швидкістю розробки, відокремленням компонентів та високою продуктивністю. Angular надає повноцінний фреймворк для розробки веб-додатків, забезпечуючи інструменти для всіх аспектів

розробки, включаючи маршрутизацію та аутентифікацію. Vue.js відомий своєю легкістю вивчення та інтуїтивним API, що дозволяє швидше розробляти веб-додатки.

Проте, незважаючи на переваги, всі ці технології мають свої недоліки. Наприклад, JavaScript може мати проблеми з крос-браузерною сумісністю, а фреймворки, як React і Angular, можуть вимагати часу для вивчення та оволодіння їх концепціями. Крім того, велика кількість альтернативних фреймворків і бібліотек може призвести до плутанини та неоднозначності при виборі правильного рішення для конкретного проекту.

Python є однією з найпопулярніших мов програмування завдяки своїй простоті вивчення та потужним можливостям. Завдяки широкому спектру фреймворків, таких як Django і Flask, Python став першим вибором для веб-розробки, особливо для великих та складних проектів. Django є повноцінним фреймворком для швидкої розробки веб-додатків, який надає готові рішення для багатьох типових задач, таких як аутентифікація, маршрутизація та робота з базою даних. Він дозволяє розробникам ефективно створювати високоякісні веб-додатки з меншим обсягом коду.

Flask, з іншого боку, є легким та гнучким мікрофреймворком, який надає базовий набір інструментів для розробки веб-додатків. Він дає розробникам більшу свободу у виборі компонентів та бібліотек для використання, що дозволяє створювати більш настроєні та легкі веб-додатки. Flask є особливо корисним для прототипування додатків та створення API [16].

Обидва фреймворки мають ряд переваг, коли мова йде про роботу з великим обсягом даних і взаємодію з зовнішніми API. Django має вбудовану підтримку для роботи з базами даних, включаючи ORM (об'єктно-реляційне відображення), що полегшує роботу з великими обсягами даних. Flask, з іншого боку, дозволяє розробникам більш гнучко налаштовувати взаємодію з зовнішніми API, що робить його привабливим вибором для проектів, які потребують великої кількості зовнішніх інтеграцій.

Загалом, як Django, так і Flask надають потужні інструменти для роботи з великим обсягом даних та взаємодії з зовнішніми API, і вибір між ними залежить від конкретних потреб проекту та вподобань розробника.

Python є однією з найпопулярніших мов програмування завдяки своїй простоті вивчення та потужним можливостям. Завдяки широкому спектру фреймворків, таких як Django і Flask, Python став першим вибором для веб-розробки, особливо для великих та складних проектів. Django є повноцінним фреймворком для швидкої розробки веб-додатків, який надає готові рішення для багатьох типових задач, таких як аутентифікація, маршрутизація та робота з базою даних. Він дозволяє розробникам ефективно створювати високоякісні веб-додатки з меншим обсягом коду.

Flask, з іншого боку, є легким та гнучким мікрофреймворком, який надає базовий набір інструментів для розробки веб-додатків. Він дає розробникам більшу свободу у виборі компонентів та бібліотек для використання, що дозволяє створювати більш настроєні та легкі веб-додатки. Flask є особливо корисним для прототипування додатків та створення API.

Обидва фреймворки мають ряд переваг, коли мова йде про роботу з великим обсягом даних і взаємодію з зовнішніми API. Django має вбудовану підтримку для роботи з базами даних, включаючи ORM (об'єктно-реляційне відображення), що полегшує роботу з великими обсягами даних. Flask, з іншого боку, дозволяє розробникам більш гнучко налаштовувати взаємодію з зовнішніми API, що робить його привабливим вибором для проектів, які потребують великої кількості зовнішніх інтеграцій.

Структура веб-сторінки для динамічного виявлення та блокування шкідливих файлів під час завантаження повинна бути ретельно спроектована з урахуванням всіх аспектів її функціональності та безпеки. Перш за все, потрібно визначити клієнтську частину або фронтенд, яка буде відповідальна за інтерфейс користувача. На цьому рівні потрібно створити домашню сторінку, де користувачі зможуть завантажити файли для аналізу, а також форму завантаження для вибору файлів [18].

Крім того, на фронтенді необхідно реалізувати інтерфейс результатів, який буде відображати результати аналізу, такі як інформація про виявлені загрози та рекомендації щодо подальших дій. Наприклад, можна відображати список виявлених шкідливих файлів, їхні типи та ступінь небезпеки.

Далі, потрібно розробити бекенд або серверну частину веб-сторінки. Цей рівень буде відповідати за обробку запитів від клієнтів та виконання всіх необхідних операцій, пов'язаних з аналізом завантажених файлів. Сюди входить маршрутизація, функції обробки файлів, які включають в себе методи статичного, динамічного та евристичного аналізу, а також взаємодію з зовнішніми сервісами, такими як антивірусні движки та сервіси хмарної безпеки.

Крім того, слід врахувати інтерфейс користувача, який буде відповідати за сповіщення користувачів про статус обробки їхніх файлів та можливість подальших дій з ними. Наприклад, можна передбачити кнопки або опції для видалення, перенесення в карантин та інших дій з виявленими шкідливими файлами.

Усі ці компоненти повинні бути впроваджені з урахуванням найвищих стандартів безпеки, щоб запобігти можливим атакам та забезпечити конфіденційність та цілісність даних користувачів. Такий підхід до структури веб-сторінки для динамічного виявлення та блокування шкідливих файлів забезпечить ефективність, безпеку та зручність користувачів у взаємодії з веб-додатком.

Серверна частина, відома як бекенд, є ключовим компонентом веб-додатка для динамічного виявлення та блокування шкідливих файлів. Її головна функція полягає в обробці запитів від клієнтської сторони, виконанні необхідної логіки та забезпеченні взаємодії з базою даних та зовнішніми сервісами. Бекенд має бути ретельно спроектований та налагоджений, щоб забезпечити ефективну та безпечну роботу всього додатку.

Структура бекенду може бути організована за допомогою різних архітектурних підходів, таких як MVC або MVVM, і включає ряд ключових компонентів. Маршрутизація - один з таких компонентів, відповідальний за визначення шляхів запитів та їхніх обробників. Контролери обробляють запити та виконують відповідні дії, використовуючи моделі для роботи з даними. Сервіси можуть забезпечувати додаткову функціональність, наприклад, взаємодію з зовнішніми сервісами безпеки [20].

Одним з важливих аспектів роботи бекенду є забезпечення безпеки додатка. Для цього можуть використовуватися різноманітні заходи, такі як перевірка доступу, валідація даних, обробка винятків та застосування найкращих практик забезпечення безпеки програмного забезпечення.

Однією з головних переваг бекенду є можливість забезпечити потужність та масштабованість додатка, особливо в умовах великого обсягу даних. Використання відповідних технологій, таких як Python з фреймворками Django або Flask, дозволяє побудувати швидкий та надійний бекенд, який може ефективно обробляти великий обсяг запитів.

Однак при розробці бекенду слід враховувати його складність та можливу вразливість до атак. Тому необхідно вкласти достатній час та увагу у розробку безпечного та надійного бекенду, який буде забезпечувати ефективну роботу всього веб-додатка для динамічного виявлення та блокування шкідливих файлів.

Структура бази даних для веб-сторінки для динамічного виявлення та блокування шкідливих файлів грає ключову роль у зберіганні, організації та доступі до інформації, необхідної для функціонування додатку. Вона повинна бути ретельно спроектована з урахуванням вимог до безпеки, ефективності та розширюваності.

Перш за все, база даних повинна включати таблиці для зберігання інформації про користувачів, файлів та результатів аналізу. Наприклад, таблиця користувачів може містити поля для зберігання ідентифікатора користувача, логіну, хешованого пароля та іншої особистої інформації. Таблиця файлів може включати дані про шляхи до завантажених файлів, їхні розміри та час завантаження. Таблиця результатів аналізу може містити інформацію про виявлені шкідливі файли, їх типи та час аналізу.

Для забезпечення ефективної роботи додатка та швидкого доступу до даних, важливо правильно налаштувати індекси та відносини між таблицями. Наприклад, можна використовувати індекси для полів, які часто використовуються в запитах, щоб прискорити пошук та фільтрацію даних. Відносини між таблицями дозволяють забезпечити цілісність даних та зручний доступ до пов'язаних записів.

Безпека бази даних є ще одним важливим аспектом, який потрібно враховувати при проектуванні структури. Це включає в себе застосування методів шифрування для зберігання чутливої інформації, правильне налаштування доступу до даних за допомогою ролей та дозволів, а також регулярне оновлення та аудит бази даних для виявлення та усунення потенційних вразливостей [19].

Користувацький інтерфейс (UI) та користувацький досвід (UX) мають вирішальне значення у контексті веб-сайтів безпеки. Оскільки основною метою веб-сайтів безпеки є захист користувачів від шкідливих атак і забезпечення безпеки їхніх особистих даних, ефективний UI/UX може значно підвищити рівень захисту та сприяти успішному виявленню та блокуванню загроз.

На початку, чіткий та інтуїтивно зрозумілий користувацький інтерфейс (UI) може сприяти кращому розумінню користувачами функцій, які забезпечують безпеку. Він допомагає зробити важливі функції, такі як завантаження файлів для аналізу або налаштування параметрів безпеки, доступними та легкодоступними для користувачів будь-якого рівня експертизи.

Крім того, добре спроектований UI може сприяти своєчасному виявленню потенційних загроз. Наприклад, інтуїтивний інтерфейс для відображення статусу безпеки або сповіщень про підозрілу активність може допомогти користувачам реагувати на потенційні атаки швидко і ефективно.

З точки зору користувацького досвіду (UX), важливо, щоб веб-сайти безпеки були зручними та приємними у використанні. Чим менше зусиль вимагається від користувачів для здійснення безпечних дій, таких як завантаження файлів для аналізу або налаштування параметрів безпеки, тим більше вони будуть схильні до використання цих функцій. Правильне розташування елементів управління, зрозумілі інструкції та доступ до допомоги в разі потреби можуть значно поліпшити користувацький досвід та зробити процес використання веб-сайту безпеки більш приємним [21].

Отже, інвестування у відповідний користувацький інтерфейс (UI) та користувацький досвід (UX) для веб-сайтів безпеки може значно підвищити рівень

захищеності користувачів та сприяти ефективному виявленню та блокуванню шкідливих загроз.

Під час розробки веб-сторінки для динамічного виявлення та блокування шкідливих файлів, процес обробки завантажених користувачами файлів відіграє критичну роль у забезпеченні безпеки системи. Цей процес включає кілька етапів, починаючи з тимчасового зберігання файлів, продовжуючи скануванням на наявність шкідливого програмного забезпечення (ПЗ) та завершуючись видаленням файлів після їх аналізу.

На першому етапі, після того як користувач завантажує файл на веб-сторінку, цей файл тимчасово зберігається на сервері. Це може бути зроблено у спеціально відведеній директорії, призначеній для тимчасового зберігання файлів до їхньої подальшої обробки.

Далі, наступний етап - сканування файлів на наявність шкідливого ПЗ. Для цього можуть використовуватися різні методи виявлення, такі як статичний, динамічний або евристичний аналіз. У процесі цього сканування програмне забезпечення аналізує вміст файлу з використанням сигнатур, емуляції виконання або аналізу поведінки, щоб виявити будь-які загрози безпеки.

Після завершення аналізу файлу на предмет наявності шкідливого ПЗ, система приймає відповідне рішення щодо подальшої обробки файлу. Якщо файл виявлено як безпечний, його можна зберегти або надіслати користувачу. У випадку виявлення шкідливого ПЗ, файл може бути автоматично видалений або поміщений у карантин для подальшого аналізу адміністратором.

Після тимчасового зберігання та сканування файлів на наявність шкідливого ПЗ, сервер взаємодіє з аналітичними інструментами та базами даних для швидкого виявлення загроз. Ця взаємодія включає кілька кроків, які спрямовані на обробку та аналіз даних для ідентифікації потенційних загроз безпеки.

Перш за все, дані про сканування, виявлені загрози та інша відповідна інформація передаються на аналітичні інструменти, такі як системи виявлення вторгнень (IDS), системи управління подіями та інцидентами (SIEM) або спеціалізовані системи аналізу

великих даних (Big Data Analytics). Ці інструменти використовуються для глибокого аналізу отриманих даних, виявлення закономірностей та аномалій, які можуть вказувати на потенційні загрози.

Після аналізу інформації аналітичні інструменти можуть генерувати алерти або сповіщення про виявлені загрози та надсилати їх адміністраторам системи або відповідальним особам для подальшої обробки. Ці алерти можуть містити детальну інформацію про виявлені загрози, включаючи тип шкідливого ПЗ, методи атаки та потенційний вплив на систему [22].

Одночасно сервер взаємодіє з базами даних, де зберігається інформація про користувачів, їхні завантажені файли та історія взаємодії з веб-сторінкою. Ця інформація може бути корисною для подальшого аналізу та виявлення аномальної активності користувачів, яка може вказувати на можливі загрози.

Взаємодія з аналітичними інструментами та базами даних дозволяє швидко виявляти потенційні загрози та приймати відповідні заходи для захисту системи від них. Цей процес допомагає забезпечити безпеку веб-сторінки та захистити користувачів від шкідливого ПЗ.

2.2 Інтеграція систем виявлення шкідливих файлів

Інтеграція зовнішніх API для сканування файлів, таких як VirusTotal або інші хмарні сервіси, є важливою складовою розробки веб-сторінки для блокування шкідливих файлів. Цей процес дозволяє використовувати готові рішення для перевірки файлів на наявність шкідливого вмісту, що сприяє забезпеченню високої надійності та ефективності системи захисту [23].

Перший крок у процесі інтеграції - це реєстрація на веб-сайті постачальника API та отримання доступу до їхніх послуг. Ключ доступу, отриманий під час реєстрації, використовується для аутентифікації запитів, які відправляються до API для сканування файлів.

Після отримання доступу до API розробники можуть інтегрувати його функціонал у веб-сторінку. Це зазвичай включає створення коду, який відправляє

файли на сканування та обробляє результати. Використання HTTP-запитів для відправки файлів і отримання відповідей від API - це загальноприйнятий підхід.

Після отримання результатів сканування розробники можуть обробити їх і використовувати для прийняття рішень у системі. Наприклад, якщо файл виявлено як шкідливий, його можна автоматично заблокувати або перемістити до карантину для подальшого аналізу.

Для забезпечення ефективної роботи системи важливо постійно моніторити використання API та вчасно реагувати на будь-які проблеми, такі як обмеження або відмови у послугах. Розробники також можуть розглядати можливість використання резервних API або розподілення навантаження між кількома постачальниками, щоб забезпечити неперервну роботу системи.

При інтеграції зовнішніх API для сканування файлів на шкідливість, важливо реалізувати ефективні механізми автентифікації, обмеження запитів і обробки результатів сканування. Ці кроки забезпечують безпеку, ефективність і стабільність роботи системи.

Автентифікація є першим кроком для забезпечення безпеки інтеграції зовнішнього API. Розробники повинні коректно і безпечно зберігати та використовувати ключі доступу, що надаються постачальником API під час реєстрації. Ключі повинні бути оброблені з великою увагою до деталей і ніколи не повинні передаватися або зберігатися у відкритому вигляді.

Обмеження запитів є важливим аспектом ефективної роботи системи. Розробники повинні ретельно вивчити ліміти та обмеження, накладені постачальником API, і розробити стратегію керування запитами для запобігання перевантаження та забезпечення надійної роботи. Це може включати встановлення обмежень на кількість запитів за певний часовий період або використання черг для розподілу навантаження.

Обробка результатів сканування є останнім кроком у процесі інтеграції API. Розробники повинні правильно інтерпретувати результати, які надходять від API, і вживати відповідних заходів у випадку виявлення шкідливого вмісту. Це може включати блокування доступу до файлу, сповіщення адміністратора системи, або

виконання додаткових заходів безпеки для запобігання подальшій розповсюдженню шкідливого ПЗ. Також важливо враховувати можливі ложнопозитивні результати сканування і розробляти механізми для їхньої обробки і управління [24].

Інтеграція скануючих модулів безпосередньо у веб-сторінку для перевірки файлів перед їхнім відправленням на сервер є важливим кроком для забезпечення безпеки користувачів. Цей процес може бути реалізований за допомогою JavaScript, що дозволяє виконувати дії на стороні клієнта, без необхідності відправляти файли на сервер для аналізу.

Інтеграція скануючих модулів на стороні клієнта зазвичай включає використання JavaScript API, яке надається постачальниками антивірусного програмного забезпечення або альтернативними сервісами сканування файлів. Ці API дозволяють виконати сканування файлу безпосередньо на пристрої користувача перед його завантаженням на сервер.

Перший крок - це вибір відповідного API або бібліотеки для виконання сканування файлів на стороні клієнта. Після цього розробники можуть інтегрувати це API у веб-сторінку шляхом включення відповідних JavaScript-файлів у код сторінки.

Після інтеграції API розробники можуть створити форму для вибору файлів для завантаження. Після вибору файлу користувачем, скануючий модуль може виконати перевірку файлу на наявність шкідливого вмісту за допомогою API. Результати сканування можуть бути відображені користувачеві у вигляді повідомлення або індикатора безпеки.

Інтеграція скануючих модулів на стороні клієнта дозволяє забезпечити швидку та ефективну перевірку файлів перед їхнім завантаженням на сервер. Цей підхід дозволяє захистити користувачів від шкідливих файлів ще до того, як вони потраплять на сервер, зменшуючи ризик виникнення загроз для безпеки даних.

Використання JavaScript для впровадження антивірусного сканування на клієнтській стороні є важливим кроком у забезпеченні безпеки веб-сторінок. Цей підхід дозволяє виявляти потенційно шкідливі файли ще до їх завантаження на сервер, що допомагає попередити можливі загрози безпеці даних та систем. Однією з основних

переваг використання JavaScript для антивірусного сканування є швидкість і простота інтеграції. JavaScript може бути легко включений у веб-сторінку без потреби встановлення додаткових програм або модулів на сервері. Це дозволяє розробникам швидко і ефективно забезпечити додатковий захист для користувачів.

Ще однією перевагою є те, що антивірусне сканування на клієнтській стороні допомагає зменшити навантаження на сервер. Лише безпечні файли будуть відправлені на сервер для подальшої обробки, що дозволяє зберегти ресурси сервера та зменшити його витрати на обслуговування. Крім того, цей підхід сприяє збереженню приватності даних, оскільки файли не відправляються на сервер для аналізу. Це може бути важливо для користувачів, які прагнуть зберегти конфіденційність своїх даних та уникнути можливих проблем з їх захистом.

Однак важливо враховувати обмеження антивірусного сканування на клієнтській стороні. Наприклад, доступність антивірусних баз даних та обмеження безпеки браузера можуть обмежувати ефективність такого сканування. Крім того, JavaScript може бути обманутим зловмисниками або вимкненим у налаштуваннях браузера, що може підірвати його ефективність як засобу захисту.

У всіх випадках важливо ретельно оцінювати потреби вашого проекту та вибирати найбільш підходящий метод захисту. Використання JavaScript для антивірусного сканування на клієнтській стороні може бути ефективним рішенням для деяких веб-додатків, але в інших випадках може бути доцільніше використовувати інші методи захисту.

Використання WebAssembly для впровадження антивірусного сканування на клієнтській стороні може бути важливим кроком у забезпеченні безпеки веб-сторінок. WebAssembly (або Wasm) - це відкритий стандарт, який дозволяє виконувати веб-додатки, написані на різних мовах програмування, таких як C ++, Rust, а також мови складення, наприклад, LLVM, безпосередньо в браузері з практично нульовим впливом на продуктивність [25].

Однією з ключових переваг використання WebAssembly є можливість виконання високопродуктивного коду, що може бути ефективно використано для антивірусного

сканування без навантаження на сервер. Це дозволяє виявляти потенційно шкідливі файли ще до їх завантаження на сервер, зменшуючи ризик інфікування системи користувача. Крім того, використання WebAssembly дозволяє виконувати сканування у реальному часі, що дозволяє виявляти загрози миттєво під час завантаження файлів.

Ще однією перевагою є те, що WebAssembly може бути легко включений у веб-сторінку без потреби встановлення додаткових програм або модулів на сервері. Це спрощує розробку та впровадження антивірусного сканування на веб-сторінках. Крім того, WebAssembly забезпечує відокремлене середовище виконання, що допомагає забезпечити безпеку виконання ненадійного коду без ризику для інших частин веб-сторінки або браузера (рис. 2.1).

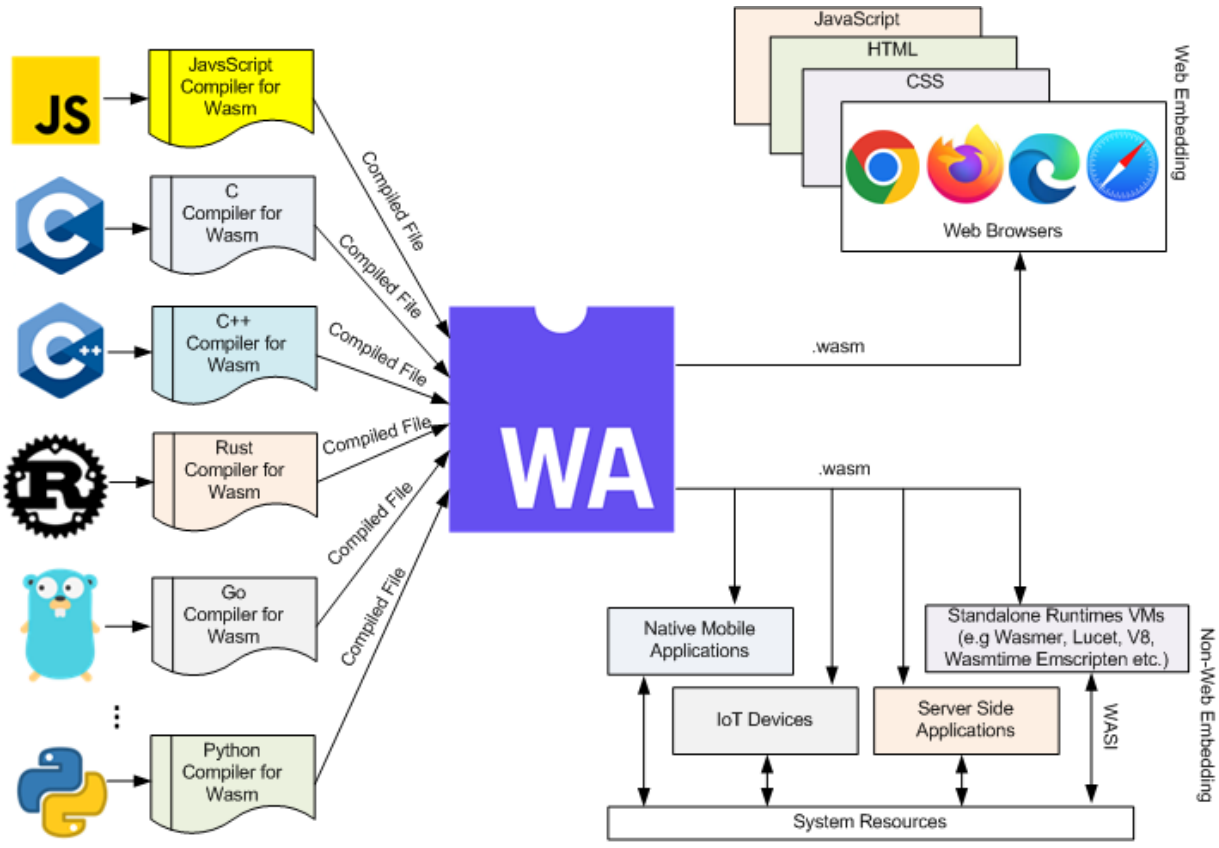


Рис. 2.1 Візуалізація функціональності WebAssembly

Проте важливо враховувати, що WebAssembly має свої обмеження і може не бути підходящим для всіх сценаріїв використання. Наприклад, обмежена підтримка деяких функцій і бібліотек може обмежити можливості антивірусного сканування. Крім того, необхідно враховувати можливість обхідних атак, які можуть використовувати

зловмисники для обману або ухилення від сканування. Тому важливо ретельно розглядати всі переваги і недоліки WebAssembly перед використанням його для антивірусного сканування на веб-сторінках.

Використання хмарних платформ для зберігання і постійного оновлення сигнатур вірусів може бути корисним рішенням для забезпечення ефективності антивірусного захисту. Хмарні платформи надають інфраструктуру для зберігання великих обсягів даних та масштабування, що дозволяє зберігати велику кількість сигнатур вірусів і оновлювати їх у реальному часі.

Однією з головних переваг використання хмарних платформ є їхня доступність та надійність. Ці платформи мають велику кількість серверів і резервні копії даних, що забезпечує безперебійну доступність і захист від втрати інформації. Крім того, хмарні платформи можуть швидко реагувати на нові загрози та оновлення сигнатур вірусів, оскільки вони мають потужні обчислювальні ресурси та автоматизовані процеси оновлення.

Іншою перевагою є гнучкість і масштабованість хмарних платформ. Вони можуть легко адаптуватися до зростаючих потреб користувачів і обсягів даних, що дозволяє забезпечувати ефективний захист навіть при великих обсягах файлів і користувачів. Крім того, завдяки глобальній розгортці серверів, хмарні платформи можуть забезпечити швидкий доступ до сигнатур вірусів для користувачів з будь-якої точки світу [32].

Проте важливо враховувати деякі питання щодо конфіденційності та безпеки даних при використанні хмарних платформ. Оскільки дані зберігаються на серверах постачальників хмарних послуг, існує ризик несанкціонованого доступу або порушення конфіденційності. Тому важливо вибирати надійних постачальників, які дотримуються високих стандартів безпеки та конфіденційності даних.

Узагальнюючи, використання хмарних платформ для зберігання і оновлення сигнатур вірусів може значно підвищити ефективність антивірусного захисту, забезпечуючи доступність, масштабованість та швидкість реакції на нові загрози.

Однак необхідно враховувати питання безпеки та конфіденційності даних при виборі постачальника хмарних послуг.

Можливість автоматичного оновлення системи є надзвичайно важливою для забезпечення безпеки та надійності веб-сторінки для виявлення та блокування шкідливих файлів. Ця функціональність дозволяє системі автоматично отримувати оновлення та патчі безпеки, що забезпечує її актуальність та готовність до виявлення нових загроз. Автоматичне оновлення також знижує ризик виникнення вразливостей, оскільки воно швидко впроваджує необхідні поправки та виправлення безпеки.

Переваги автоматичного оновлення включають забезпечення безпеки системи шляхом автоматичного встановлення оновлень безпеки та виправлень помилок. Це дозволяє підтримувати систему в актуальному стані та запобігати виникненню проблем, пов'язаних з вразливостями. Крім того, автоматичне оновлення звільняє адміністраторів від необхідності ручного встановлення оновлень, зменшуючи час і зусилля, потрібні для підтримки системи.

Однак автоматичне оновлення також може мати свої недоліки. Наприклад, недолік може виникнути, якщо оновлення призведе до конфліктів між різними компонентами системи або викличе непередбачувану поведінку. Також існує ризик, що після оновлення може виникнути проблема з функціональністю або сумісністю з іншими програмами або сервісами.

Машинне навчання відіграє ключову роль у виявленні шкідливих файлів, оскільки це дозволяє аналізувати великі обсяги даних та виявляти в них закономірності, які можуть вказувати на наявність шкідливих або підозрілих об'єктів. Інтеграція моделей машинного навчання дозволяє розробляти алгоритми, які автоматично виявляють та класифікують шкідливі файли без необхідності вручного втручання [31].

Однією з основних переваг використання машинного навчання є здатність моделей адаптуватися до нових типів загроз та змінюваного ландшафту кібербезпеки. Моделі можуть навчатися на основі актуальних даних та виявляти нові шаблони атак, які не були раніше відомі. Це дозволяє постійно підтримувати високий рівень ефективності виявлення навіть у змінних та динамічних середовищах кіберзагроз.

Для використання машинного навчання у виявленні шкідливих файлів потрібно спочатку підготувати навчальні дані, які включають в себе як позитивні, так і негативні приклади шкідливих файлів. Потім можна використовувати різноманітні алгоритми машинного навчання, такі як класифікаційні моделі, нейронні мережі та кластеризаційні методи, для створення моделей, які здатні автоматично виявляти та класифікувати нові файли.

Процес навчання моделей машинного навчання для виявлення шкідливих файлів включає кілька етапів, що спрямовані на підготовку та оптимізацію алгоритмів для ефективного виявлення потенційних загроз. Першим кроком є підготовка навчального набору даних, який містить як позитивні (шкідливі файли), так і негативні (нешкідливі файли) приклади. Ці дані потім розділяються на тренувальний та тестовий набори для оцінки точності моделі.

Далі проводиться навчання моделі на тренувальному наборі, підбираються оптимальні параметри та алгоритми для досягнення найвищої точності. Після навчання модель перевіряється на тестовому наборі для оцінки її продуктивності та валідності. У разі необхідності параметри моделі можуть бути оптимізовані або викориговані.

Після успішного навчання та валідації моделі вона готова до впровадження в реальний час для аналізу файлів, завантажених на веб-сторінку. Модель може бути інтегрована у серверну частину веб-сторінки та використовуватися для автоматичного виявлення шкідливих файлів під час їх завантаження [32].

Після впровадження моделі в реальний час важливо постійно відслідковувати її продуктивність та ефективність, оновлюючи її при необхідності та адаптуючи до нових типів шкідливих програм. Також важливо забезпечити механізми моніторингу та реагування на випадки, коли модель не виявляє нові загрози або надто часто спрацьовує на фальшиві позитиви, що дозволяє підтримувати високий рівень захисту веб-сторінки від шкідливих файлів.

Інтерфейс користувача (UI) та звітність є важливою частиною веб-сторінки для виявлення та блокування шкідливих файлів. Добре спроектований UI дозволяє

користувачам зручно взаємодіяти з системою та отримувати необхідну інформацію про стан сканування їх файлів та його результати.

Одним з ключових елементів інтерфейсу є панель керування, де користувач може завантажити файли для сканування та бачити статус сканування. Наприклад, на цій панелі можуть бути розміщені кнопки для завантаження файлів, індикатори прогресу сканування та кнопка для перегляду детальної інформації про результати.

Для представлення результатів сканування може використовуватися таблиця, де кожен рядок відповідає завантаженому файлу, а стовпці відображають різні параметри, такі як назва файлу, час сканування, статус (чи виявлено шкідливе ПЗ), тип загрози та інші важливі дані [30].

Крім того, важливо забезпечити можливість фільтрації та сортування результатів для зручності користувача. Наприклад, користувач може швидко знайти файли, які мають певний тип загрози або переглянути найбільш недавно скановані файли.

У звітній частині також можуть бути відображені загальні статистичні дані, такі як кількість завантажених файлів, кількість виявлених загроз, час останнього сканування та інші показники, що допомагають користувачеві зрозуміти стан безпеки його системи.

Розробка інтуїтивно зрозумілих UI компонентів, таких як індикатори прогресу, вимагає уваги до деталей та розуміння потреб користувача. Ось кілька ключових аспектів, які варто врахувати під час їх розробки:

Індикатор прогресу повинен чітко вказувати стан завдання, щоб користувач міг зрозуміти, скільки часу залишилося або на якому етапі він знаходиться.

Використання анімації може зробити індикатор прогресу більш привабливим та інтуїтивно зрозумілим. Наприклад, анімація заповнення індикатора зліва направо або обертання індикатора може надати візуального контексту процесу [29].

Використання різних кольорів може допомогти візуально виділити різні стани або типи завдань. Наприклад, червоний колір може вказувати на помилку, зелений - на успіх, а жовтий - на процес виконання.

Індикатор прогресу повинен бути адаптивним до різних розмірів екрану та пристроїв, щоб забезпечити коректне відображення на всіх пристроях, включаючи мобільні.

У прикладі функція updateProgressBar (рис. 2.2) оновлює ширину прогрес-бару відповідно до вказаного прогресу. Додавання анімації та кольорової кодифікації може бути реалізоване за допомогою CSS або JavaScript, залежно від ваших потреб та вимог дизайну.

```
<div>
  <div id="progress-container">
    <div id="progress-bar"></div>
  </div>

  <h1>File Analyzer</h1>
  <input type="file" (change)="onFileSelected($event)">
</div>
```

```
  updateProgressBar(progress: number) {
    const progressBar = document.getElementById('progress-bar') as HTMLElement;
    progressBar.style.width = progress + '%';
  }
}
```

Рис. 2.2 Приклад коду на JavaScript для створення простого індикатора прогресу

Щоб надати користувачам можливість отримувати сповіщення про результати сканування та управляти знайденими загрозами, можна використовувати модальні вікна або сповіщення в реальному часі. Ось кілька ключових елементів коду для цього:

Використання модальних вікон може дозволити користувачам отримувати докладну інформацію про результати сканування та вибирати опції для кожного знайденого файлу.

```
<div class="upload">
  <h1>File Analyzer</h1>
  <input type="file" (change)="onFileSelected($event)">
</div>

<mat-card class="popup" *ngIf="isPopup" (click)="this.isPopup = false">
  <div class="popup_container">
    <mat-card-header id="progress-container">
      <div id="progress-bar"></div>
    </mat-card-header>
    <mat-card-content class="reports" *ngIf="reports; else loading">
      <div class="reports_block">
        <h2 class="reports_title">Reports for File {{ fileName }}</h2>
        <div>
          <p>Detected: {{ detectedCount }}</p>
          <p>Not Detected: {{ notDetectedCount }}</p>
        </div>
      </div>
      <mat-list class="reports_lists">
        <mat-list-item class="list" *ngFor="let key of reportsTitle">
          <nav>{{ key }} -</nav>
          <nav [ngClass]="{'active': reports[key].detected}">{{ reports[key].detected ? 'Detected' : 'Not Detected' }}
          </nav>
        </mat-list-item>
      </mat-list>
    </mat-card-content>
    <mat-card-actions class="actions">
      <button mat-button class="close" (click)="removeFile()">Remove</button>
      <button mat-button class="confirmed" (click)="confirmeFile()">Confirmed</button>
    </mat-card-actions>

    <ng-template #loading>
      <p>Loading...</p>
    </ng-template>
  </div>

</mat-card>
```

Рис. 2.3 Створення функцій JavaScript для обробки опцій кожного файлу, таких як видалення або ігнорування попередження.

Функції на рис. 2.4 та рис. 2.5 можна викликати при натисканні на відповідні кнопки в модальному вікні. Вони можуть взаємодіяти з сервером за допомогою AJAX-запитів для виконання потрібних дій з файлами.

```
public removeFile() {
  const input = document.querySelector('input[type="file"]') as HTMLInputElement;
  if (input) {
    input.value = '';
  }
}
```

Рис. 2.4 Код для видалення файлу

```
public confirmeFile() {
  this.router.navigate(['/admin/added/result'], {queryParamsHandling: 'merge'}).then();
}
```

Рис. 2.5 Код для підтвердження аналізу файлів

2.3. Тестування та оптимізація веб-сторінки

Оптимізація та тестування веб-сторінки є критично важливими етапами розробки, які дозволяють покращити продуктивність, безпеку та користувацький досвід. Під час оптимізації веб-сторінки важливо враховувати різні аспекти, такі як завантаження ресурсів, швидкість завантаження сторінки, ефективність запитів до сервера та безпека.

Одним з ключових методів оптимізації є мінімізація та об'єднання файлів CSS та JavaScript, а також стискання зображень та інших медіа-ресурсів для зменшення їх розміру. Додатковою стратегією є використання кешування, що дозволяє зберігати копії сторінок або ресурсів на клієнтському пристрої для швидкого доступу під час наступних відвідувань. Інтеграція з CDN також може допомогти покращити швидкість завантаження, розподіляючи ресурси на серверах, розташованих ближче до кінцевих користувачів.

Тестування безпеки веб-сторінки включає в себе ручне та автоматизоване сканування на вразливості, пенетраційне тестування та тестування на відмову. Це допомагає виявляти та усувати потенційні проблеми безпеки, такі як кросс-сайтовий скриптінг (XSS), SQL-ін'єкції, а також захист від атак переповнення буфера та інших загроз. Крім того, регулярне оновлення системи та її складових, виявлення та виправлення вразливостей, а також моніторинг безпеки допомагають забезпечити стійкість до потенційних атак [29].

Основні типи тестування безпеки веб-сторінок включають ручне тестування, автоматизоване сканування вразливостей, пенетраційне тестування та тестування на відмову. Розглянемо інструменти та техніки, які використовуються для кожного з цих типів тестування:

2.3.1 Ручне тестування

Зазначені техніки тестування безпеки веб-сторінок відіграють критичну роль у забезпеченні безпеки та надійності систем. Під час інспектування вихідного коду розробники детально переглядають HTML, CSS та JavaScript-код сторінки, аналізуючи його на предмет можливих вразливостей. Вони шукають ознаки можливих атак, таких як міжсайтовий скриптинг (XSS) або SQL-ін'єкції, та вживають заходів для їх

запобігання, таких як екранування введених даних та використання параметризованих запитів до бази даних.

Тестування форм та введення даних дозволяє розробникам перевірити вразливості, пов'язані зі специфічними типами введених даних. Вони вводять спеціально сформовані дані в форми для виявлення потенційних вразливостей, таких як XSS або міжсайтова подделка запиту (CSRF). Це дозволяє перевірити, чи належним чином обробляються та екрануються дані, що вводяться користувачем, а також чи застосовані належні заходи безпеки для запобігання атакам.

Тестування автентифікації та авторизації включає спроби входу з неправильними обліковими даними, спроби доступу до захищених ресурсів без належних дозволів, а також перевірку правильності обробки сесійних та куки-даних. Це дозволяє виявити можливі слабкі місця у механізмах автентифікації та авторизації та прийняти відповідні заходи для їх поліпшення.

2.3.2 Автоматизоване сканування вразливостей

OWASP ZAP (Zed Attack Proxy) є відкритим програмним забезпеченням, яке надає широкий набір можливостей для автоматизованого сканування веб-додатків на наявність різних вразливостей. Ці вразливості можуть включати міжсайтовий скриптинг (XSS), SQL-ін'єкції, міжсайтову подделку запиту (CSRF) та інші. OWASP ZAP дозволяє виявляти ці вразливості шляхом активного тестування веб-додатків та надає детальні звіти про знайдені проблеми безпеки [24].

Burp Suite є комплексним набором інструментів для тестування безпеки веб-додатків. Він включає в себе сканер вразливостей, який автоматично перевіряє веб-додатки на наявність різних видів вразливостей. Крім того, Burp Suite має перехоплювач HTTP-трафіку, який дозволяє аналізувати та модифікувати запити та відповіді між клієнтом і сервером, щоб виявити потенційні вразливості та випробувати різні атаки на веб-додатки.

Nikto є інструментом для виявлення відомих вразливостей у веб-серверах та веб-додатках. Він автоматично сканує веб-сайти на предмет різних видів вразливостей, таких як незахищені каталоги, відомі вразливості програмного забезпечення та інші.

Nikto надає користувачеві звіти про знайдені проблеми безпеки, що дозволяє розробникам вживати заходів для їх усунення.

2.3.2 Пенетраційне тестування

Пенетраційне тестування - це процес, під час якого спеціалізовані фахівці (пенетратори) спробують проникнути в інформаційну систему з метою виявлення та використання вразливостей, які можуть бути використані зловмисниками. Для цього вони використовують різноманітні інструменти та техніки, що дозволяють їм проникнути в систему та отримати доступ до конфіденційної інформації або здійснити інші недозволені дії.

Один із найбільш відомих інструментів для пенетраційного тестування - це Metasploit (рис. 2.6). Це відкритий фреймворк, який надає широкі можливості для експлуатації вразливостей та отримання доступу до системи. Metasploit містить більше тисячі експлоітів, модулів та інструментів, які допомагають пенетраторам здійснювати різноманітні атаки на інформаційні системи з метою виявлення потенційних вразливостей.

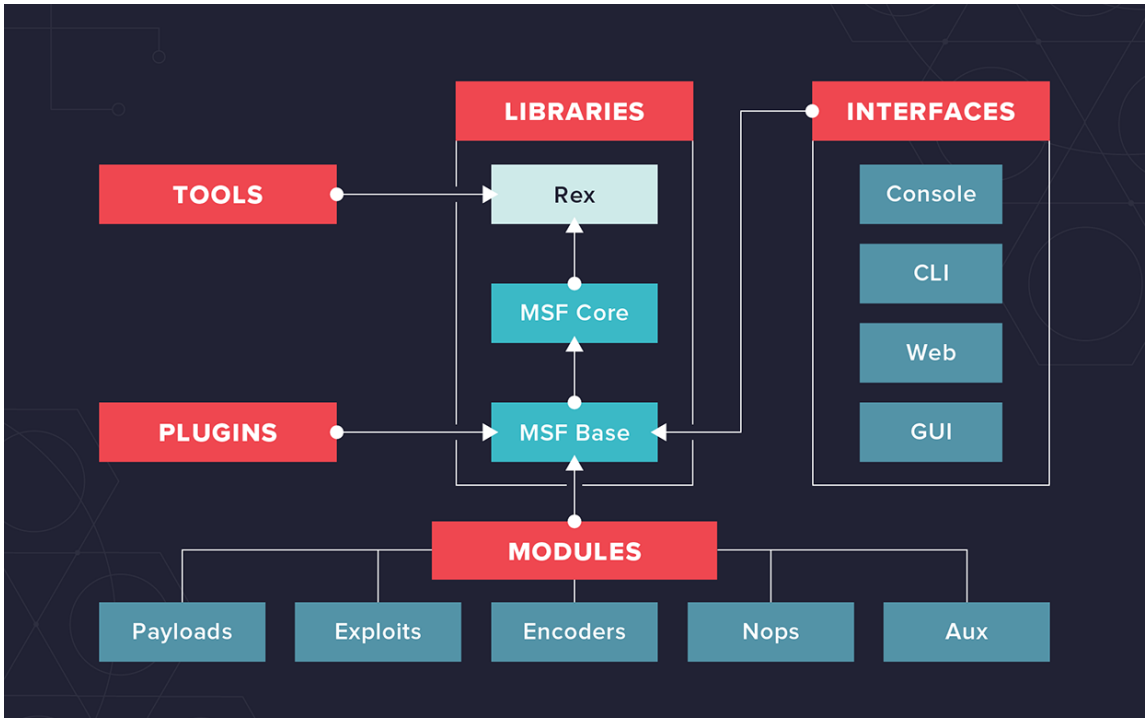


Рис. 2.6 Архітектура Metasploit

Ще одним важливим інструментом для пенетраційного тестування є Nmap. Це інструмент для сканування мережі, який дозволяє виявляти активні хости та відкриті порти. Використання Nmap допомагає пенетраторам ідентифікувати потенційні точки входу для атак та визначити, які сервіси працюють на цих хостах. Це дозволяє пенетраторам виявити слабкі місця в мережевій інфраструктурі та скерувати свої зусилля на їх експлуатацію.

Тестування на відмову (або навантаження) - це процес, під час якого випробовується витривалість системи під великим навантаженням, щоб визначити, як вона реагує на інтенсивне використання та чи відповідає вона очікуваним стандартам продуктивності. Для цього використовуються спеціальні інструменти, які генерують велику кількість запитів до сервера або додатку, створюючи таким чином велике навантаження [24].

Один із таких інструментів - Apache JMeter. Це інструмент для проведення стрес-тестування та тестування на відмову, який дозволяє моделювати різні сценарії навантаження на сервер та визначати його межі витривалості. Використання Apache JMeter дозволяє провести широкий спектр тестів на відмову, від простих до складних, та отримати детальну інформацію про продуктивність системи під час інтенсивного навантаження.

Ще одним інструментом для тестування на відмову є Siege. Цей інструмент також генерує велику кількість запитів до сервера, але він робить це за допомогою простого та легкого в користуванні інтерфейсу командного рядка. Siege дозволяє створювати різні сценарії навантаження та визначати, як система реагує на це навантаження. Він є швидким та ефективним інструментом для тестування на відмову, що дозволяє виявити потенційні проблеми з продуктивністю системи та вчасно вирішити їх.

Процес пенетраційного тестування починається з планування, в якому визначаються цілі тестування, обсяг робіт, методи і інструменти, що будуть використані. Після цього проводиться збір інформації про цільову систему, включаючи виявлення портів, служб, програмного забезпечення та іншої інформації, що може бути використана для атаки. Наступним кроком є сканування та виявлення вразливостей, в

ході якого застосовуються різні інструменти, такі як сканери портів та вразливостей, для пошуку слабких місць у системі. Після виявлення вразливостей проводиться експлуатація, коли тестируються різні методи атаки, щоб перевірити, чи можливо скористатися вразливостями для незаконного доступу до системи або отримання конфіденційної інформації. В результаті аналізу отриманих даних і результатів тестування розробляється звіт, в якому детально описуються знайдені вразливості та рекомендації щодо їх виправлення.

Пенетраційне тестування дозволяє ідентифікувати потенційні слабкі місця в архітектурі веб-сторінки та мережевій інфраструктурі, такі як недостатній контроль доступу, незахищені мережеві сервіси, слабка автентифікація та інші вразливості, які можуть бути використані для атаки. Цей тип тестування допомагає зміцнити захист веб-сторінки шляхом виявлення та виправлення вразливостей перед тим, як вони можуть бути використані зловмисниками. Таким чином, пенетраційне тестування є важливим етапом в забезпеченні безпеки веб-сторінки і захисту від потенційних кібератак [25].

Розглянувши різні типи тестування безпеки, варто визначити їхню важливість та призначення для забезпечення надійного рівня захисту веб-сайтів та додатків.

Ручне тестування забезпечує можливість здійснення детального аналізу коду та інфраструктури додатка людиною, що дозволяє виявляти складні та контекстуальні вразливості, які можуть бути пропущені автоматизованими інструментами. Це дозволяє здійснити повний перехід по всім можливим шляхам атаки та визначити найбільш критичні вразливі місця, які потребують уваги.

Автоматизоване сканування вразливостей, зокрема інструменти, які були вказані, дозволяють швидко та ефективно виявляти загрози безпеки, такі як SQL-ін'єкції, XSS атаки, недоліки в конфігурації сервера та інші. Вони допомагають виявити типові проблеми безпеки шляхом автоматизованого аналізу великої кількості коду та інфраструктури, що значно зменшує час, необхідний для виявлення вразливостей та їх виправлення.

Пенетраційне тестування дозволяє моделювати реальні атаки на систему з метою виявлення слабких місць у захисті. Воно включає в себе використання різних методів

атаки, інструментів та технік, щоб перевірити стійкість системи до різних видів загроз. Пенетраційні тести дозволяють виявити більш складні та контекстуальні вразливості, які можуть бути пропущені іншими методами тестування.

Тестування на відмову використовується для оцінки стійкості системи до навантаження та атак з боку зловмисників. Воно дозволяє визначити, як система реагує на велику кількість запитів або спроби змусити її відмовити. Таке тестування важливо для забезпечення високої доступності та стійкості системи у найрізноманітніших умовах експлуатації.

Ручне тестування зазвичай є ефективним у виявленні стандартних вразливостей, таких як ін'єкції SQL, XSS, CSRF та інші, оскільки цей підхід дозволяє тестувальникам вручну перевіряти кожен аспект веб-додатка або сайту. Ручне тестування забезпечує можливість виявити контекстуальні вразливості та використовувати творчий підхід для пошуку потенційних загроз безпеки, що можуть бути пропущені автоматизованими інструментами. Однак ручне тестування може бути витратним за часом і не завжди ефективним у виявленні всіх можливих вразливостей.

Автоматизовані інструменти для сканування вразливостей, такі як OWASP ZAP, Nessus, Burp Suite та інші, також є ефективними у виявленні стандартних вразливостей. Вони використовують заздалегідь визначені правила та алгоритми для пошуку паттернів або ознак, що вказують на наявність вразливостей. Ці інструменти досить ефективно виявляють типові проблеми безпеки, такі як ін'єкції SQL, XSS, CSRF, оскільки вони можуть автоматично перевіряти велику кількість коду та інфраструктури. Вони дозволяють швидко сканувати та виявляти вразливості у великих проектах з великою кількістю коду та сторінок.

Пенетраційне тестування, також використовується для виявлення стандартних вразливостей, але його основна мета - це спроба проникнути в систему, емулюючи дії зловмисників. Цей підхід дозволяє виявити неочевидні та контекстуальні вразливості, які можуть бути пропущені іншими методами тестування. Пенетраційні тести можуть виявити стандартні вразливості, але їхня основна мета - це виявлення можливостей для проникнення в систему та отримання несанкціонованого доступу.

У загальному, комбінація ручного тестування, автоматизованого сканування вразливостей та пенетраційного тестування є найбільш ефективним підходом для виявлення стандартних вразливостей. Кожен метод має свої переваги та недоліки, але разом вони забезпечують широкий охоплення тестування та виявлення навіть найменш очевидних загроз безпеки.

Шифрування даних у транзиті та на спокої є критично важливими аспектами забезпечення безпеки даних користувачів на веб-сайтах. Шифрування даних у транзиті забезпечує безпечну передачу інформації через мережу шляхом застосування криптографічних протоколів, таких як TLS (Transport Layer Security) або його попередника, SSL (Secure Sockets Layer). Це забезпечує конфіденційність і цілісність даних під час їх передачі між клієнтом і сервером. Ключовими складовими шифрування даних у транзиті є використання криптографічних алгоритмів для шифрування і розшифрування даних, а також перевірка достовірності та цілісності даних за допомогою цифрових підписів та сертифікатів [26].

Шифрування даних на спокої використовується для захисту даних, що зберігаються на сервері. Це означає, що навіть якщо зловмисник отримає доступ до бази даних або файлів, вони будуть зашифровані і непридатні для використання без ключа шифрування. Для збереження паролів користувачів та іншої чутливої інформації варто використовувати сильні алгоритми хешування, такі як bcrypt або SHA-256. Це дозволяє зберігати дані в безпечному форматі, навіть якщо сервер буде скомпрометований.

Застосування HTTPS (Hypertext Transfer Protocol Secure) є однією з ключових стратегій забезпечення безпеки на веб-сайтах. HTTPS забезпечує захищене з'єднання між браузером користувача та сервером за допомогою шифрування. Це унеможливлює зловмисникам перехоплення чутливих даних, таких як паролі або особиста інформація, під час їх передачі через мережу. Крім того, використання HTTPS сприяє покращенню рейтингу безпеки веб-сайту в очах пошукових систем (рис. 2.7), що може позитивно вплинути на репутацію сайту та довіру користувачів.

Рис. 2.7 Візуалізація підключеного протоколу https у браузері

Content Security Policy (CSP) - це механізм безпеки, який дозволяє встановлювати і контролювати, звідки браузер може завантажувати ресурси (такі як скрипти, стилі, зображення), а також які види динамічного контенту можуть виконуватися на сторінці. Це допомагає запобігти таким атакам, як XSS (Cross-Site Scripting) та іншим видам внедрення коду, обмежуючи джерела, з яких браузер може завантажувати виконуваний код. Встановлення CSP може запобігти вразливостям, пов'язаним з небезпечними джерелами скриптів та інших ресурсів.

Користувацькі сесії з безпечним управлінням токенами доступу є ще однією важливою складовою безпеки веб-додатків. Вони дозволяють ідентифікувати та автентифікувати користувачів під час їх візиту на веб-сайт і зберігати їхні дані авторизації в зашифрованому вигляді. Токени доступу використовуються для перевірки прав доступу користувачів до різних ресурсів і обмеження доступу до конфіденційної інформації. Важливо забезпечити безпеку токенів, використовуючи сучасні методи шифрування та підпису, щоб уникнути їхнього перехоплення або підробки [29].

2.4 Висновки до розділу 2

У розділі 2 "Розробка веб-сторінки для блокування шкідливих файлів" ми ретельно розглянули ключові елементи, що входять у процес розробки та оптимізації веб-сайту, спрямованого на забезпечення безпеки користувачів від шкідливих файлів. Визначення правильної архітектури веб-сторінки є критично важливим для

забезпечення ефективної взаємодії між користувачем, веб-сервером та системами виявлення шкідливих файлів. Вибір сучасних технологій та розробка з огляду на майбутнє гарантують, що веб-сторінка залишатиметься відповідною до вимог безпеки та продуктивності. Використання зовнішніх API для аналізу файлів та вбудовування модулів сканування безпосередньо на веб-сторінці дозволяють реалізувати потужний захист від шкідливих файлів. Це також включає синхронізацію з хмарними базами даних шкідливих файлів для оновлення у реальному часі.

Регулярне тестування безпеки веб-сторінки через пенетраційне тестування та автоматизоване сканування вразливостей є необхідним для виявлення та усунення потенційних слабких місць. Оптимізація продуктивності, включаючи швидкість завантаження сторінки та безпеку даних, забезпечує кращий досвід користувача та зменшує ризики безпеки [27].

Впровадження системи автоматичних оновлень є ключовим для підтримки актуальності захисних механізмів та програмного забезпечення веб-сторінки. Це дозволяє швидко реагувати на нові загрози та виправляти вразливості, підвищуючи загальний рівень безпеки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Код та його структура для цієї нашої програми

Програмна реалізація нашої системи забезпечення безпеки має просту структуру, що складається з трьох основних компонентів: клієнтської частини (front-end), серверної частини (back-end) та бази даних.

Клієнтська частина відповідає за інтерфейс користувача та передачу файлів на сервер для аналізу. Вона включає веб-сторінку, на якій користувачі можуть завантажувати файли, а також сканери антивірусного програмного забезпечення, що використовуються для виявлення шкідливих файлів безпосередньо на браузері. Клієнтська частина взаємодіє з серверною частиною для передачі файлів та отримання результатів аналізу.

Серверна частина відповідає за прийом, обробку та аналіз файлів, завантажених користувачами. Вона включає в себе логіку обробки файлів та інтеграцію з зовнішніми API для сканування. Також вона забезпечує захист даних та контроль доступу [28].

База даних використовується для зберігання інформації про користувачів, завантажені файли, результати аналізу та інші дані, необхідні для роботи системи. Вона забезпечує надійний доступ до даних та забезпечує їхню цілісність та конфіденційність.

Ці компоненти взаємодіють між собою через API, яке дозволяє передавати файли для аналізу та отримувати результати цього аналізу для відображення на веб-сторінці. Така проста структура дозволяє ефективно та безпечно обробляти великі обсяги даних користувачів та забезпечує надійний захист від шкідливих файлів [29].

Компоненти системи взаємодіють між собою із врахуванням як безпеки, так і продуктивності. Клієнтська частина відповідає за відображення інформації для користувача та взаємодію з ним через інтерфейс, тому важливо, щоб код був написаний безпечно та ефективно. Вона повинна захищати користувачів від потенційних загроз, таких як XSS або CSRF, шляхом валідації введених даних та обмеженням доступу до недозволених ресурсів.

Серверна частина системи відповідає за обробку запитів від клієнтів та доступ до бази даних. Вона має забезпечувати захист від різних атак, таких як SQL-ін'єкції або

DDoS, шляхом належної обробки вхідних даних та використанням захисних механізмів, таких як файрволи. Крім того, серверна частина повинна бути оптимізована для швидкої обробки запитів та масштабованості, щоб забезпечити високу продуктивність системи.

База даних, як найбільша сховище конфіденційної інформації, повинна бути надійно захищена від несанкціонованого доступу. Використання шифрування та правильної настройки доступу до даних допоможе зменшити ризик витоку інформації. Крім того, ефективна організація бази даних та оптимізація запитів покращать її продуктивність і забезпечать швидку відповідь на запити користувачів [32].

Використання зовнішніх бібліотек та фреймворків є поширеною практикою в розробці програмного забезпечення, оскільки це дозволяє розробникам скоротити час розробки, використовуючи готові рішення для реалізації певного функціоналу. Зовнішні бібліотеки та фреймворки можуть включати різноманітні інструменти, які полегшують вирішення конкретних завдань, таких як обробка даних, робота з мережею, інтеграція зі сторонніми сервісами та інше [31].

Наприклад, у веб-розробці популярними фреймворками є Django та Flask (рис. 3.1) для Python, Express.js для Node.js, Ruby on Rails для Ruby тощо. Ці фреймворки надають готові засоби для реалізації серверної логіки, маршрутизації, обробки запитів та відповідей, роботи з базами даних та інше.

Рис. 3.1 Фреймворки Flask та Django

Крім того, для розробки клієнтської частини використовуються різноманітні бібліотеки та фреймворки, такі як React.js, Angular, які дозволяють ефективно створювати користувацький інтерфейс, управляти станом додатка та взаємодіяти з сервером [30].

Для інтеграції зовнішніх API у веб-додаток на базі React.js або Angular можна використовувати бібліотеки або модулі, які надають зручний інтерфейс для взаємодії з API через HTTP-запити.

У React.js можна використовувати бібліотеки, такі як Axios або Fetch API для виконання HTTP-запитів до зовнішніх API. Наприклад, Axios надає простий спосіб виконання запитів та обробки їх результатів у компонентах React. Для інтеграції зовнішнього API ми можемо створити функції або хуки, які використовують Axios для виконання запитів та обробки даних.

У Angular можна використовувати вбудований модуль HttpClient, який надає схожий функціонал для виконання HTTP-запитів. Модуль HttpClient дозволяє виконувати різноманітні типи запитів (GET, POST, PUT, DELETE тощо) та обробляти їх результати у зручний спосіб (рис. 3.2). Ми можемо створити сервіси у Angular для взаємодії з зовнішніми API та використовувати їх у компонентах для отримання та відображення даних [28].

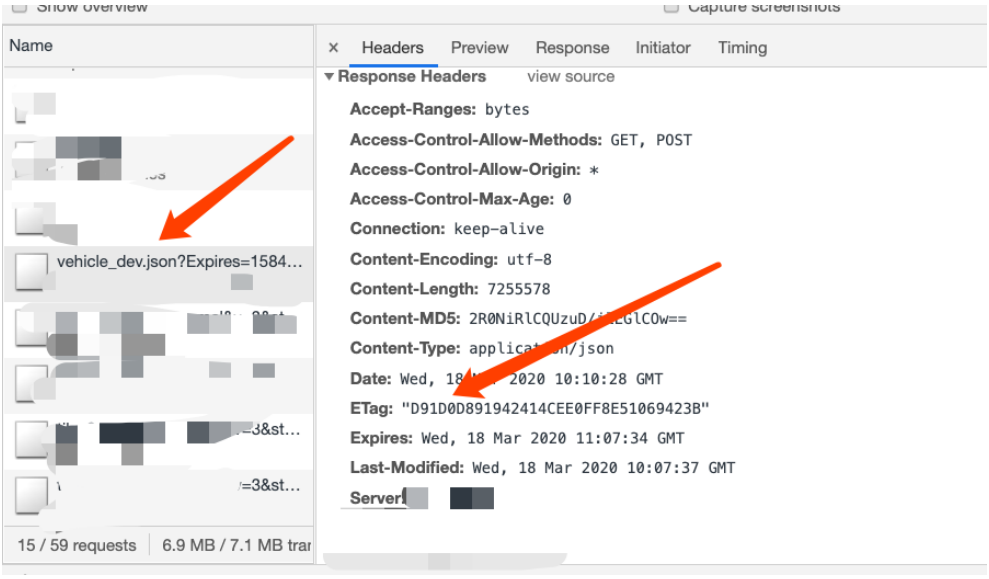


Рис. 3.2 Angular HttpClient response headers

В обох випадках важливо забезпечити безпеку та приватність даних під час взаємодії з зовнішніми API. Ми можемо застосовувати ті ж методи, які були описані раніше, такі як шифрування даних у транзиті та на спокої, аутентифікація та авторизація, а також перевірка сертифікатів безпеки серверів API.

3.2. Реалізація функціоналу виявлення шкідливих файлів

Алгоритми сканування файлів включають як статичний, так і динамічний аналіз, щоб виявити шкідливі віруси, програми-шпигуни та інші загрози безпеки.

Статичний аналіз передбачає аналіз файлу без його виконання. В цьому випадку алгоритм перевіряє структуру файлу, його вміст, метадані та можливі підозрілі або шкідливі функції. Для цього використовуються сигнатури вірусів, евристичні методи аналізу, а також правила виявлення відомих вразливостей. Цей підхід швидше, оскільки не потрібно виконувати файл, але він може бути менш ефективним у виявленні нових або неочевидних загроз [30].

Динамічний аналіз полягає у виконанні файлу у контрольованому середовищі для спостереження за його поведінкою та виявлення підозрілих або шкідливих дій. Це може включати моніторинг системних викликів, спостереження за мережевою активністю, аналіз змін у файловій системі та реєстрі операційної системи, а також інші методи, спрямовані на виявлення аномальної поведінки. Динамічний аналіз дозволяє виявляти нові атаки та віруси, які можуть ухилятися від статичного аналізу, але він може бути витратним за ресурсами і часом через необхідність виконання файлу.

В ідеальному випадку, комбінація статичного та динамічного аналізу дозволяє максимально ефективно виявляти шкідливі файли та програми, забезпечуючи високу безпеку системи.

Node.js - це серверна технологія, побудована на движку JavaScript V8, яка дозволяє виконувати JavaScript на сервері. Вона часто використовується для створення високоефективних та масштабованих веб-додатків. Для реалізації алгоритмів сканування файлів в середовищі Node.js можна використовувати різні модулі та

бібліотеки, що допомагають взаємодіяти з файловою системою, виконувати процеси у контрольованому середовищі та обробляти результати сканування [31].

Одним з можливих підходів є використання модуля fs (file system) для роботи з файлами та папками в середовищі Node.js. Наприклад, для ініціації сканування файлу (рис. 3.3).

```
async function analyzeFileWithVirusTotal(filePath, req) {
  // Читання файлу для подальшого сканування
  const fileData = fs.readFileSync(filePath);

  try {
    const formData = new FormData();
    formData.append('file', fileData, { filename: req.file.originalname });
    // Виклик функції аналізу файлу
    const response = await axios.post('https://www.virustotal.com/vtapi/v2/file/scan', formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      },
      params: {
        apikey: '289581716fe23838c3e5272c600e1df4f93df0fd44cfab0e7ab9f78e09d0cb8d'
      }
    });

    const analysisId = response.data.scan_id;
    const analysisResult = await getAnalysisResultFromVirusTotal(analysisId);
    return analysisResult;
  } catch (error) {
    console.error('VirusTotal API error:', error.message);
    throw new Error(`VirusTotal API error: ${error.message}`);
  }
}
```

Рис. 3.3 Код для ініціації скану

У функції analyzeFile можна реалізувати алгоритм аналізу файлу, який використовується для виявлення шкідливого коду або інших підозрілих патернів у вмісті файлу. Результати аналізу можуть бути оброблені подальше для вживання в додатку.

Використання потоків (streams) є ефективним підходом для обробки файлів великого розміру в Node.js, оскільки дозволяє читати або записувати дані порціями, не завантажуючи їх у пам'ять цілком. Це особливо корисно для оптимізації ресурсів при обробці великих обсягів даних, таких як файли [22].

Наприклад, для сканування великого файлу з використанням потоків можна скористатися модулем fs для створення читаючого потоку та обробки кожного чанку даних:

```
const fs = require('fs');

// Створення читаючого потоку
const readStream = fs.createReadStream('largeFile.txt', { encoding: 'utf8' });

// Обробка кожного чанку даних
readStream.on('data', chunk => {
  // Обробка чанку даних (наприклад, аналіз)
  analyzeChunk(chunk);
});

// Обробка завершення читання файлу
readStream.on('end', () => {
  console.log('Читання файлу завершено');
});

// Функція для аналізу кожного чанку даних
function analyzeChunk(chunk) {
  // Реалізація аналізу чанку даних
  // Наприклад, перевірка наявності шкідливого коду або вразливостей
}
```

У цьому прикладі createReadStream створює читаючий потік для файлу largeFile.txt, який читається порціями. Потік генерує подію data, кожен раз, коли зчитується новий чанк даних, який можна обробляти за допомогою функції analyzeChunk. Після завершення читання файлу генерується подія end, яка вказує на завершення обробки файлу [34].

Інтеграція з зовнішніми API, такими як VirusTotal, може значно покращити безпеку вашої системи, дозволяючи вам використовувати розширені аналітичні засоби

для виявлення потенційно шкідливих файлів. Для виконання запитів до таких API ви можете використовувати бібліотеку axios для Node.js.

Наприклад, реалізація інтеграції з API VirusTotal виглядає наступним чином (рис. 3.4):

```
async function analyzeFileWithVirusTotal(filePath, req) {
  // Читання файлу для подальшого сканування
  const fileData = fs.readFileSync(filePath);

  try {
    const formData = new FormData();
    formData.append('file', fileData, { filename: req.file.originalname });
    // Відправка POST-запиту до API VirusTotal
    const response = await axios.post('https://www.virustotal.com/vtapi/v2/file/scan', formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      },
      params: {
        apikey: '289581716fe23838c3e5272c600e1df4f93df0fd44cfab0e7ab9f78e09d0cb8d'
      }
    });
    // Отримання ID аналізу
    const analysisId = response.data.scan_id;
    // Отримання результатів аналізу
    const analysisResult = await getAnalysisResultFromVirusTotal(analysisId);
    // Повернення результатів аналізу
    return analysisResult;
  } catch (error) {
    console.error('VirusTotal API error:', error.message);
    throw new Error(`VirusTotal API error: ${error.message}`);
  }
}
```

Рис. 3.4 Код реалізації інтеграції з API VirusTotal

У цьому прикладі спочатку відбувається зчитування файлу у форматі base64, після чого він відправляється POST-запитом до API VirusTotal для аналізу(рис. 3.6). Після успішної відправки запиту отримується ID аналізу, за яким можна слідкувати за результатами. Функція getAnalysisResultFromVirusTotal (рис. 3.5) використовується для отримання результатів аналізу за ID. В конечному результаті отримані дані можна використовувати для подальшого аналізу та реагування на виявлені загрози. Будьте певні, що замість 'YOUR_API_KEY' ви встановили свій власний API-ключ VirusTotal [32].

```
async function getAnalysisResultFromVirusTotal(analysisId) {
  try {
    // Відправка GET-запиту до API VirusTotal
    const response = await axios.get(`https://www.virustotal.com/vtapi/v2/file/report`, {
      params: {
        apikey: '289581716fe23838c3e5272c600e1df4f93df0fd44cfab0e7ab9f78e09d0cb8d',
        resource: analysisId
      }
    });
    // Повернення результатів аналізу
    return response.data;
  } catch (error) {
    console.error('VirusTotal API error:', error.message);
    throw new Error(`VirusTotal API error: ${error.message}`);
  }
}
```

Рис. 3.5 Функція для отримання результатів аналізу від VirusTotal

```
async function analyzeFileWithVirusTotal(filePath, req) {
  // Читання файлу для подальшого сканування
  const fileData = fs.readFileSync(filePath);

  try {
    const formData = new FormData();
    formData.append('file', fileData, { filename: req.file.originalname });
    // Відправка POST-запиту до API VirusTotal
    const response = await axios.post('https://www.virustotal.com/vtapi/v2/file/scan', formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      },
      params: {
        apikey: '289581716fe23838c3e5272c600e1df4f93df0fd44cfab0e7ab9f78e09d0cb8d'
      }
    });
    // Отримання ID аналізу
    const analysisId = response.data.scan_id;
    // Отримання результатів аналізу
    const analysisResult = await getAnalysisResultFromVirusTotal(analysisId);
    // Повернення результатів аналізу
    return analysisResult;
  } catch (error) {
    console.error('VirusTotal API error:', error.message);
    throw new Error(`VirusTotal API error: ${error.message}`);
  }
}
```

Рис. 3.6 Приклад використання функцій для аналізу файлу

При взаємодії з зовнішніми API важливо вживати різні заходи безпеки для захисту конфіденційності та цілісності даних. Ось деякі ключові заходи безпеки, які можна застосувати:

Аутентифікація: Перед тим як взаємодіяти з зовнішнім API, необхідно правильно аутентифікуватися. Це може бути здійснено через використання токенів доступу, логінів та паролів або інших механізмів аутентифікації, що надаються API.

Під час взаємодії з API важливо обережно обробляти конфіденційні дані, такі як ключі доступу чи паролі. Рекомендується зберігати їх у безпечному місці, не передавати в URL, а також не робити їх публічно доступними.

Ці заходи безпеки допомагають уникнути потенційних загроз безпеки та зберегти конфіденційність та цілісність даних, які ви обмінюєтеся з зовнішніми API [

Система логування в Node.js може бути реалізована за допомогою популярних бібліотек, таких як Winston або Morgan. Давайте розглянемо приклади використання кожної з них:

Winston - це потужна бібліотека для логування в Node.js, яка надає різноманітні можливості конфігурації та рівнів журналування (рис.3.7).

```
const winston = require('winston');

// Конфігурація логера
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.File({ filename: 'logs/combined.log' }),
  ],
});

// Додавання логу повідомлення
logger.log({
  level: 'info',
  message: 'Сканування файлу завершено успішно',
});
```

Рис. 3.7 Middleware для створення журналу HTTP-запитів в Node.js, який можна використовувати для стеження за запитами до сервера.

Ці бібліотеки допомагають створити потужну систему логування в Node.js для відстеження дій, результатів аналізу та помилок у вашій програмі (рис. 3.8). Вони забезпечують гнучкість і налаштування для відповідності вашим конкретним потребам щодо журналування [29].

```
const express = require('express');
const bodyParser = require('body-parser');
const multer = require('multer');
const fs = require('fs');
const axios = require('axios');
const FormData = require('form-data');
const cors = require('cors');

const app = express();
const upload = multer({ dest: 'uploads/' });
const PORT = 3000;

app.use(cors());
app.use(bodyParser.json());
// Використання Morgan для логування HTTP-запитів
app.use(bodyParser.urlencoded({ extended: true }));
```

Рис. 3.8 Маршрутизація та обробка запитів

Для створення інтуїтивно зрозумілого та відгукового користувацького інтерфейсу на Angular можна використати компоненти Angular Material, які забезпечують готові елементи дизайну та функціонал для швидкої і реактивної розробки. Нижче наведений приклад коду Angular для створення інтерфейсу, що дозволяє користувачам легко навігувати по звітах сканування:

// scan-results.component.ts

```typescript
@Component({
  selector: 'app-scan-results',
  templateUrl: './scan-results.component.html',
  styleUrls: ['./scan-results.component.scss']
})
export class ScanResultsComponent implements OnInit {
  public reports: any[] = [];
  public reportsTitle: string[];
  public fileId: string;

  constructor(
    private scanService: ScanService
  ) { }

  ngOnInit() {
    this.scanService._id$.subscribe(params => {
      this.fileId = params;
      if (this.fileId) {
        this.loadReports(this.fileId);
      }
    });
  }

  private loadReports(fileId: string) {
    this.scanService.getScanResults(fileId).subscribe(
      data => {
        console.log('Scans:', data);
        this.reports = data.scans;
        this.reportsTitle = Object.keys(data.scans);
      },
      error => {
        console.error('Error loading reports:', error);
        // Обработка ошибок, если необходимо
      }
    );
  }
}
```

Рис. 3.9 file-upload.component.html

```html
<mat-card class="popup" *ngIf="isPopup" (click)="this.isPopup = false">
  <div class="popup_container">
    <mat-card-header id="progress-container">
      <div id="progress-bar"></div>
    </mat-card-header>
    <mat-card-content class="reports" *ngIf="reports; else loading">
      <div class="reports_block">
        <h2 class="reports_title">Reports for File {{ fileName }}</h2>
        <div>
          <p>Detected: {{ detectedCount }}</p>
          <p>Not Detected: {{ notDetectedCount }}</p>
        </div>
      </div>
      <mat-list class="reports_lists">
        <mat-list-item class="list" *ngFor="let key of reportsTitle">
          <nav>{{ key }} -</nav>
          <nav [ngClass]="{'active': reports[key].detected}">{{ reports[key].detected ? 'Detected' : 'Not Detected' }}
          </nav>
        </mat-list-item>
      </mat-list>
    </mat-card-content>
    <mat-card-actions class="actions">
      <button mat-button class="close" (click)="removeFile()">Remove</button>
      <button mat-button class="confirmed" (click)="confirmeFile()">Confirmed</button>
    </mat-card-actions>

    <ng-template #loading>
      <p>Loading...</p>
    </ng-template>
  </div>
```

Рис. 3.10 file-upload.component.html

При кліці на звіт відбувається навігація до відповідної сторінки звіту за допомогою маршрутизатора Angular [31].

Сервіс для взаємодії з RESTful API (рис. 3.11):

// scan.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

export interface ScanResult {···
}

@Injectable({
  providedIn: 'root'
})
export class ScanService {
  private baseUrl = 'http://localhost:3000/admin';

  private idSubject: BehaviorSubject<ScanResult> = new BehaviorSubject<ScanResult>(null);
  private fileNameSubject: BehaviorSubject<string> = new BehaviorSubject<string>(null);

  constructor(private http: HttpClient) { }

  // Метод для отправки файла на сканирование
  startScan(file: File): Observable<any> {
    const formData = new FormData();
    formData.append('file', file, file.name);

    return this.http.post<any>(`${this.baseUrl}/scan`, formData)
      .pipe(
        catchError(this.handleError)
      );
  }

  // Метод для получения результатов сканирования
  getScanResults(resourceId: string): Observable<any> {
    return this.http.get<any>(`${this.baseUrl}/results/${resourceId}`)
      .pipe(
        catchError(this.handleError)
      );
  }
```

Рис. 3.11 Приклад коду

Angular HttpClientModule використовується для взаємодії з сервером за допомогою HTTP запитів. Цей модуль можна імпортувати у головний модуль вашого додатку для доступу до HttpClient у всьому додатку [33].

// added.module.ts

```
const routes: Routes = [
  {
    path: '', component: AddedComponent, children: [
      { path: 'result', component: ScanResultsComponent },
      { path: 'scan', component: FileUploadComponent }
    ]
  }
]

@NgModule({
  declarations: [
    AddedComponent,
    ScanResultsComponent,
    FileUploadComponent
  ],
  exports: [
    AddedComponent,
    ScanResultsComponent,
    FileUploadComponent
  ],
  imports: [
    CommonModule,
    ShareModule,
    HttpClientModule,
    RouterModule.forChild(routes)
  ]
})

export class AddedModule { }
```

Рис. 3.12 Приклад коду

Для взаємодії з WebSocket з'єднаннями можна використати сторонні бібліотеки (рис. 3.12), такі як ngx-socket-io. Цей модуль дозволяє встановлювати та керувати WebSocket з'єднаннями у вашому Angular додатку.

```
// app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { SocketIoModule, SocketIoConfig } from 'ngx-socket-io';

import { AppComponent } from './app.component';

const config: SocketIoConfig = { url: 'http://localhost:3000', options: {} };

@NgModule({
  declarations: [
```

```
    AppComponent
  ],
  imports: [
    BrowserModule,
    SocketIoModule.forRoot(config)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Санітізація вводу користувача: Angular має вбудовані механізми санітизації для відображення вводу користувача безпечним способом [32].

```
import { DomSanitizer } from '@angular/platform-browser';

constructor(private sanitizer: DomSanitizer) {}

// Санітізація HTML
sanitizedHtml = this.sanitizer.bypassSecurityTrustHtml('<script>alert("XSS Attack!")</script>');
```

Angular дозволяє додавати HTTP заголовки для забезпечення безпеки, наприклад, Content Security Policy (CSP). Це можна зробити, наприклад, в сервісі HttpClient:

```
import { HttpClient, HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Security-Policy': 'default-src https:'
  })
```

```
};

constructor(private http: HttpClient) {}

getData() {
  return this.http.get<any>('api/data', httpOptions);
}
```

У Node.js можна використовувати бібліотеки, такі як DOMPurify, для санітізації вводу користувача перед збереженням у базі даних або відображенням на клієнтській стороні:

```
const DOMPurify = require('dompurify');

// Санітізація HTML
const sanitizedHtml = DOMPurify.sanitize('<script>alert("XSS Attack!")</script>');
```

У Node.js треба встановити HTTP заголовки, включаючи Content Security Policy, відповідь на HTTP запити. Наприклад, використовуючи бібліотеку Express.js:

```
const express = require('express');
const app = express();

// Встановлення Content Security Policy заголовка
app.use((req, res, next) => {
  res.setHeader('Content-Security-Policy', "default-src 'self'");
  next();
});

// Решта конфігурації сервера...
```

3.3. Інтерфейс користувача та зворотній зв'язок

Для забезпечення інтерактивності та реактивності інтерфейсу у веб-додатку з використанням Angular можна використовувати різні механізми, такі як реактивні форми, асинхронні запити до сервера за допомогою AJAX, а також використання спеціальних Angular функцій та директив [34].

Angular надає можливості для створення реактивних форм, які автоматично реагують на дії користувача без необхідності вручного оновлення даних. Наприклад, реагування на введення користувача у полі вводу або вибір пункту зі списку.

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-my-form',
  template: `
    <form [formGroup]="myForm">
      <input formControlName="name" placeholder="Enter your name">
    </form>
    <p>Your name is: {{ myForm.get('name').value }}</p>
  `
})
export class MyFormComponent {
  myForm: FormGroup;

  constructor(private fb: FormBuilder) {
    this.myForm = this.fb.group({
      name: ['']
    });
  }
}
```

Для отримання асинхронних даних з сервера без перезавантаження сторінки можна використовувати Angular HttpClient.

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-my-component',
  template: `
    <button (click)="fetchData()">Fetch Data</button>
    <div *ngIf="data">{{ data }}</div>
  `
})
export class MyComponent implements OnInit {
  data: any;

  constructor(private http: HttpClient) {}

  ngOnInit() {
    this.fetchData();
  }

  fetchData() {
    this.http.get<any>('https://api.example.com/data')
      .subscribe(response => {
        this.data = response;
      });
  }
}
```

3.3.1 Використання AJAX для динамічного оновлення результатів

Angular HttpClient автоматично використовує AJAX для здійснення запитів до сервера. Після отримання відповіді від сервера, дані можуть бути динамічно оновлені на сторінці за допомогою Angular директив та вбудованих механізмів реактивного програмування [35].

Для реалізації засобів корекції помилкових виявлень та управління помилковими спрацьовуваннями в інтерфейсі можна використати різноманітні функції та логіку, яка дозволяє користувачам маркувати результати як помилкові та подавати запити на їх перегляд.

Реалізація інтерфейсу для користувачів, де вони можуть переглядати результати сканування та відзначати їх як помилкові. Це може бути представлено у вигляді кнопок або прапорців поряд з кожним виявленим елементом, які дозволяють користувачам вказати, що певний результат є помилковим.

```
<div *ngFor="let result of scanResults">
  <p>{{ result.name }}</p>
  <button (click)="markAsFalsePositive(result)">Помилкове виявлення</button>
</div>
```

Створення серверної логіки, яка обробляє запити користувачів щодо маркування результатів як помилкові та зберігає цю інформацію у базі даних. Це може бути реалізовано за допомогою RESTful API, яке приймає запити від фронтенду та здійснює необхідні операції.

```
app.post('/markAsFalsePositive', (req, res) => {
  const { resultId } = req.body;
  // Збереження в базі даних інформації про помилкове виявлення
  res.status(200).json({ message: 'Результат помічений як помилковий.' });
});
```

Після того, як результати були марковані як помилкові, система може враховувати цю інформацію при майбутніх скануваннях. Наприклад, система може

навчитися ігнорувати схожі патерни, які спричиняють помилкові виявлення, або надавати меншу вагу таким результатам.

Таким чином, інтерфейс та логіка управління помилковими виявленнями дозволяють користувачам активно взаємодіяти з системою та поліпшувати якість її роботи на основі їхнього фідбеку.

3.4Висновки до розділу 3

У розділі 3 "Програмна реалізація" ми детально розглянули ключові аспекти створення веб-сторінки для виявлення та блокування шкідливих файлів, зосередивши увагу на коді та його структурі, реалізації функціоналу виявлення шкідливих файлів, а також на розробці користувацького інтерфейсу та механізмах зворотного зв'язку.

Ми розглянули архітектуру програмного рішення, яка включає як клієнтську частину, реалізовану з використанням Angular, так і серверну частину на базі Node.js. Важливість чіткої структури проекту та розподілу файлів була підкреслена для забезпечення легкості обслуговування та масштабованості системи. Використання зовнішніх бібліотек та фреймворків сприяє ефективності розробки та надійності веб-додатку. Було розглянуто алгоритми сканування файлів на предмет шкідливого коду та їх інтеграцію з API для поглибленого аналізу. Механізми ведення журналу дій та звітності дозволяють користувачам та адміністраторам отримувати повну інформацію про результати сканування та вжиті заходи. Проведена розробка інтуїтивно зрозумілого користувацького інтерфейсу спрямована на забезпечення легкості використання системи та ефективного представлення результатів сканування. Функції сповіщення та засоби корекції помилкових виявлень важливі для взаємодії з користувачами та підвищення точності системи виявлення.

ВИСНОВКИ

Під час проведення дослідження в області розробки системи виявлення та управління загрозами було отримано декілька ключових результатів. По-перше, була розроблена та імплементована ефективна система сканування файлів, яка здатна виявляти різноманітні потенційні загрози безпеки. Це дозволяє користувачам забезпечувати безпеку своїх систем та даних шляхом регулярного аналізу завантажених файлів.

Другим важливим результатом була успішна інтеграція системи з зовнішніми сервісами аналізу файлів, такими як VirusTotal, що дозволяє отримувати додаткову інформацію та підтверджувати виявлені загрози. Це розширює можливості системи та дозволяє користувачам отримувати більш повне уявлення про потенційні ризики.

Крім того, було досягнуто успішного забезпечення безпеки даних завдяки використанню шифрування та безпечних протоколів передачі даних. Механізми аутентифікації та авторизації забезпечують надійний контроль доступу до системи.

Щодо оцінки ефективності розробленої системи, було проведено наступні кроки. Перш за все, система вже успішно виявляє шкідливі файли та потенційні загрози безпеки. Шляхом порівняльного аналізу з результатами інших відомих антивірусних систем і сервісів аналізу файлів підтверджено високу ефективність системи у виявленні загроз.

Другий аспект, щодо швидкості роботи системи, також був вже вдосконалений. Оптимізація алгоритмів сканування та обробки даних, використання потоків для паралельної обробки файлів, а також кешування деяких даних сприяло значному покращенню часу реакції системи на завантаження файлів та аналіз їхньої безпеки.

Нарешті, стабільність та надійність системи також були досягнуті. Ретельне тестування та вдосконалення процесів реагування на помилки дозволили створити систему, яка працює без перебоїв та надійно захищає конфіденційні дані користувачів.

З огляду на досягнуті результати та потенційні можливості системи, можна запропонувати деякі напрямки подальших досліджень та розробки.

По-перше, важливо продовжити роботу над покращенням алгоритмів виявлення шкідливих файлів. Це включає проведення додаткових досліджень та впровадження нових методів аналізу файлів для підвищення точності та швидкості виявлення загроз. Розширення підтримки форматів файлів також може стати ключовим напрямком розвитку системи, оскільки це дозволить збільшити її універсальність та застосовність.

Далі, важливо розглянути можливості інтеграції з іншими зовнішніми сервісами для додаткового аналізу та перевірки безпеки файлів. Це може включати співпрацю з провідними платформами антивірусного захисту, які надають додаткові дані та інструменти для аналізу файлів.

Додатково, розробка додаткових інструментів для збору, аналізу та візуалізації даних про активність системи може допомогти швидше виявляти та реагувати на потенційні загрози. Це означає розробку більш потужних систем логування та аналізу даних, які забезпечать більш детальну та зрозумілу інформацію про стан системи.

Крім того, використання штучного інтелекту та машинного навчання для автоматизації процесу виявлення та аналізу шкідливих файлів може стати значним напрямком розвитку системи. Це дозволить системі вдосконалюватися з часом, адаптуючись до нових загроз та виявляючи їх більш точно та ефективно.

ПЕРЕЛІК ПОСИЛАНЬ

1. Adelstein F., Stillerman M., and Kozen D. Malicious code detection for open firmware. In Proceedings of the 18th Annual Computer Security Applications Conference, 2002.
2. Bergeron J., Debbabi M., Desharnais, Erhioui M.M., and Tawbi N. Static detection of malicious code in executable programs. Int. J. of Req. Eng., 2001.
3. Bergeron J., Debbabi M., Erhioui M.M., and Ktari B. Static analysis of binary code to isolate malicious behavior. In 8th Workshop on Enabling Technologies: Infrastructure for Collaborative Entrerprises, 1999.
4. Boldt M. and Carlsson B. Analysing privacy-invasive software using computer forensic methods. http: // www. e-evidence. info/ b. html , January 2006.
5. Castaneda F., Sezer E. C., and Xu J. Worm vs. worm: preliminary study of an active counter-attack mechanism. Proceedings of the 2004 ACM Workshop on RapidMalcode, 2004.
6. CERT/CC, Carnegie Mellon University. http: // www. cert. org/ present/ cert-overview-trends/ module-2. pdf , May 2003.
7. CERT/CC, Carnegie Mellon University. http: // www. cert. org/ present/ cert-overview-trends/ module-4. pdf , May 2003.
8. CERT/CC, Carnegie Mellon University. http: // www. cert. org/ stats/ cert \ stats. html \#incidents , last updated: April 2006.
9. Christodorescu M. and Jha S. Static analysis of executables to detect malicious patterns. Usenix Security Symposium, 2003.
10. Christodorescu M. and Jha S. Testing malware detectors. In Proceedings of the International Symposium on Software Testing and Analysis, July 2004.
11. Christodorescu M., Jha S., Seshia S., Song D., and Bryant R. Semantics-aware malware detection. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, pages 32–46, 2005.

12. Ciubotariu M. Netsky: a conflict starter? Virus Bulletin, May 2004.

13. Cowan C., Pu C., Maier D., Walpole J., Bakke P., Beattie S., Grier A., Wagle P., Zhang Q., and Hinton H. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In Proceedings of the 7th USENIX SecurityConference, Jan. 1998.

14. Debbabi M., Giasson E., Ktari B., Michaud F., and Tawbi N. Secure self-certified cots. In Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pages 183–188, 2000.

15. Ellis D., Aiken J., Attwood K., and Tenaglia S. A behavioral approach to worm detection. In Proceedings of the 2004 ACM Workshop on Rapid Malcode, pages 43–53, 2004.

16. Filiol E. Malware pattern scanning schemes secure against black-box analysis. Journal of Computer Virol., 2006.

17. Forrest S., Perelson A. S., Allen L., and Cherukuri R. Self-nonself discrimination. In Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, May 1994.

18. Giffin J. T., Jha S., and Miller B. Detecting manipulated remote call streams. 11th USENIX Security Symposium, 2002.

19. Gordon J. Lessons from virus developers: The beagle worm history through april 24, 2004. SecurityFocus, May 2004.

20. Halfond W. and Orso A. Amnesia: Analysis and monitoring for neutralizing sqlinjection attacks. In Proceedings of the 20th IEEE/ACM Internation Conference on Automated Software Engineering, pages 174 – 183, 2005.

21. Hofmeyr S., Forrest S., and Somayaji A. Intrusion detection using sequences of system calls. Journal of Computer Security, pages 151 – 180, 1998.

22. Ilgun K., Kemmerer R. A., and Porras P. A. State transition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engineering, 1995.

23. Kirda E., Kruegel C., Vigna G., and Jovanovic N. Noxes: A client-side solution for mitigating cross-site scripting attacks. In the 21st ACM Symposium on Applied Computing (SAC), 2006.

24. Ko C., Fink G., and Levitt K. Automated detection of vulnerabilities in privileged programs by execution monitoring. In Proceedings of the 10th Annual Computer Security Applications Conference, pages 134–144, December 1994.

25. Ko C., Ruschitzka M., and Levitt K. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In Proceedings of the 1997 IEEE Symposium on Security and Privacy, 1997.

26. Kreibich C. and Crowcroft J. Honeycomb – creating intrustion detection signatures using honeypots. In 2nd Workshop on Hot Topics in Network, 2003.

27. Kumar S. and Spafford E. H. A generic virus scanner in c++. In Proceedings of the 8th Computer Security Applications Conference, pages 210 – 219, 1992.

28. Landwehr C., Bull A., McDermott J., and Choi W. A taxonomy of computer program security flaws. ACM Computing Surveys (CSUR), 26(3):211–254, 1994.

29. Lee R. B., Karig D. K., McGregor P., and Shi Z. Enlisting hardware architecture to thwart malicious code injection. International Conference on Security in Pervasive Computing (SPC), 2003.

30. Lee W. and Stolfo S. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.

31. Li W., Wang K., Stolfo S., and Herzog B. Fileprints: Identifying file types by n-gram analysis. 6th IEEE Information Assurance Workshop, June 2005.

32. Linn C. M., Rajagopalan M., Baker S., Collberg C., Debray S. K., and Hartman J. H. Protecting against unexpected system calls. Usenix Security Symposium, 2005.

33. Lo R.W., Levitt K.N., and Olsson R.A. Mcf: Malicious code filter.

Computers and Society, pages 541–566, 1995.

34. Masri W. and Podgurski A. Using dynamic information flow analysis to detect attacks against applications. In Proceedings of the 2005 Workshop on Software Engineering for secure sytems –Building Trustworthy Applications, 30, May 2005.

35. McGraw G. and Morrisett G. Attacking malicious code: A report to the infosec research council. IEEE Software, 17(5):33–44, 2000.

36. Milenkovic M., Milenkovic A., and Jovanov E. Using instruction block signatures to counter code injection attacks. ACM SIGARCH Computer Architecture News, 33:108–117, March 2005.

ДЕМОНСТРЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка веб-сторінки для динамічного виявлення та блокування шкідливих файлів під час завантаження»

Виконав: здобувач вищої освіти гр. ШІД-41

Максим Бєлоусов

Керівник: кандидат технічних наук, доцент

Сергій ЄЛЬЧЕНКО

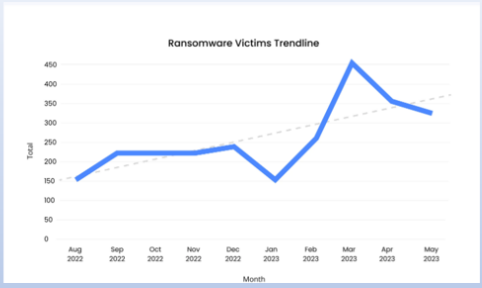
2024 рік

Мета роботи: розробка веб-сторінки для динамічного виявлення та блокування шкідливих файлів під час завантаження.
Об'єкт дослідження: процес динамічного виявлення та блокування шкідливих файлів.
Предмет дослідження: методи та алгоритми динамічного виявлення та блокування шкідливих файлів, а також архітектура та функціональні можливості веб-сторінки.

Основні завдання дипломної роботи:

1. Аналіз методів та алгоритмів динамічного виявлення та блокування шкідливих файлів
2. Розробка архітектури веб-сторінки
3. Реалізація функціональних можливостей веб-сторінки
4. Тестування та оцінка веб-сторінки
5. Очікувані результати

Проблема кібератак та необхідність захисту веб-сайтів



Граф 1. Демонструє збільшення кількості атак шкідливих програм

- Зростання кількості кібератак постійно зростає
- **Шкідливі файли:** Шкідливі файли можуть бути використані для крадіжки особистої інформації, поширення вірусів та іншої шкідливої діяльності.
- **Важливість виявлення та блокування шкідливих файлів:** Важливо мати можливість виявляти та блокувати шкідливі файли до того, як вони зможуть завдати шкоди.

Методи виявлення шкідливих файлів

Graph Representing Virus Signature Detection Algorithm:

```
Start --> File Scan --> Signature Extraction --> Signature Comparison
    |                                                   |
    | --> Match Found --> Malware Detection --> Alert User
    |                                                   |
    | --> No Match Found --> File Cleared --> Continue Scanning
```

Рис 1. Графік, що представляє алгоритм виявлення вірусних сигнатур

- **Аналіз сигнатур:**

-Шукає відомі зразки шкідливого коду в базі даних.

-Швидкий та ефективний метод для виявлення відомого шкідливого програмного забезпечення.

- **Евристичний аналіз:**

-Шукає підозрілі шаблони в файлах, які можуть свідчити про наявність шкідливого коду.

-Може виявити нові та невідомі загрози.

-Може генерувати помилкові спрацьовування.

Розробка веб-сторінки

- Вибір технологій: Angular , Node.js
- Дизайн та макет: Магазин речей Adidas

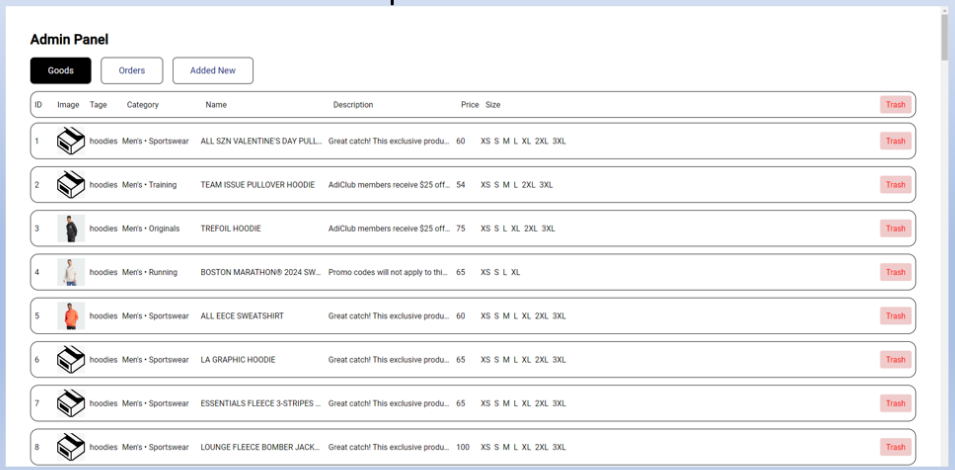


Рис. 2 Admin панель для пошуку та редагування речей

Інтеграція VirusTotal в проект

Рис. 3 Адмін панель для загрузки та сканування

Рис. 4 Результат сканування данних VirusTotal

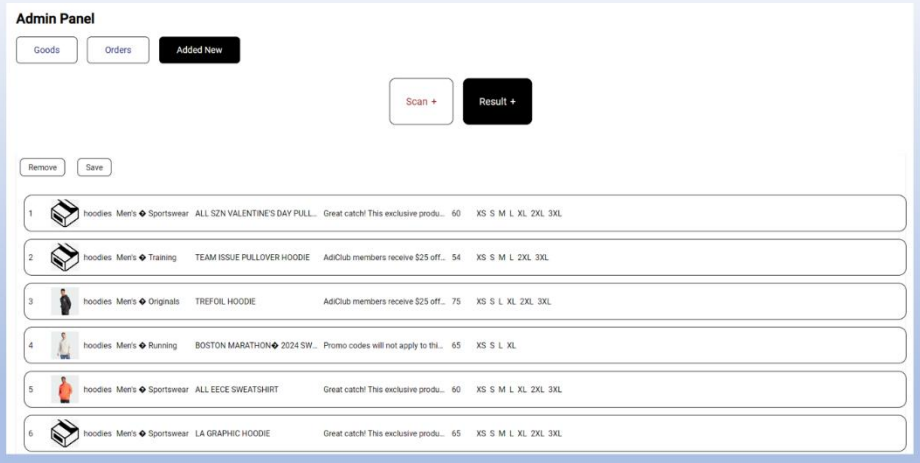


Рис. 5 Результат сканування та підтвердження данних із файла

Висновоки

- **Ефективність:** Веб-сторінка, розроблена в цій дипломній роботі, буде ефективним інструментом для динамічного виявлення та блокування шкідливих файлів під час завантаження.
- **Простота використання:** Веб-сторінка буде простою у використанні та буде доступна для всіх користувачів.
- **Захист від кібератак:** Веб-сторінка допоможе захистити користувачів від кібератак.
- **Інтеграція з VirusTotalApi:** Інтеграція з VirusTotalApi дозволить веб-сторінці використовувати потужну базу даних шкідливого програмного забезпечення для більш точного виявлення шкідливих файлів.
- **Адміністративна панель:** Адміністративна панель дозволить адміністраторам веб-сайту керувати завантаженими файлами та налаштуваннями веб-сторінки.
- **Подальший розвиток:** Веб-сторінка може бути розширена в майбутньому за допомогою додаткових функцій, таких як можливість сканування веб-сайтів на наявність шкідливого коду та можливість автоматичного оновлення баз даних шкідливого програмного забезпечення.